UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

IMPLEMENTACIÓN DEL ATAQUE SINKHOLE EN EL PROTOCOLO RPL

TESIS

Que para obtener el título de

Ingeniero en Telecomunicaciones

P R E S E N T A

Emanuel Ramírez Romero

DIRECTOR DE TESIS

Dr. Luis Francisco García Jiménez

Ciudad Universitaria, Cd. Mx., 2020

Agradecimientos

Agradezco a mis padres quienes con su apoyo, esfuerzos y cariño incondicional, he podido llegar a la culminación de esta etapa en mi formación académica.

Al Dr. Luis Francisco García Jiménez por su dirección, enseñanza y apoyo en la realización de esta tesis.

Por otro lado agradezco el apoyo brindado por parte del proyecto DGAPA-PAPIIT IA105520

Índice general

Índice de figuras

Índice de pseudocódigos

Índice de códigos

Resumen

Una de las tareas principales dentro de las redes IoT, es extender la conectividad de los objetos de uso cotidiano como son sensores, computadoras, televisiones, lavadoras, con el fin de que estos objetos puedan intercambiar información y tomar decisiones que automaticen tareas de la vida cotidiana. Sin embargo, muchos de estos objetos operan con recursos limitados de procesamiento, memoria y batería. Por ello, en los últimos años se han estudiado y propuesto protocolos de comunicación que puedan funcionar correctamente en este tipo de sistemas. El estándar más utilizado para el encaminamiento de paquetes en sistemas con recursos limitados, es conocido como *routing protocol for low-power and lossy networks* (RPL), el cual basa su funcionamiento en la jerarquización de los nodos que conforman la red, es decir, todas las rutas encaminan los paquetes hacía un nodo principal que se denomina nodo raíz, encargado de recibir toda la información y enviar esta información a Internet o a algún servidor.

Uno de los aspectos que se ha descuidado en este protocolo ha sido la seguridad, la cual de acuerdo al estándar RFC6550 se basa en la generación y procesamiento de mensajes, los cuales se encargan de asegurar la versión de los mensajes de control de la red RPL. Sin embargo, este protocolo está expuesto a una gran variedad de ataques, cuyas consecuencias pueden deteriorar el rendimiento y los recursos de la red.

Algunos de los ataques más conocidos en RPL son *black* y *gray hole*, en los cuales un nodo malicioso captura el tráfico que pasa por él con el fin de descartarlo y que no llegue a su destino. Por otro lado, uno de los ataques más dañinos a la estructura de la red es el ataque *sinkhole*, en el cual un nodo malicioso se encarga de establecer nuevas rutas mediante la modificación de su rango. De esta forma los nodos cercanos a él creen haber encontrado una mejor ruta y estos envían su tráfico a través del nodo malicioso. Aunque existen propuestas de detección del ataque sinkhole como el uso de detección de intrusos (IDS), muchas de estas estrategias solo se han simulado a través de software como Cooja. Por lo que en esta tesis, se propone una implementación real tanto del protocolo RPL, así como del ataque sinkhole para medir cómo este ataque afecta a la estructura de la red. Más aún, esta implementación servirá como base para estudiar otros ataques y a su vez como base para proponer contra medidas en el protocolo RPL. También, ayudará a comprender de mejor forma los factores que no suelen estar programados en un simulador y cómo éstos afectan a los protocolos. Para ello, se implementa una red ad-hoc conformada por un conjunto de Raspberries Pi modelo 3B, las cuales permiten registrar y analizar las vulnerabilidades del protocolo RPL ante un ataque sinkhole. Con los resultados obtenidos se da una demostración clara del desvío de los paquetes y cómo este ataque afecta de manera trascendental la topología de la red dependiendo de la ubicación del nodo malicioso.

Capítulo 1

Introducción

La búsqueda por conectar dispositivos de uso cotidiano, capaces de medir variables físicas cuya información registrada esté siempre a nuestro alcance, ha potencializado el paradigama de IoT. Las redes IoT se caracterizan por permitir la conexión entre una gran variedad de dispositivos que operan con diferentes tipos de hardware, donde muchos de estos dispositivos tienen la característica de operar con recursos limitados en cuanto a procesamiento, memoria y batería.

El protocolo más utilizado que cumple con los requerimientos para mantener la conectividad entre los dispositivos IoT es conocido como *routing protocol for low-power and lossy networks* (RPL). Este protocolo se basa en la jerarquización de los nodos mediante un parámetro denominado rango. Dentro de la red se define un nodo principal denominado nodo raíz. El rango representa la posición relativa con respecto a los otros nodos y al nodo raíz, es decir, el rango de cada nodo estrictamente decrece cuando se acerca al nodo raíz y se incrementa cuando se aleja del nodo raíz.

De acuerdo al RFC 6550 [1], se emplean tres mensajes para la creación de rutas en el protocolo RPL (DIO, DAS y DIS). El primero de ellos se denomina mensaje DIO, el cual contiene la información necesaria para construir una instancia de RPL. Cada vez que un nodo recibe un mensaje DIO configura sus parámetros, selecciona a su padre y determina su rango mediante la función objetivo referida en el mensaje DIO. El valor del rango del nodo corresponde a su posición en el grafo con respecto al nodo raíz y siempre debe ser más grande que el rango de sus padres. Posteriormente el nodo envía este mensaje DIO a sus vecinos. Al final de este proceso, todos aquellos nodos que intercambiaron el mensaje DIO tienen un camino para llegar al nodo raíz mediante la elección de los padres de cada nodo. El mensaje DAO es usado para construir las rutas del nodo raíz hacia las hojas. Los padres añaden las rutas recibidas por otros mensajes DAO y las envían recursivamente a sus padres mediantes mensajes DAO. En otras palabras, se encarga de propagar la información de los diferentes destinos que se encuentran desde el nodo raíz hacia las hojas del árbol. Finalmente, cada vez que un nuevo nodo quiere incorporarse a la red, debe enviar un mensaje DIS con el fin de solicitar un mensaje DIO de sus vecinos.

Dado que la construcción de rutas depende solo de los mensajes anteriormente mencionados, este protocolo puede ser vulnerable a ataques y de esta forma comprometer la operatividad de la red. Entre los ataques más significativos que podemos encontrar está el ataque sinkhole, el cual advierte un camino beneficioso para atraer el tráfico cercano de sus vecinos, el ataque wormhole el cual tiene como objetivo interrumpir la topología de la red y el flujo de tráfico, mediante la creación de un canal de comunicación entre dos atacantes que transmiten todo el tráfico que pasa a través de dicho canal, el ataque de clonación de ID, en el cual el atacante clona la identidad de otro nodo para ganar acceso al tráfico destinado al nodo. También, existen otros ataques que se enfocan en dañar diferentes características de la operación de la red y que operan en diferentes capas del modelo OSI.

Específicamente, en esta tesis se analiza los daños que puede ocasionar el ataque sinkhole dentro de una red IoT que opera con el protocolo RPL. Para ello a diferencia de la mayoría de los trabajos relacionados donde solo simulan el protocolo y los diferentes

ataques, esta tesis implementa el protocolo RPL y el ataque sinkhole mediante el uso de Raspeberries con el fin de analizar cuan vulnerable es una red IoT ante un ataque sinkhole. Más aún, esta implementación servirá como base para estudios de otros ataques en escenarios reales y a su vez para el análisis de contra medidas.

1.1. Definición del problema

El protocolo RPL emplea el rango para jerarquizar a los nodos en el árbol. Gracias a este rango y a la selección de padres, los nodos que pertenecen a la red pueden establecer rutas de encaminamiento con el fin de que toda la información generada en los nodos viaje hacia el nodo raíz, el cual suele estar conectado a una red externa con acceso a Internet o algún servidor en la nube. Sin embargo, un nodo malicioso puede auto asignarse un rango lo más bajo posible con el fin de atraer la mayor cantidad de paquetes a sí mismo, con el fin de analizar la información y no retransmitirla a ningún otro nodo; este modo de operación se conoce como ataque sinkhole. Una vez que el ataque sinkhole es efectivo, la integridad de la información que se transmite a lo largo de la red puede verse comprometida, ya que esta puede quedar expuesta ante al atacante.

El estándar RPL (definido en el RFC 6550[1]) emplea mecanismos de seguridad, sin embargo, la complejidad y el tamaño de los mensajes pueden representar el uso de muchos recursos en comparación con los que cuentan los dispositivos usados comúnmente en las redes IoT.

1.2. Hipótesis

El cambio de rango y la posición estratégica de un nodo malicioso permiten establecer un ataque sinkhole y vulnerar la red al atraer la mayor parte del tráfico hacía el nodo malicioso.

1.3. Meta General

Analizar y cuantificar los daños que el ataque sinkhole puede ocasionar en una red que emplea el protocolo RPL mediante el diseño de una red ad hoc con Raspberries y un sensor DTH11 que sea capaz de transmitir valores de temperatura y humedad.

1.4. Metodología

Para alcanzar la meta de este trabajo se divide en las siguientes tres etapas.

- Analizar detalladamente la operación del protocolo RPL, mediante el análisis de varios artículos de investigación y el RFC 6550[1], lo cual permite comprender el protocolo para posteriormente implementarlo.
- Crear una topología árbol que ejecute el protocolo RPL mediante el lenguaje de programación Python, la cual contiene un sensor de temperatura y humedad DTH11 que transmite datos al nodo raíz.
- Implementar el ataque Sinkhole en el lenguaje Python con el fin de atacar a la red, y visualizar el ataque mediante la herramienta Wireshark.

1.5. Contribución

Entre los diferentes artículos relacionados con el protocolo RPL y sus vulnerabilidades en seguridad, se dan extensas explicaciones sobre la forma en la que opera el protocolo y las características del ataque sinkhole. También se muestran resultados de simulaciones realizadas en el simulador Cooja de Conitiki. Sin embargo, la

contribución principal de esta tesis es demostrar de una manera física los daños que un ataque Sinkhole puede producir en una red IoT que opera con el protocolo RPL y mostrar procesos que en un simulador no se pueden crear, cómo localizar el punto óptimo para realizar el ataque, ya que en el mundo físico los patrones de radiación de las antenas cambian constantemente.

1.6. Descripción del contenido

- En el capítulo 2 se presentan los trabajos relacionados con el funcionamiento del protocolo RPL, también se muestran los mensajes de control empleados en el protocolo RPL, así como los campos usados para la implementación de la red. También se hace una simulación del protocolo RPL en el simulador cooja, y se presenta el pseudocódigo de los tres tipos de nodos que se emplearon en la red, así como el diagrama de conexiones empleado para la medición de humedad y temperatura con el sensor DHT11.
- En el capítulo 3 se explica el atacante sinkhole y las principales técnicas para contrarrestar el ataque sinkhole, dando una explicación de las ventajas y desventajas que estas pueden presentar.
- En el capítulo 4 se dan los detalles de la topología de red que se implementó y se demuestran los resultados obtenidos mediante el flujo de tráfico con wireshark.
- En el capítulo 5 se presentan las conclusiones generales y la verificación de la hipótesis.

Capítulo 2

RPL

2.1. RPL

El estándar de encaminamiento para las *low power and lossy networks* (LLN) es conocido como RPL [1]. Este protocolo emplea técnicas que optimizan la comunicación entre los dispositivos. Uno de los aspectos más relevantes de este protocolo, es que todos los caminos se dirigen a un punto denominado nodo raíz, el cual actúa como frontera y conecta la LLN con Internet o alguna otra red.

En [2], los autores definen la operación del protocolo RPL, el cual conecta a los dispositivos mediante la creación de canales de comunicación acíclicos denominados *destination oriented acyclic graph* (DODAG), estos canales transmiten la información de manera unidireccional hasta llegar al nodo raíz, cada nodo que pertenece a la red es identificado mediante su dirección IP. Cada nodo tiene un rango asignado, el cual indica la posición relativa que tiene con respecto a los nodos vecinos y al DODAG raíz. El rango estrictamente se decrementa en los nodos que se acercan al DODAG raíz y se incrementa en los nodos que se alejan del nodo DODAG raíz.

Inicialmente, el nodo raíz es el único nodo activo en el DODAG, este nodo difunde mensajes de control denominados *DODAG information object* (DIO), con el fin de configurar los parámetros que ayudan a construir y mantener la topología de la red. Entre los parámetros se encuentran el rango del nodo, el DODAG ID, la instancia ID, el número de versión, el modo de operación (MOP), entre otros. Una vez que un nodo recibe varios mensajes DIO de diferentes fuentes, éste selecciona a una de estas fuentes como su padre, el cual será el siguiente salto para llegar al nodo raíz. La forma en la que los nodos seleccionan a su padre se define a partir de la función objetivo (OF), esta función toma como entrada diferentes métricas como son latencia, potencia de transmisión o confiabilidad del enlace para transformarlas en un rango.

A pesar de que el el calculo del rango se deja como tarea a la función objetivo, el rango debe implementar propiedades genéricas independientemente de la función objetivo. Como característica particular el rango de los nodos debe ir decreciendo mientras los paquetes siguen su destino, por lo que se considera el rango una representación escalar de la ubicación o radio de un nodo dentro de un DODAG. El rango no es un costo de ruta, sin embargo su valor puede ser derivado e influir en las métricas de ruta. El rango tiene las siguientes propiedades:

- Tipo: Es un valor numérico abstracto
- Función: Es la expresión de una posición relativa dentro del DODAG con respecto a sus vecinos y no necesariamente es un indicador o una expresión de la distancia o costo de camino a la raíz.
- Estabilidad: La estabilidad del rango determina la estabilidad de la topología.
- Propiedades: El rango se incrementa de una manera estrictamente monótona y se puede utilizar para validar una progresión desde o hacia la raíz.
- Resumen: El rango no tiene una unidad física, sino un rango de incremento por salto, donde la asignación de cada incremento será determinada por la Función Objetivo.

Dentro del protocolo RPL, para la formación del DODAG, se definen otros mensajes de control entre los cuales se encuentra el mensaje *destination advertisement object* (DAO). Este permite a un nodo enviar su prefijo para establecer rutas de bajada. También está el mensaje *DODAG solicitation object* (DIS). Este mensaje se emplea para solicitar información acerca de los parámetros de configuración.

A continuación se detallan los elementos que componen al protocolo RPL bajo las especificaciones del RFC6550.

2.2. Mensajes de control de RPL

RPL cuenta con tres mensajes de control tanto para formar las rutas hacia el nodo raíz, así como para el mantenimiento de la red.

- DODAG information solicitation (DIS).
- DODAG information object (DIO).
- Destination advertisement object (DAO).

2.2.1. Mensaje DIS

El mensaje DIS solicita un mensaje DIO con el fin de que el nodo conozca el modo de operación de RPL, así como para sondear su vecindario en busca de nuevos nodos cercanos. De acuerdo con el RCF6550[1], la estructura del mensajes DIS consta de tres campos:

- **Flags:** Campo de 8 bits que se reserva para banderas.
- **Reserved:** Campo de 8 bits que no se utiliza.
- **Options:** Campo de 8 bits para mandar solicitudes de información, con el fin de que cualquier nodo que quiera formar parte de la red reciba la información necesaria para poder transmitir y tener conocimiento acerca de quienes son sus vecinos más cercanos. Entre las opciones se encuentran Pad1, PadN y Solicited Information. Para propósitos de esta tesis solo se emplea el campo Options.

Figura 2.1: Campos del mensaje DIS de acuerdo al RFC6550[1].

2.2.2. Mensaje DIO

El mensaje DIO contiene la información necesaria para la construcción y configuración de la red RPL, mediante los parámetros en el mensaje DIO, los nodos pueden elegir a su padre y mantenerse como un miembro activo de la red. El mensaje DIO consta de los siguientes campos:

- **RPLInstanceID:** Campo de 8 bits que indica a cuál instancia RPL pertenece el DODAG.
- **Version Number:** Campo de 8 bits que establece la versión del DODAG.
- **Rank:** Campo de 16 bits que indica el rango del nodo.
- **Grounded (G):** Indica si el DODAG anunciado puede satisfacer la aplicación definida como meta.
- **Mode of operation (MOP):** Identifica el modo de operación de la red RPL.
- **DODAGPrefrence (Prf):** Campo de 3 bits que define cómo el nodo raíz de la red es comparado con otros nodos raíces.

- **Destination advertisement trigger sequence number (DTSN):** Campo de 8 bits que se emplea para mantener las rutas de bajada.
- **Flags:** Campo de 8 bits reservado para banderas.
- **Reserved:** Campo de 8 bits que no se utiliza.
- **DODAGID:** Campo de 128 bits con la dirección IPv6 establecida por el nodo raíz.
- **Options:** Contiene las opciones válidas que pueden ser Pad1, PadN, DAG Metric Container, Routing Information, DODAG Configuration y Prefix Information.

Para fines de esta tesis se consideran los campos Options y Rank, ya que los parámetros G, MOP, Prf, version, RPLInstanceID y DODAGID son iguales para todos aquellos nodos que no son el nodo raíz, excepto el rango y el DTSN.

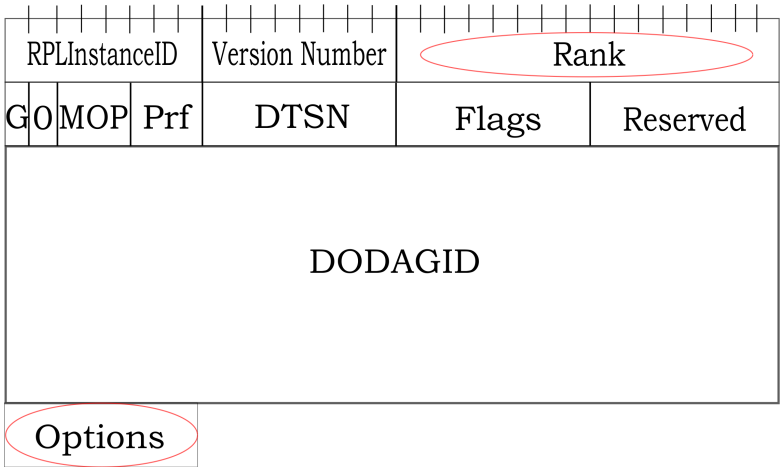

Figura 2.2: Campos del mensajes DIO de acuerdo al RFC6550[1].

2.2.3. Mensaje DAO

El mensaje DAO se emplea para propagar información de los destinos hacía las rutas de subida a lo largo de toda la red. En el método de almacenamiento los mensajes DAO se envían del hijo a su padre inmediato y para el método de no almacenamiento los mensajes son enviados al nodo raíz. El mensaje DAO cuenta con los siguientes campos:

- **RPLInstanceID:** Campo de 8 bits que indica la topología asociada a la red que se aprendió a partir de los mensajes DIO.
- **K:** Bandera que indica que se espera que el receptor mande un mensaje de confirmación.
- **D:** Bandera que indica que el campo DODAGID está presente.
- **Flags:** Campo de 6 bits que se mantiene sin uso y está reservado para banderas.
- **Reserved:** Campo de 8 bits sin uso.
- **DAOSequence:** Incrementa en cada mensaje DAO único.
- **DODAGID:** Campo de 128 bits que contiene un entero sin signo establecido por el DODAG raíz que identifica un DODAG.
- **Options:** El mensaje DAO tiene como opciones válidas: Pad1, PadN, RPL Target, Transit Information y RPL Target Descriptor.

Para esta tesis, en el mensaje DAO se considera el campo Options con la ópcion de Transit Information, ya que se emplea para que un nodo transmita información acerca de los destinos y sus atributos.

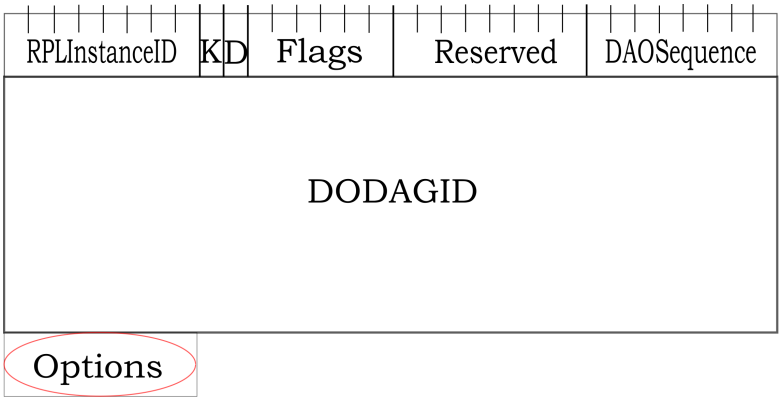

Figura 2.3: Campos del mensaje DAO de acuerdo al RFC6550[1].

2.3. Operación del protocolo RPL

A continuación para comprender el funcionamiento del protocolo RPL y el envío de mensajes de control de una manera más detallada, se utiliza el sistema operativo Contiki y el simulador Cooja. Se muestra una red con tres nodos en la que el nodo 1 se define como el RPL border router y los nodos 2 y 3 como sky-websens. Está red tiene una topología DAG con tres nodos donde el rango del nodo 1 abarca al nodo 2, el rango del nodo 2 abarca a los nodos 1 y 3 y el nodo 3 solo tiene en su rango al nodo 2 (ver figura 2.4).

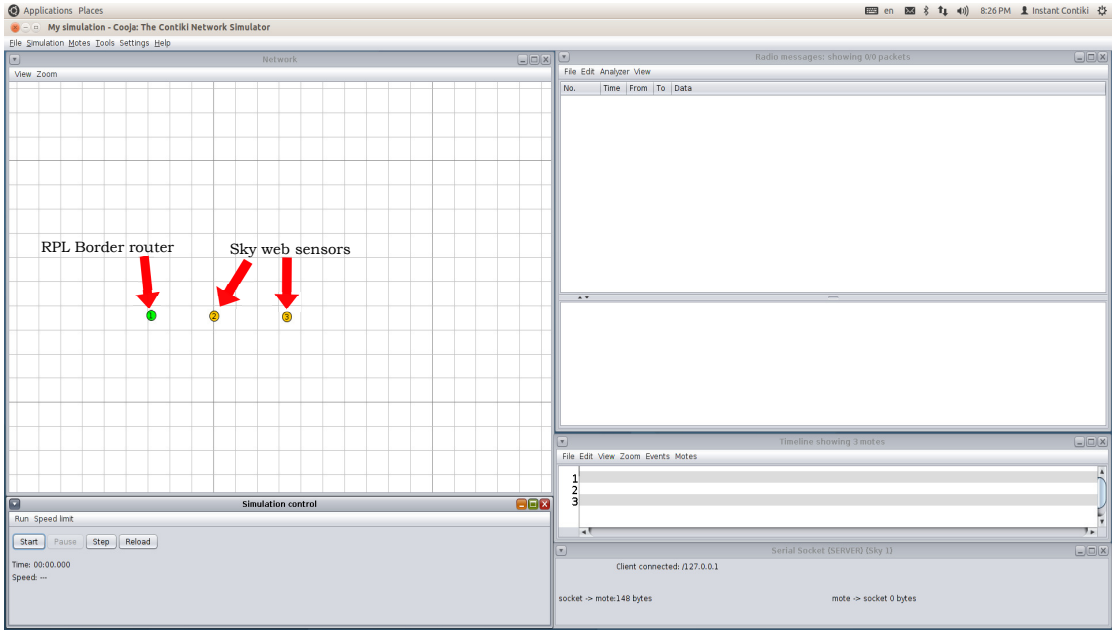

Figura 2.4: Simulación Cooja.

A través de la ventana `Radio messages` (ver figura 2.4) se puede visualizar el envío de los paquetes de cada nodo, a quiénes va dirigido y el tipo de paquete que se transmite. En la figura 2.5 se pueden observar los primeros mensajes DIS que se registran enviados por el nodo 2, el primero es enviado por difusión debido a que no se define un destinatario.

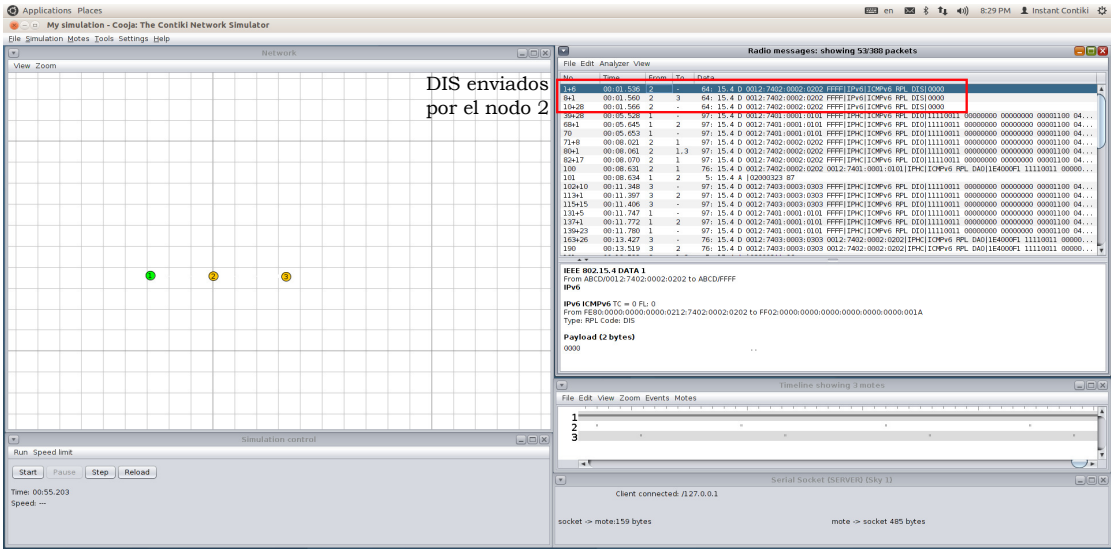

Figura 2.5: Mensajes DIS enviados por el nodo dos.

Una vez que el nodo 1 recibe los mensajes DIS, este responde enviando mensajes DIO (ver figura 2.6) con un rango y los demás parámetros sobre los cuales opera la red.

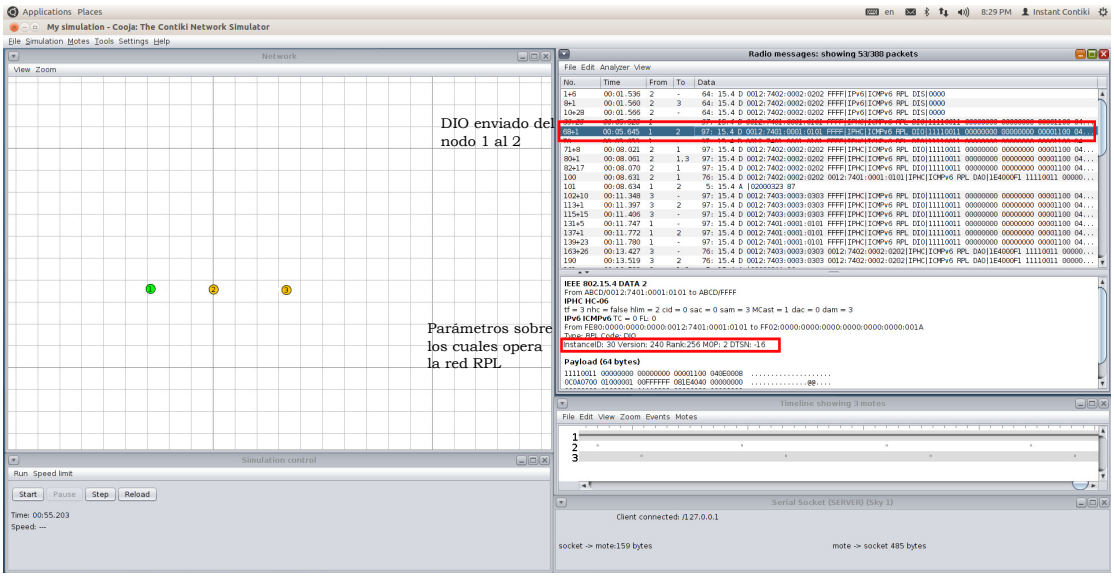

Figura 2.6: Mensaje DIO enviado por el nodo uno.

Posteriormente se le asigna un rango al nodo 2, con lo cual este puede transmitir mensajes DIOs a los demás nodos de la red, como se observa en la figura 2.7, donde el nodo 2 transmite mensajes DIO al nodo 1 y 3. En estos mensajes se puede observar el rango del nodo, los cuales son más grandes que el nodo 1 (nodo raíz).

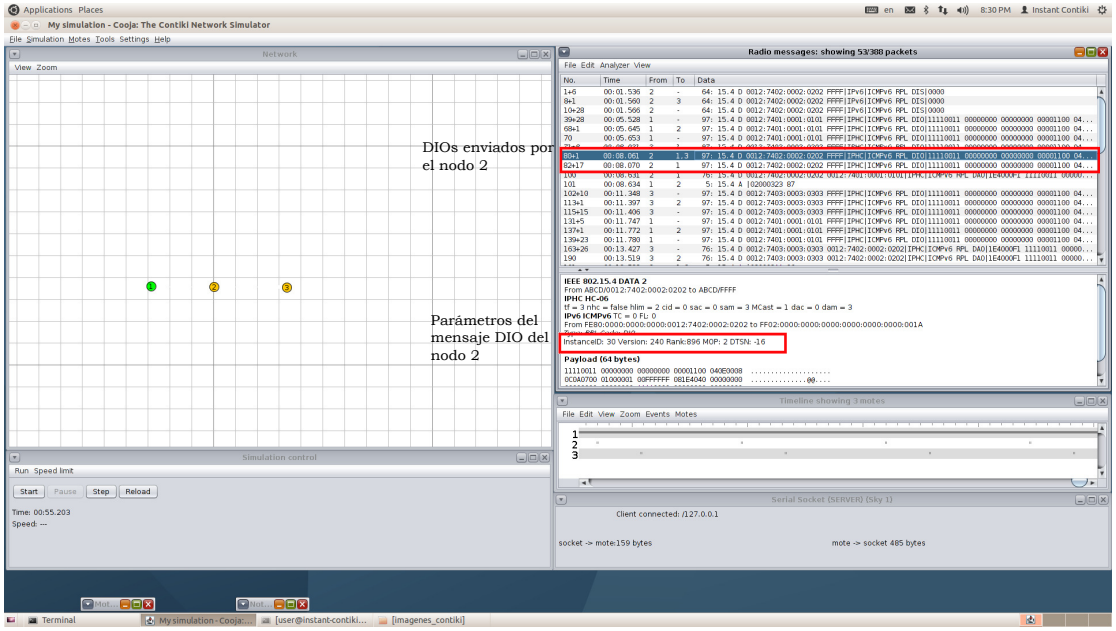

Figura 2.7: Mensajes DIO enviados por el nodo dos.

Con el posicionamiento del nodo 2 en la red, este transmite mensajes DAO con información de los destinos al nodo raíz (ver figura 2.8).

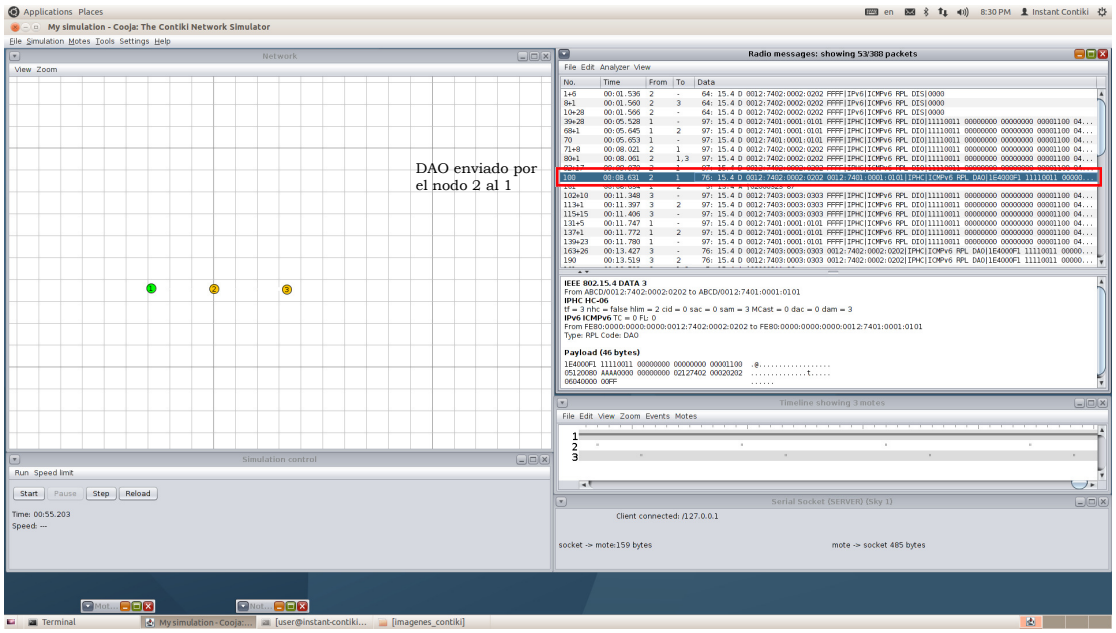

Figura 2.8: Mensaje DAO enviado por el nodo 2 al 1.

Cuando el nodo 3 recibe un rango después de recibir el DIO del nodo 2 como se aprecia en la figura 2.7, este comienza a transmitir sus mensajes DIO a los demás nodos de la red, el cual muestra un rango mucho mayor a los que se muestran en las figuras 2.6 y 2.7.

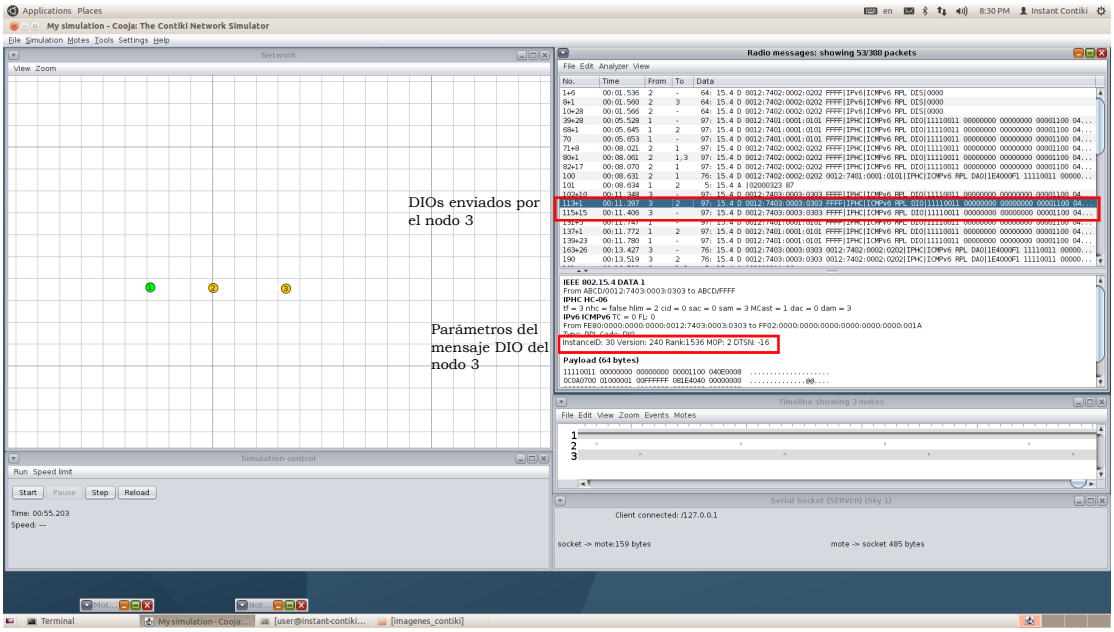

Figura 2.9: Mensajes DIO enviados por el nodo 3.

Una vez con un rango asignado y posicionado en la red, el nodo 3 transmite mensajes DAO sobre la información de los destinos que tiene (ver figura 2.10).

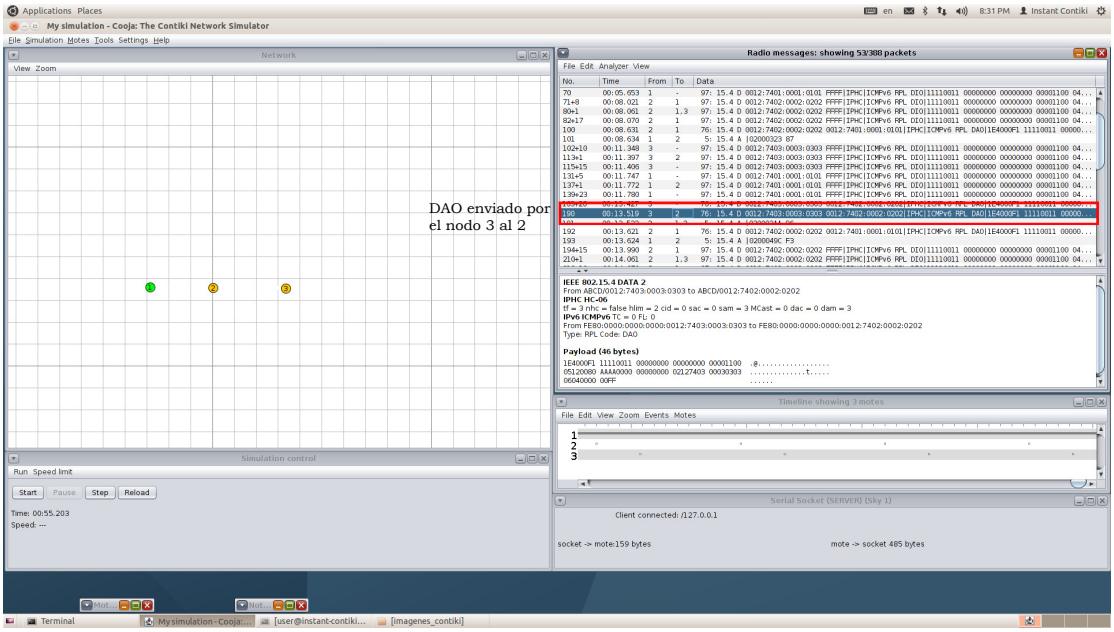

Figura 2.10: Mensaje DIO enviado por el nodo 3.

El proceso de envío de DIOs, DIS y DAO se repite con cierta frecuencia con el fin de que la red se pueda mantener actualizada y que la información pueda ser recibida hasta el nodo raíz (nodo 1).

A continuación, se explican los tres tipos de nodos que se utilizan en este trabajo de tesis.

2.3.1. Nodo raíz

El nodo raíz consta de dos acciones principales: el envío de DIOs cada vez que se recibe un DIS y la recepción de mensajes DAO. Mediante el uso de identificadores en el campo *Options* se identifican los tipos de mensajes que se reciben, por lo que

el comportamiento del nodo raíz se presenta a continuación en el algoritmo 1. Como se puede observar el comportamiento del nodo raíz mantiene el socket abierto en la espera de recibir algún mensaje de control, de los cuales analiza el contenido en el campo Options, cuando este contiene un valor 1 es porque recibió un DIS, por lo cual responde al nodo con un mensaje DIO, especificando el rango que le corresponde, si el valor contenido es un 3 se ha recibido un mensaje DAO, con lo cual se actualizan las rutas de bajada de la red.

Para propósitos de esta tesis y para facilitar la identificación de los mensajes se le asignó un 1 para identificar los DIS, 2 para los mensajes DIO, 3 para los DAO y 4 para las muestras que se miden en el sensor, lo cual no está establecido en el RFC6550, los detalles del código para el nodo raíz se pueden encontrar en los apéndices (ver código 6.1).

```
1 while socket abierto:
2 |   leer campo options del mensaje recibido;
3 |   if options==1:
4 |   |   responder DIS enviando un DIO;
5 |   if options==3:
6 |   |   actualizar rutas de bajada del DAO recibido
7 fintq
```

Algoritmo 1: Pseudocódigo para el nodo raíz.

En esta tesis no se considera tomar acción al recibir algún mensaje DIO de un nodo vecino debido a que la red se construye a través del rango como parámetro para encaminar los mensajes y se sabe que el nodo raíz siempre tiene el menor rango, por lo que no es necesario realizar algún tipo de actualización en el nodo raíz cuando se recibe un mensaje DIO.

2.3.2. Nodo

Un nodo que quiere formar parte de la red debe realizar dos procesos, primero debe transmitir por difusión un mensaje DIS para obtener los parámetros sobre los cuales opera la red y posteriormente debe responder a las solicitudes de información, además debe transmitir los paquetes que recibe de su padre y transmitir DAOs. A continuación se presenta el pseudocódigo (ver algoritmo 2), los detalles del código se

pueden observar en los apéndices (ver código 6.2).

```
1  while socket abierto:
2  |  leer campo options del mensaje recibido;
3  |  if options==1:
4  |  |  if Si no tengo rango asignado:
5  |  |  |  No respondo al DIS
6  |  |  else:
7  |  |  |  responder DIS enviando un DIO;
8  |  if options==2:
9  |  |  if No tengo rango asignado:
10 |  |  |  Realizo OF para que se me asigne un rango y asigno el transmisor del
   |  |  |   DIO como mi padre
11 |  |  if El rango de mi padre > al rango del DIO recibido:
12 |  |  |  Transmisor del DIO es mi nuevo padre
13 |  |  else:
14 |  |  |  Es una ruta de bajada y se envía un DAO
15 |  |  if Mi rango asignado < al rango del DIO recibido:
16 |  |  |  Es ruta de bajada y actualizo mis rutas de bajada y envío un DAO
17 |  if options==3:
18 |  |  Se actualizan las rutas de bajada del DAO recibido if Si tengo un padre:
19 |  |  |  Envío DAO a mi padre
20 |  if options==4:
21 |  |  if Tengo un padre:
22 |  |  |  Envío datos medidos por el sensor a mi padre
23 fintq
```

Algoritmo 2: Pseudocódigo para un nodo en la red RPL.

La forma en la cual se le asigna un rango a un nodo es mediante la función objetivo (OF), para propósitos de esta tesis se asignó mediante el envío de ecos (ICMP *pings*) al nodo que envía el mensaje DIO. Para ello, se toma el tiempo promedio de un conjunto de pings. Con este tiempo se multiplica por un valor de 100 y se le adiciona el rango del DIO recibido con el fin de asegurar que reciba un rango mayor que el de su padre.

2.3.3. Nodo sensor

El nodo sensor realiza las mismas tareas que un nodo que forma parte de la red RPL, sin embargo, debe realizar una tarea extra que consiste en medir valores de temperatura y humedad los cuales se transmiten a su padre. Para realizar la medición de estos parámetros se utilizó el sensor DHT11. En la figura 2.11 se presentan las conexiones con la Raspberry.

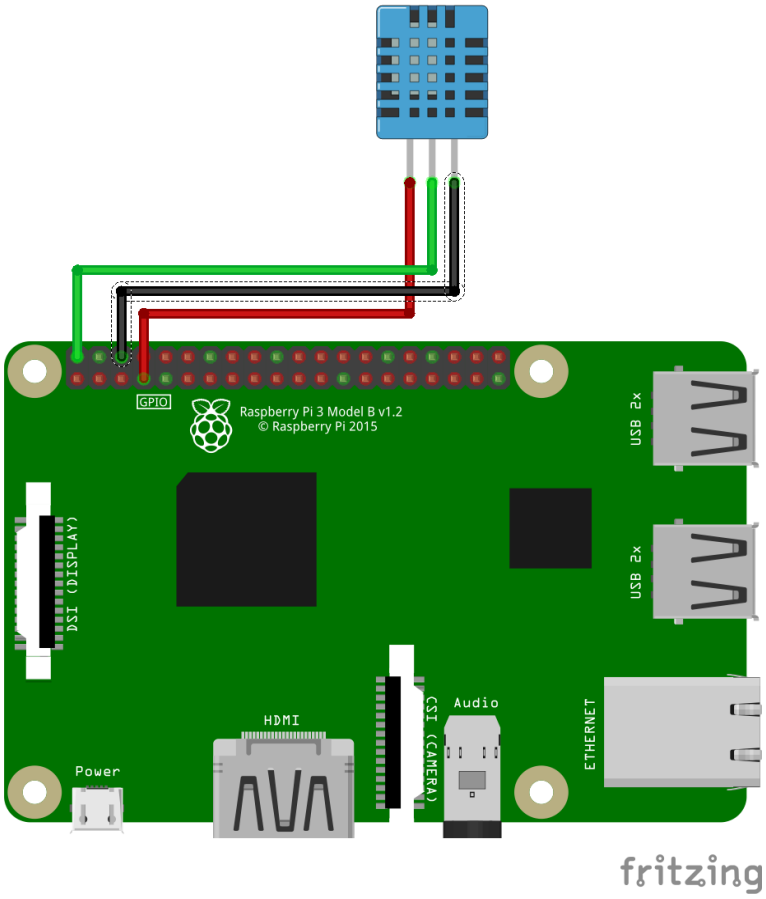

Figura 2.11: Diagrama de conexiones para el sensor DHT11.

El sensor DHT11 consta de un rango de humedad del $20\,\%$ a $90\,\%$ y un rango de temperatura de 0^oC a 50^oC y con una precisión de humedad del $\pm 5\,\%$ y de temperatura de $\pm 2^oC$. El pseudocódigo que se emplea en esta tesis se muestra en el algoritmo 3. Como se puede observar no se considera que el nodo sensor pueda recibir datos de algún otro sensor, ya que para propósitos de esta tesis se consideró como el último nodo de la red siendo el único que realiza esta tarea y que el encaminamiento de los paquetes es de subida hacia el nodo raíz, los detalles del código elaborado para el

funcionamiento de este nodo se pueden observar en la sección 6 (ver código 6.3).

```
1  while socket abierto:
2  |  leer campo options del mensaje recibido;
3  |  if options==1:
4  |  |  if Si no tengo rango asignado:
5  |  |  |  No respondo al DIS
6  |  |  else:
7  |  |  |  responder DIS enviando un DIO;
8  |  if options==2:
9  |  |  if No tengo rango asignado:
10 |  |  |  Realizo OF para que se me asigne un rango y asigno el transmisor del
   |  |  |   DIO como mi padre
11 |  |  if El rango de mi padre > al rango del DIO recibido:
12 |  |  |  Transmisor del DIO es mi nuevo padre
13 |  |  else:
14 |  |  |  Es una ruta de bajada y se envía un DAO
15 |  |  if Mi rango asignado < al rango del DIO recibido:
16 |  |  |  Es ruta de bajada y actualizo mis rutas de bajada y envío un DAO
17 |  if options==3:
18 |  |  Se actualizan las rutas de bajada del DAO recibido if Si tengo un padre:
19 |  |  |  Envío DAO a mi padre y comienzo a sensar y transmitir a mi padre
20 fintq
```

Algoritmo 3: Pseudocódigo para nodo sensor de la red RPL.

2.4. Resumen del capítulo

En este capítulo se presentó la operación del protocolo RPL, la formación de la red mediante los mensajes de control empleados como DIS, DIO y DAO para el encaminamiento de los datos. Se utiliza el simulador Cooja para comprender de mejor manera como se ejecuta este protocolo. Se muestran los campos que conforman cada mensaje de control del protocolo RPL, detallando la función de cada uno de ellos.

Capítulo 3

Sinkhole

3.1. Ataque sinkhole

En [3, 4], los autores hacen una recopilación de artículos relacionados con el protocolo RPL y los diversos ataques que pueden comprometer la integridad de la red. Uno de los ataques más dañinos a la topología de la red es el ataque sinkhole. De acuerdo a los autores en [5], la forma de operar de sinkhole es a través de un nodo malicioso, el cual advierte un camino ficticio y atrae la mayor cantidad de tráfico de los nodos cercanos a él. Ya que el rango es uno de los parámetros que los nodos emplean para determinar que ruta usar por defecto, el atacante de sinkhole advierte un mejor rango a sus nodos vecinos, atrayendo a los nodos de bajada con el fin de que éste sea visto como padre.

En [2] por ejemplo, los autores implementan un ataque sinkhole en una red RPL simulada en Cooja, cambiando el rango que se anuncia mientras se mandan los mensajes de control DIO. Sus resultados muestran que el ataque afecta de manera significativa a la red RPL, ya que ocasiona que una gran parte de los mensajes sean atraídos hacia el nodo malicioso. El ataque sinkhole es capaz de interrumpir la operación de la red, más aún si se combina con algún otro ataque puede llegar a ser muy peligroso. En [6, 7], los autores explican algunos métodos para defenderse y detectar este tipo de ataques, como son los sistemas de detección de intrusos (IDS), la recuperación de los padres y la autenticación del rango.

Una de las medidas empleadas para contrarrestar el ataque sinkhole se describe en [8], donde los autores proponen una técnica de autenticación de versión y de rango (VeRA). Este esquema evita que un nodo interno anuncie un rango falso mediante el envío de DIOs firmados, donde la firma del mensaje se construye a través de una cadena *hash* unidireccional. De esta manera el nodo receptor puede autenticar los mensajes de control y determinar si un nodo a lo largo de la ruta ha anunciado un rango falso. Posteriormente, la técnica VeRA se mejoró con la autenticación de topología (TRIAL) [9], en la cual los autores consideran la repetición de rango y los ataques de suplantación de identidad. Estos esquemas garantizan que la posición de un nodo no sea falsificada en la red.

Los autores en [7] combinan la técnica de VeRA y la técnica de recuperación de padres. Esta técnica se basa en la cuantificación de la tasa de entregas de mensajes de control al nodo raíz. Es decir, si el nodo raíz no recibe suficientes datos de un nodo por un cierto tiempo, éste incluye el ID del nodo en los futuros DIOs, con el fin de que los nodos que vean el ID en los mensajes DIO pongan a ese padre en la lista negra y busquen otro padre. Sin embargo, esta técnica solo detecta el ataque sinkhole, pero no es capaz de prevenirlo. Otra técnica de detección del ataque sinkhole se presenta en [10], donde los autores proponen la creación de una base de datos que registra el número de saltos a sus vecinos. De esta forma se calcula el número promedio de saltos y lo compara con un número mínimo de saltos. Si el valor mínimo es muy pequeño comparado con el número promedio, entonces se levanta una alerta de vulnerabilidad ante el ataque sinkhole.

En [11] se propone una clasificación de los diversos ataques en tres grupos de

acuerdo al objetivo del ataque, en el caso del ataque sinkhole el principal objetivo se enfoca en dañar la topología de la red. También existe la posibilidad de combinar el ataque, lo que puede ocasionar daños más relevantes a la red RPL de acuerdo a los autores en [2], donde aseguran que el ataque sinkhole puede ser combinado con el ataque de reenvío, el cual reenvía todos los mensajes de control de RPL y tira el resto del tráfico, también es posible combinarlo con el ataque de inundación de *Hello*, lo que permite transmitir mensajes con una fuerte potencia de señal con el fin de promover una métrica falsa de ruta.

En [10], los autores hacen una recopilación de varios artículos que detallan las características del ataque sinkhole, se resalta que este ataque se realiza en la capa de red del modelo OSI [12], que además es un ataque activo donde su objetivo es la pérdida de paquetes [13], la principal amenaza es que la confiabilidad y disponibilidad de la red se ve comprometida debido a la atracción del tráfico al nodo malicioso [14], también existe un alto nivel de daños ya que el atacante puede usar de manera ilegal la información que obtiene [13] y las posibilidades de detección son más difíciles cuando el nodo atacante se encuentra cerca de la estación base (nodo raíz) y las posibilidades de prevenir el ataque se ven mermadas cuando no se provee una autenticación del nodo [14][15].

Los autores en [9] proponen un mecanismo denominado TRAIL. Este se encarga de descubrir y aislar los nodos falsos mientras estos atacan la jerarquía del protocolo RPL. Sin embargo, TRIAL presenta problemas en el cual un nodo hijo escoge un nodo atacante como su padre, ya que un nodo hijo no puede determinar que su nodo padre es un nodo atacante. Ante estas vulnerabilidades, los autores en [16] proponen un algoritmo de selección de padres, para asegurar que un nodo hijo seleccione a un nodo legítimo como su padre mediante el uso de un umbral. Para ello, cada nodo obtiene un rango máximo y promedio de sus nodos vecinos. Por lo que cuando un nodo atacante pretende establecer un rango más bajo que los nodos legítimos, estos pueden ser rechazados como padres legítimos al presentar un rango extremadamente bajo.

Existen mecanismos tales como los sistemas de detección de intrusos (IDS), que tienen como enfoque de seguridad el detectar los ataques mediante la recolección de datos que se obtienen a través de varios parámetros de la red. SVELTE propuesto en [7], emplea detectores basados en firmas y anomalías. Con tres módulos colocados en el nodo raíz, mediante el primer módulo denominado mapaeador 6LoWPAN se colecta información relacionada del RPL y reconstruye la red en el nodo raíz, el segundo módulo analiza los datos mapeados y detecta intrusiones. Finalmente, el tercer módulo considera un pequeño *firewall* diseñado para filtrar el tráfico no deseado antes de que entre a la red. Sin embargo, el uso de mecanismos más complejos como firewalls o mapeos de toda la red implica un consumo mayor de recursos que suele ir en contra del objetivo del protocolo de operar con escasos recursos.

Con el propósito de reducir los falsos positivos y el consumo de recursos generado por SVELTE, los autores en [17] proponen IDS INTI. Este sistema combina enfoques de vigilancia, confianza y reputación para la detección de ataques sinkhole mediante un análisis del desempeño de los dispositivos. En [18] también se propone una mejora a IDS SVELTE, el cual usa una cantidad finita de estados de máquina para representar los estados que están relacionados con la estabilidad de la topología de la red y analiza esos estados de transición mediante un archivo de seguimiento. Este IDS está basado en la formación de grupos o clústers donde cada líder del clúster cuenta con un IDS que le permite grabar la información necesaria de sus nodos miembros. Este mecanismo reduce el consumo de recursos debido a que se reduce el procesamiento en cada nodo y se concentra en el líder del clúster. Sin embargo, esto puede representar un problema, debido a la centralización del mecanismo.

3.1.1. Implementación del ataque sinkhole

El ataque sinkhole se enfoca en atraer la mayor cantidad de tráfico mediante un nodo malicioso que altera su rango para mostrarse como el padre óptimo a todos los nodos que se encuentren en su rango de transmisión.

El ataque comienza mediante un envío constante de mensajes DIS con los cuales el nodo malicioso pretende obtener información sobre como opera la red RPL. Como se explicó en el capitulo 2 sobre la operación de RPL, en los mensajes DIO se encuentra la información necesaria para conocer la forma de operar de la red, ya que la respuesta de un mensaje DIS es el envío de un DIO, esto facilita al nodo malicioso obtener conocimiento del rango de sus vecinos con el fin de que pueda auto asignarse un rango óptimo para parecer el padre óptimo. El pseudocódigo propuesto para que el nodo malicioso opere es el siguiente (ver algoritmo 4):

```
1  while socket abierto:
2  |  Enviar mensajes DIS y leer campo options del mensaje recibido;
3  |  if options==1:
4  |  |  if Si no tengo rango asignado:
5  |  |  |  No respondo al DIS
6  |  |  else:
7  |  |  |  responder DIS enviando un DIO if No he recibido DAOs:
8  |  |  |  |  Disminuyo mi rango
9  |  if options==2:
10 |  |  Se lee el rango del DIO recibido y se autoasigna un rango muy bajo
11 fintq
```

Algoritmo 4: Pseudocódigo para nodo sinkhole.

Como se observa en el algoritmo 4, el nodo malicioso mantiene su socket abierto con el fin de enviar mensajes DIS y poder recibir un DIO como respuesta, esto con el fin de obtener información necesaria para la operación de la red. Al mantener un socket abierto el nodo malicioso recibe mensajes DIS, si no tiene un rango asignado no responde a la solicitud de información, en caso contrario, responde a las solicitudes de información enviando su DIO. Si éste no recibe como respuesta mensajes DAO, decrementa su rango (lo cual se hará hasta recibir mensajes DAO). En el caso de que se reciba un mensaje DIO, lee el rango del nodo que envió el DIO y se asigna un valor más bajo del rango de su vecino.

Dadas las dificultades que puede representar obtener el rango de un nodo que no se encuentra dentro del área de transmisión del nodo malicioso y asumiendo el caso en que dicho nodo sea el padre de alguien más, se debe realizar la disminución del rango hasta recibir un mensaje DAO, para un mejor detalle sobre el ataque y la implementación se puede observar el código implementado en el apéndice (ver código 6.4).

3.2. Resumen del capítulo

En este capítulo se presentó una explicación sobre el modo de operación del ataque sinkhole, así como las contra medidas más relevantes para identificar o mitigar el ataque sinkhole. Sin embargo, muchas de estas medidas requieren de muchos recursos de memoria y energía o de algoritmos de cifrado.

Capítulo 4

Implementación y resultados

Este capítulo explica con detalle la topología de red que se implementó para montar un escenario real del protocolo RPL usando el rango como parámetro de encaminamiento. También se describen los elementos que se usaron para diseñar el ataque sinkhole y se muestran los resultados que se registraron mediante el analizador de tráfico Wireshark.

4.1. Implementación

Para la implementación física del protocolo RPL se diseñó una red ad-hoc, cuyos IDs son las direcciones IPv4, a pesar de que el estándar RPL está basado en direcciones IPv6, en esta tesis se asignaron direcciones IPv4 con el fin de facilitar la implementación, ya que esto no afecta el ataque y permite analizar el comportamiento de la red cuando esta emplea el rango para el encaminamiento de los mensajes. En la figura 4.1 se presenta la topología de la red y sus rangos de cobertura, así como las IPs utilizadas.

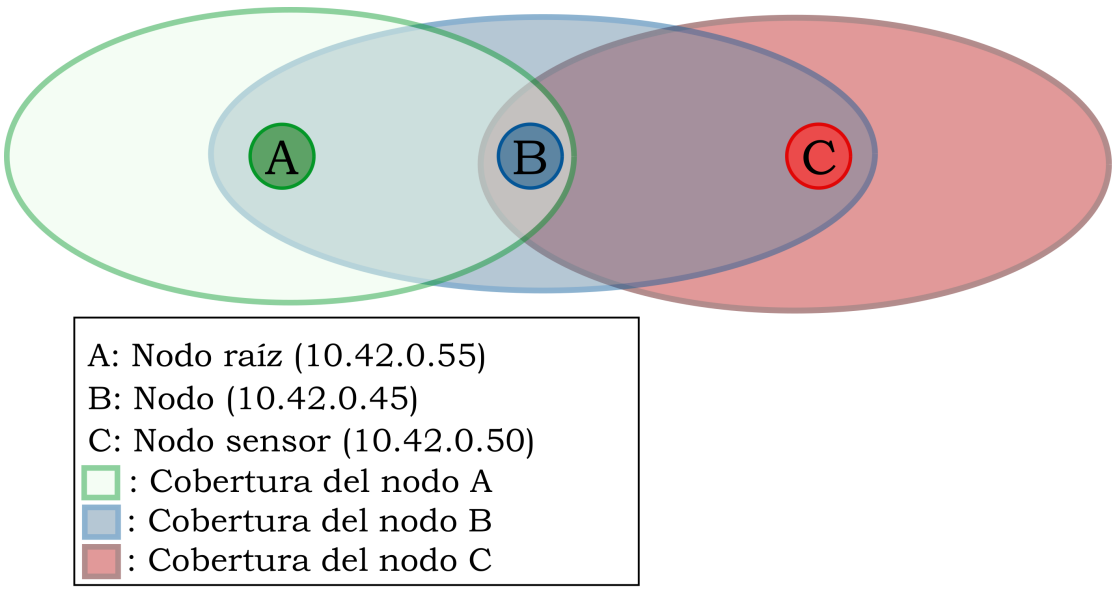

Figura 4.1: Topología de red, IPs de los nodos y rangos de cobertura.

El hardware empleado en los nodos B y C son Raspberries pi 3, debido a su movilidad, compatibilidad con el lenguaje Python y con el sensor DTH11, el cual puede medir temperatura y humedad a través del diagrama mostrado en la figura 2.11.

En el Apéndice 6 se muestran los códigos del nodo raíz, los nodos de red y el nodo malicioso. La figura 4.2 muestra la configuración de la red (similar a la mostrada en el capítulo 2). En esta figura el nodo C a través del sensor DTH11 registra valores de

temperatura y humedad que son enviados al nodo A. El nodo X representa al nodo malicioso que realiza el ataque sinkhole. Se puede observar que el nodo X solo tiene al nodo B en su rango de cobertura, el cual está a un salto del nodo raíz. La ejecución del ataque se realiza con una Raspeberry pi 3 con la dirección IP 10.42.0.59.

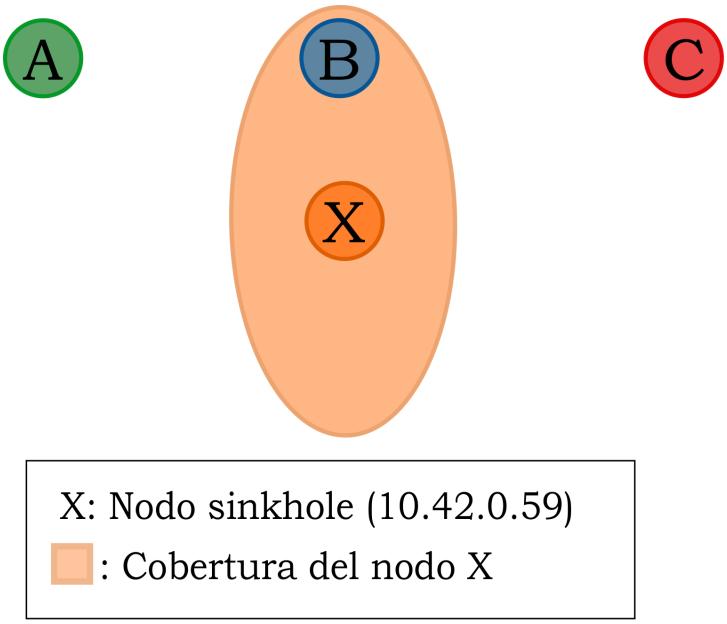

Figura 4.2: Rango de cobertura del nodo sinkhole.

Con el propósito de obtener la posición óptima del nodo atacante, se utilizó la técnica llamada *transmission power control* (TPC), en la cual se selecciona una potencia de transmisión con el fin de que solo el nodo más cercano pueda escuchar las transmisiones del nodo malicioso, lo que decrece la precisión de los algoritmos de localización por un cierto factor. Para ello se utilizo la idea propuesta en [19], donde mediante el comando *iwcongif* del *shell* de Linux, se puede controlar la potencia de transmisión.

4.2. Resultados

4.2.1. Resultados de RPL

La figura 4.3 muestra el envío de un mensaje DIS por parte del nodo B. Se puede observar en esta figura que la dirección fuente es 10.42.0.45 y la dirección destino es la dirección broadcast de la red 10.42.0.255.

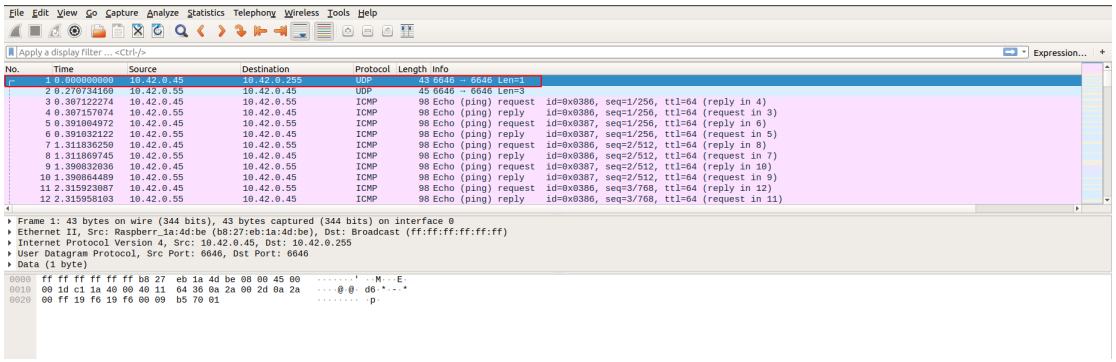

Figura 4.3: Envío de un mensaje DIS del nodo B al nodo A.

Posteriormente, el nodo A responde el DIS mediante un DIO al nodo B como se muestra en la figura 4.4 a la dirección 10.42.0.45. El rango del nodo B se calcula con

el uso de la función OF mediante el envío de paquetes ICMP (pings) al nodo A. En la figura 4.5 se observa las peticiones y respuestas de los paqutes ICMP. Una vez que se le asigna un rango al nodo B, este nodo envía un DAO de 45 bytes que se muestra en la figura 4.6 proveniente de la dirección 10.42.45 a la dirección 10.42.0.55, el cual al no tener destinos de bajada estará vacío. El proceso de envío de mensajes DIS, DIO y DAO se repite en varias ocasiones como se puede observar en la figura 4.7 con el fin de descubrir todas las rutas en la DAG.

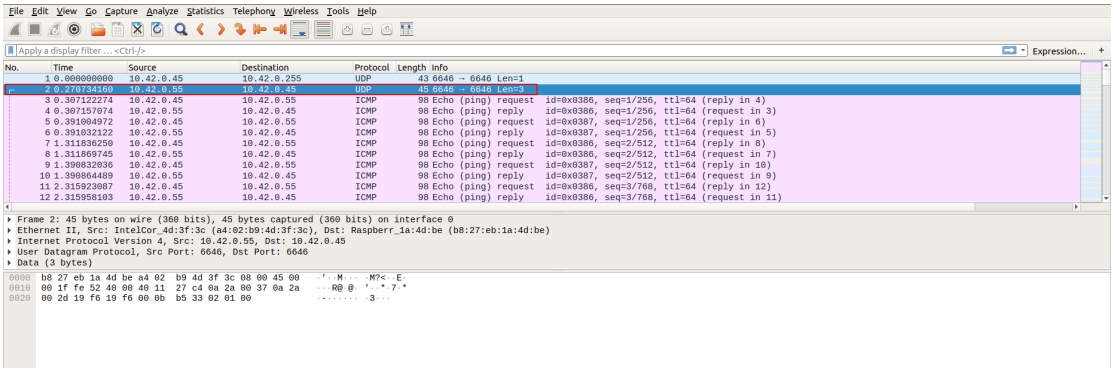

Figura 4.4: Envío de un mensaje DIO del nodo A al nodo B.

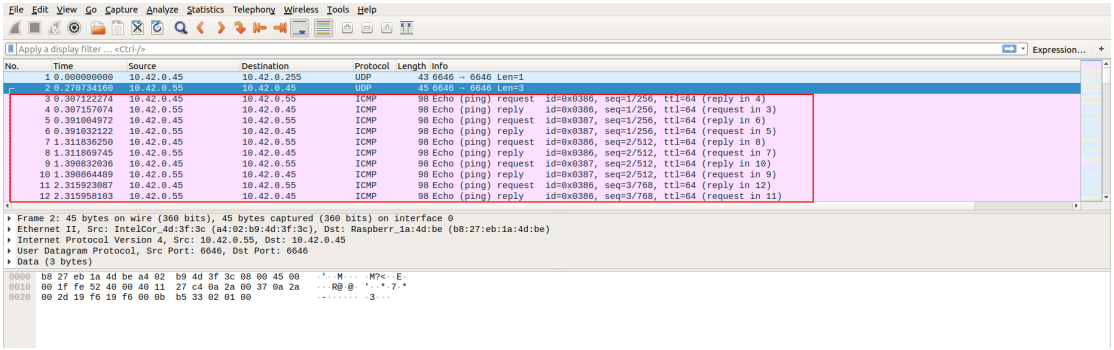

Figura 4.5: Mensajes ICMP para la asignación de rango al nodo B.

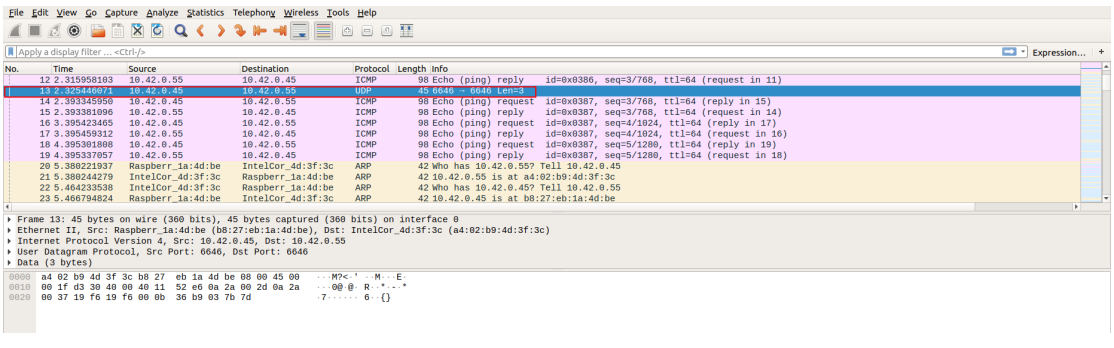

Figura 4.6: Envío de mensaje DAO del nodo B al nodo A.

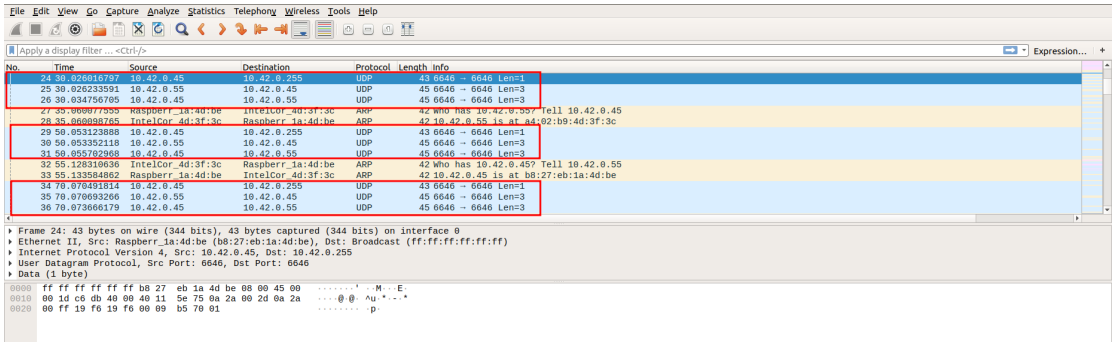

Figura 4.7: Envío de mensajes DIS, DIO y DAO entre los nodos A y B.

Una vez que el nodo B pertenece a la red, el nodo sensor (C) manda un mensaje DIS en broadcast como se muestra en las figura 4.8 mediante un mensaje de 43 bytes de longitud, donde la fuente es la dirección 10.42.0.50 y la dirección destino es la dirección broadcast 10.42.0.255.

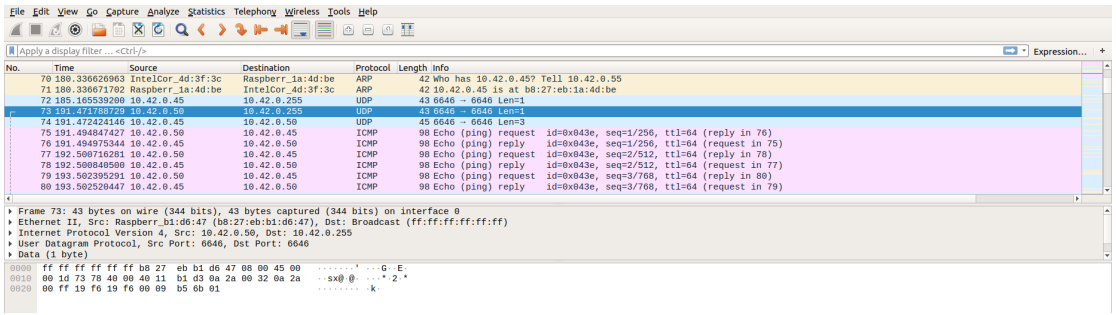

Figura 4.8: Envío de un mensaje DIS del nodo C.

Como respuesta, el nodo B envía un DIO al nodo C y se realiza la OF para que se le asigne un rango al nodo C, estos mensajes se pueden observar en la figura 4.9, donde se tiene un mensaje cuya fuente es la dirección 10.42.0.45 y con destino la dirección 10.42.0.50. Posteriormente se realizan los *request* y *reply* del protocolo ICMP para obtener el rango del nodo C que quiere formar parte de la red.

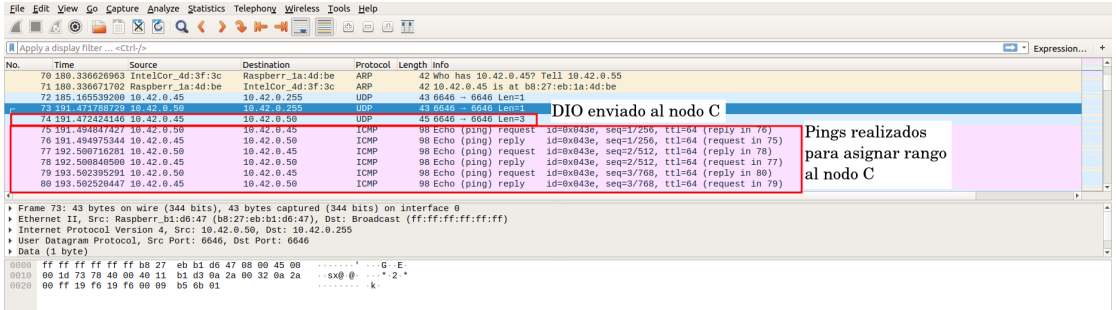

Figura 4.9: Envío de un mensaje DIO del nodo B al C y asignación de rango.

Una vez que el nodo C (nodo sensor) es parte de la red y tiene un padre asignado, este comienza a medir valores de temperatura y humedad y los transmite a su padre quien es el nodo B, quien a su vez tiene como padre asignado al nodo A.

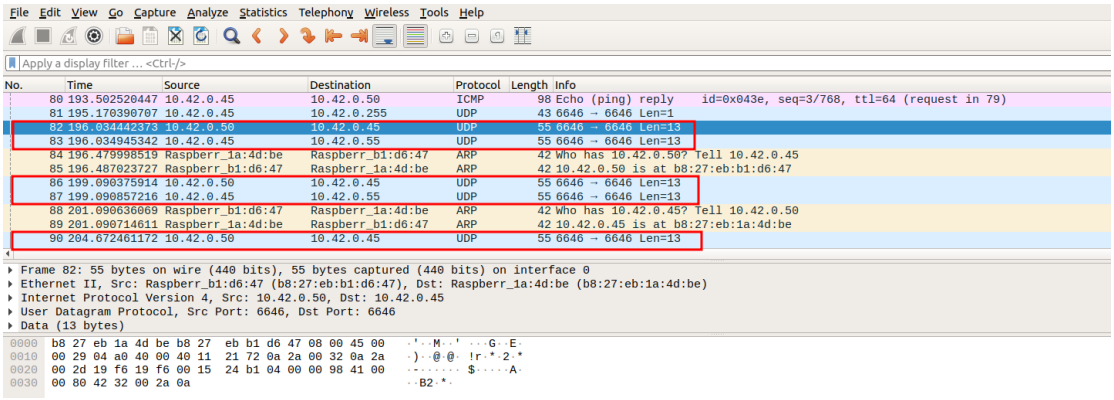

Figura 4.10: Envío de los valores medidos por el sensor.

En la figura 4.10 se puede observar el tráfico registrado por el sensor y como estos mensajes son transmitidos de la dirección IP 10.42.0.50 (nodo C) a la dirección 10.42.0.45 (nodo B) y posteriormente de la 10.42.0.45 a la 10.42.0.55 (nodo A), con lo cual se puede observar que el encaminamiento de los paquetes se hace de acuerdo a los rangos de mayor a menor.

4.2.2. Resultados del sinkhole

Una vez que la red se encuentra operando y que los mensajes se transmiten del nodo C al nodo A, se realiza el ataque sinkhole siguiendo el rango de cobertura que se muestra en la figura 4.2. En la figura 4.11 se muestra la transmisión de mensajes DIS en broadcast por parte del nodo X con el fin de que reciba los parámetros de operación de la red mediante los mensajes DIO. En esta figura también se puede ver cómo la dirección 10.42.0.59 envía un mensaje DIS a la dirección broadcast 10.42.0.255 que posteriormente este mensaje es respondido por un mensaje de 45 bytes (DIO) por la dirección 10.42.0.45.

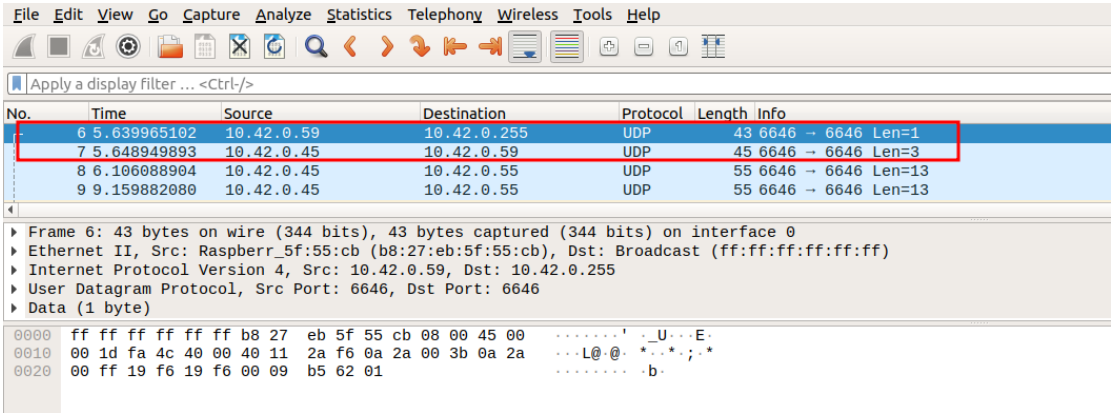

Figura 4.11: Envío de un mensaje DIS del nodo X y respuesta de un mensaje DIO.

Como se mencionó anteriormente, el ataque sinkhole no es inmediato, ya que éste debe encontrar un rango tal que los nodos vecinos lo tomen como nuevo padre. Esto se puede observar en el envío de paquetes, ya que se siguen transmitiendo algunos paquetes hacia el nodo A. Esto es debido al tiempo que tarda en asignarse un mejor rango para el nodo B y en la espera de que el nodo X envíe sus mensajes DIO. Una vez que el mensaje DIO del nodo X es recibido por el nodo B como se puede observar en la figura 4.12, el nodo B con la dirección 10.42.0.45 cambia de padre y transmite los paquetes al nodo X en vez de al nodo A como se observa en la figura 4.12, lo cual se ve reflejado en el tráfico del nodo A donde deja de recibir estos paquetes sin tener conocimiento de la dirección IP del nodo X como se puede observar en las figuras 4.13 y 4.14.

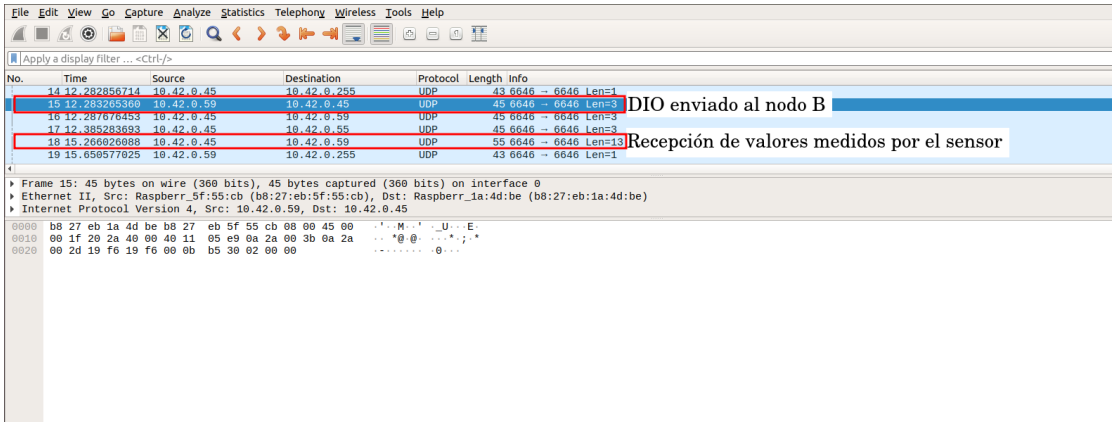

Figura 4.12: Envío de DIO del nodo X y recepción de valores medidos por el sensor.

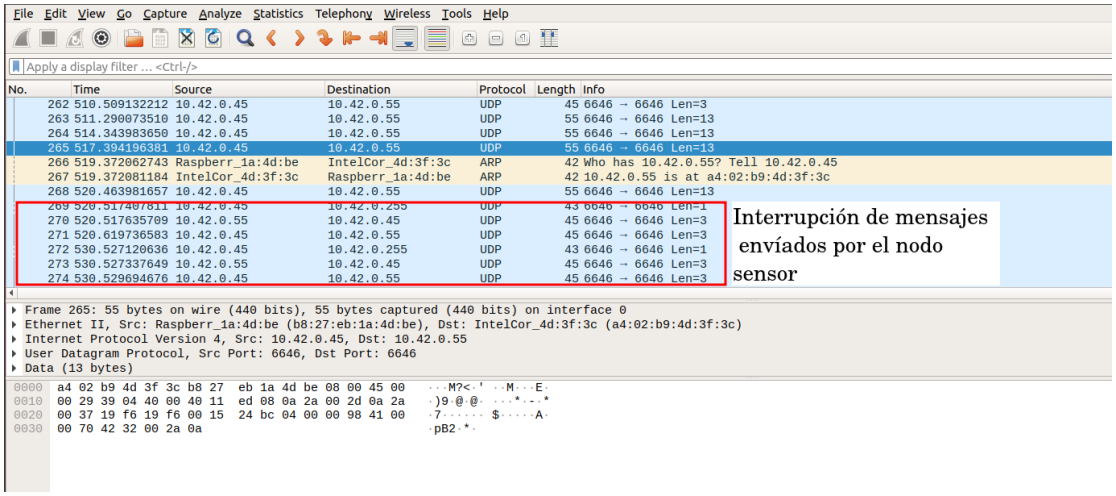

Figura 4.13: Interrupción de los mensajes del nodo C al nodo A.

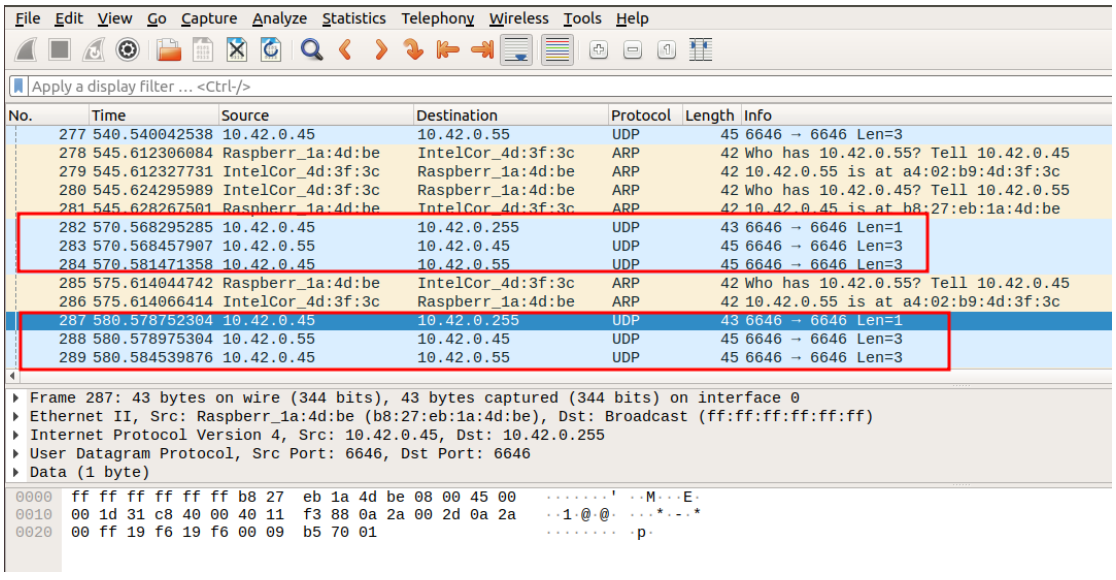

Figura 4.14: Tráfico registrado en el nodo A una vez que se ejecuta el ataque sinkhole.

Analizando a detalle el tráfico registrado en el nodo atacante para este ejemplo, se registra el envío de un mensaje DIS por broadcast a través de la máscara de red

255.255.255.0, el envío de este mensaje se dio en el tiempo $t_0 = 5{,}639965102$seg, la respuesta a la solicitud de información por parte de un nodo activo de la red se dio en el tiempo $t_1 = 5{,}648949893$seg, por lo que el tiempo de respuesta de la red a un DIS de un nuevo nodo se dio en $8{,}984791$ms. En el tiempo $t_2 = 12{,}283265360$seg el nodo atacante transmite un DIO falso mostrando un mejor rango, posteriormente en el tiempo $t_3 = 15{,}266026088$seg se registra la recepción de los datos medidos por el sensor de temperatura y humedad, por lo que se puede deducir que al nodo B con dirección IP 10.42.0.45 le tomó $2{,}982760728$seg en asignar al nodo sinkhole como su nuevo padre.

Enfocándonos en el tráfico recibido por el nodo B con dirección IP 10.42.0.45, se puede hacer un análisis sobre la cantidad de los mensajes que fueron medidos por el sensor y si fueron encaminados correctamente hasta el nodo raíz.

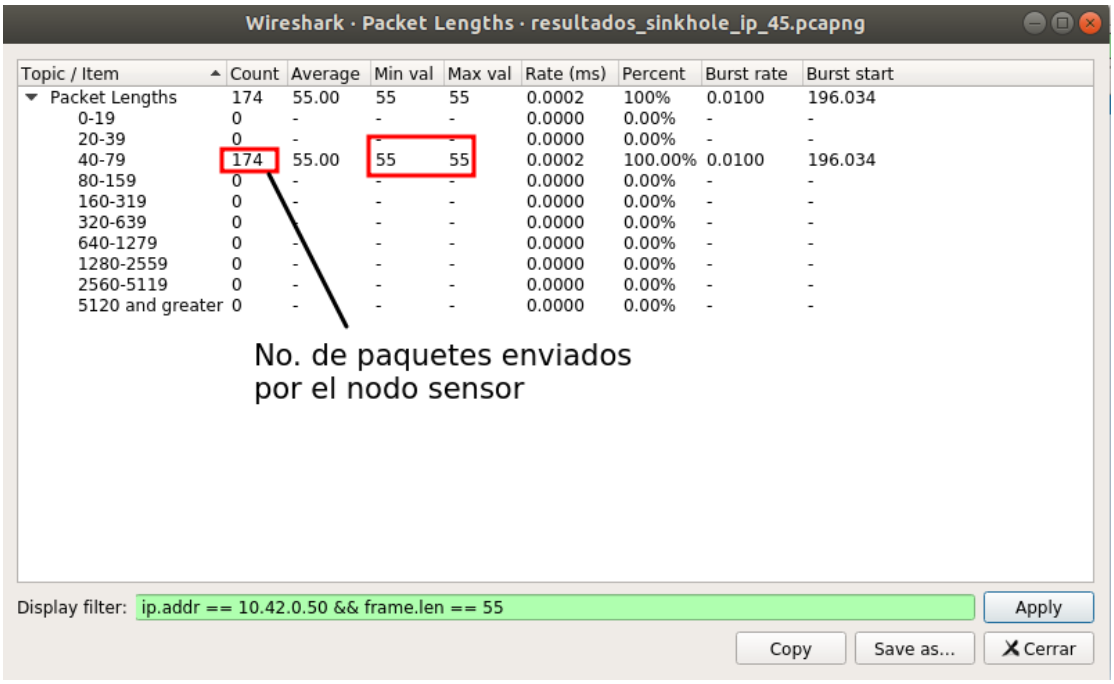

Figura 4.15: Paquetes totales enviados por el nodo sensor

De acuerdo al filtrado del tráfico captado en el nodo B, se recibieron 174 paquetes con una longitud de frame de 55 bytes (longitud que corresponde a los datos que se envían por el nodo sensor), se registró que 85 paquetes fueron enviados al nodo raíz, lo que corresponde a un $48{,}85\,\%$, mientras que 87 paquetes fueron enviados al nodo sinkhole, lo que representa el $50\,\%$ de los paquetes totales. Existen dos paquetes no se enviaron ni al nodo raíz ni al sinkhole por lo que podría decirse que el $1{,}15\,\%$ de los paquetes se perdieron en el medio.

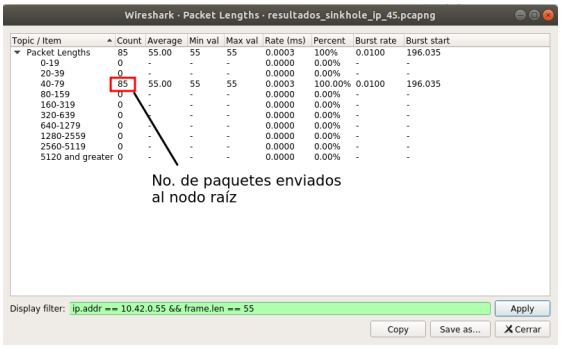

Figura 4.16: Paquetes enviados al nodo raíz

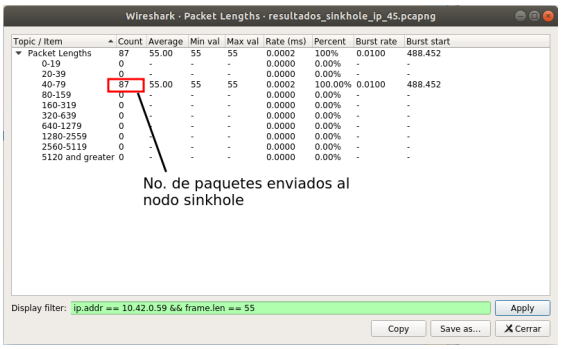

Figura 4.17: Paquetes enviados al nodo sinkhole

4.3. Resumen del capítulo

En este capitulo se explica la topología de la red empleada, se detallan las direcciones IPs de cada nodo que forma parte de la red, así como sus rangos de transmisión. Se explica a detalle el proceso para ejecutar el ataque sinkhole y se define el rango de transmisión del nodo malicioso, así como la dirección IP asignada. Se transmiten datos de temperatura y humedad hacia el nodo raíz y se observa el movimiento del tráfico mediante el analizador Wireshark. Posteriormente se ejecutó el ataque sinkhole para desviar el tráfico y evitar que este llegara al nodo raíz, la desviación e interrupción de los mensajes se registró mediante Wireshark.

Capítulo 5

Conclusiones

5.1. Conclusiones generales

La formación de una red RPL requiere solo de tres mensajes de control, la solicitud de información mediante mensajes DIS, el envío de los parámetros de operación de la red y los diferentes destinos que conforman la red mediante los mensajes DIO y DAO. Gracias a esto se facilita la comunicación entre los nodos y a su vez se cumple el propósito que tiene este protocolo para dispositivos de recursos limitados. Sin embargo, son esas mismas facilidades las que exponen su integridad, así como la información que se transmite a través de ella.

Como se ha mencionado en esta tesis, el protocolo RPL es susceptible a varios ataques. Por ejemplo, el ataque sinkhole es uno de los que ocasiona daños en la estructura de la red, ya que redefine los caminos empleados en la transmisión de los paquetes. Este cambio en las rutas genera un problema en la seguridad dentro de la misma, ya que al no llegar la información hasta el nodo raíz, no se puede garantizar que la información esté comprometida o que los atacantes puedan hacer mal uso de la información.

La simplicidad de asignar un sólo parámetro como el rango para definir al padre de un nodo es lo que facilita a los atacantes redirigir la información y de esta manera obtener patrones o inclusive leerla si no está cifrada.

En relación con las pruebas realizadas, es posible deducir que la posición del nodo que ejecuta el ataque es un factor importante en cuanto a la cantidad de información que puede llegar a dicho nodo. Como se observó en los resultados, al colocar el nodo en un punto entre el último salto y fuera del rango del nodo raíz se puede considerar como un punto de alto riesgo debido a que al ejecutarse el ataque se obtiene la información de todos los destinos de bajada, así también el estar fuera del rango del nodo raíz puede complicar la identificación del destino de la información. Esto da una clara evidencia física de las repercusiones que se tienen en el tema de seguridad, debido a la simplicidad en la que opera RPL. Por otro lado, la implementación física muestra detalles que no son visibles en un simulador, cómo localizar el punto óptimo del ataque, ya que en el mundo físico los patrones de radiación de las antenas cambian constantemente y eso puede hacer que los nodos pierdan la conexión o que el nodo raíz identifique al nodo malicioso.

5.2. Verificación de la hipótesis

Como se escribió en la hipótesis de la sección 1.2:

> "El cambio de rango y la posición estratégica de un nodo malicioso permiten establecer un ataque sinkhole y vulnerar la red al atraer la mayor parte del tráfico hacía el nodo malicioso"

Una vez que se establece la red RPL y la correcta transmisión de los datos, como se observa en la figura 4.10, es posible establecer el escenario para ejecutar el ataque

sinkhole como se muestra en el diagrama de la figura 4.2, donde se establece la posición del nodo atacante y se registra el inicio de la ejecución del ataque (ver figura 4.11) mediante el envío de mensajes DIS en broadcast, así como la respuesta recibida con mensajes DIO. Gracias a esto el nodo atacante se establece dentro de la red RPL, ya que este es capaz de enviar DIOs falsos que muestran un camino beneficioso dentro de la red, como se puede apreciar en la figura 4.12. Más aún, el nodo atacante es capaz de recibir los datos que se miden en el sensor DHT11. Esta interrupción también se registra en el tráfico recibido por el nodo raíz en las figuras 4.13 y 4.14, esto a su vez demuestran que al no estar el nodo atacante dentro del área de cobertura del nodo raíz, este último no tiene conocimiento de la presencia del nodo malicioso, lo que permite una libre operación del ataque. Todo lo anterior muestra como el cambio de rango en el nodo malicioso permite vulnerar la topología y la información de la red.

Capítulo 6

Apéndices

6.1. Código nodo raíz

```
import socket as s
from thread import start_new_thread
from struct import*
from subprocess import Popen, PIPE
import re
import time

class RPL_border:
    UPPORT=6646
    TCPPORT=7646
    broadcast_ip='10.42.0.255'

    def __init__(self,mi_ip):
        self.mi_ip=mi_ip
        self.vecinos=[]
        self.rank=0
        self.id=1
        self.down={}

    def config(self):
        sock=s.socket(s.AF_INET,s.SOCK_DGRAM)
        sock.setsockopt(s.SOL_SOCKET,s.SO_REUSEADDR,1)
        sock.setsockopt(s.SOL_SOCKET,s.SO_BROADCAST,1)
        sock.bind(('',self.UPPORT))
        time.sleep(3)
        start_new_thread(self.recibir_udp,(sock,))
        time.sleep(15)

    def recibir_udp(self,sr):
        while True:
            data,addr=sr.recvfrom(1024)
            opt=unpack('b',data[0:1])[0]
            if addr[0]!=self.mi_ip:
                if addr[0] not in self.vecinos:
                    self.vecinos.append(addr[0])
            if opt == 1:
                print "DIS recibido de:", addr[0]
                self.DIO_message(addr[0],sr)
                print "Vecinos:",self.vecinos

            if opt == 3:
                print "DAO RECIBIDO DE:",addr[0]
                DAO=data[1:len(data)]
                rutas=eval(DAO)
                self.down.update({addr[0]:rutas})
                print "Rutas de bajada:",self.down

    #Creacion y envio de mensajes DIO
    def DIO_message(self,addr,sk):
        Option=pack('b',2)
```

```
        Rank=pack('H',self.rank)
          DIO=Option+Rank
          print "ENVIANDO DIO a:",addr
          sk.sendto(DIO,(addr,self.UPPORT))

HOST='10.42.0.55'
rpl_border=RPL_border(HOST)
rpl_border.config()

while True:
    time.sleep(1)
    pass
```

códigos 6.1: Código empleado para el nodo raíz

6.2. Código nodo

```
import socket as s
from thread import start_new_thread
from struct import*
import time
from subprocess import Popen, PIPE
import re
import json

class RPL_nodo:
    UPPORT=6646
    TCPPORT=7646
    broadcast_ip='10.42.0.255'

    def __init__(self,mi_ip):
        self.mi_ip=mi_ip
        self.padre={}
        self.rank=0
        #self.version=0
        #self.id=0
        #self.DODAGID=1
        self.down={}
        self.vecinos=[]

    def config(self):
        sock=s.socket(s.AF_INET,s.SOCK_DGRAM)
        sock.setsockopt(s.SOL_SOCKET,s.SO_REUSEADDR,1)
        sock.setsockopt(s.SOL_SOCKET,s.SO_BROADCAST,1)
        sock.bind(('',self.UPPORT))
        time.sleep(3)
        start_new_thread(self.recibir_udp,(sock,))
        time.sleep(15)
        start_new_thread( self.DIS_message , ( sock , ) )
        time.sleep(15)

    #Creacion y envio de mensajes DIS
    def DIS_message(self,st):
        while True:
            opcion=1
            Option=pack('b',opcion)
            mensaje=Option
            print "####################ENVIANDO DIS#########################"
            st.sendto(mensaje,(self.broadcast_ip,self.UPPORT))
            time.sleep(10)


    def recibir_udp(self,sr):
        while True:
            data,addr=sr.recvfrom(1024)
            vecino=[addr[0]]

            if addr[0]!=self.mi_ip:
                if addr[0] not in self.vecinos:
```

```
53                      self.vecinos.append(addr[0])

55                  opt=unpack('b',data[0:1])[0]
                    if opt==1 and addr[0]!=self.mi_ip:
57                      print "DIS recibido de:",addr[0]
                        if self.rank==0:
59                          print "No formo parte de la red RPL:"
                        else:
61                          self.DIO_message(addr[0],sr)

63                  elif opt==2:
                        print"DIO recibido de:",addr[0]
65                      rango=unpack('H',data[1:3])[0]
                        print "Rango de DIO recibido:",rango
67
                        if self.rank==0:
69                          self.rank=self.rank+rango+int(10*self.OF(vecino))
                            print "Mi rango es:",self.rank
71                      #else:
                        #    if self.padre.keys()==[]:
73                          print "Yo soy tu padre:",addr[0]
                            self.padre.update({addr[0]:rango})
75
                        elif self.padre.values()>[rango] and addr[0] not in self.padre.
    keys():
77                          print "Yo soy tu nuevo padre:",addr[0]
                            self.padre.clear()
79                          self.padre.update({addr[0]:rango})
                            if self.down=={}:
81                              print "No hay rutas de bajada"
                            else:
83                              #ENVIAR DAO
                                self.DAO(addr[0],sr)
85                      elif self.rank < rango and addr[0] != self.mi_ip:
                            print "Es ruta de bajada:",addr[0]
87                          self.down.update({addr[0]:rango})
                            print "Mis rutas de bajada",self.down
89                          self.DAO(addr[0],sr)
                        self.DAO(addr[0],sr)
91
                    elif opt==3:
93                      print "DAO recibido de:",addr[0]
                        DAO=data[1:len(data)]
95                      rutas=eval(DAO)
                        print "ACULIZANDO RUTAS DE BAJADA..."
97                      self.down.update(rutas)
                        print "Rutas de bajada:",self.down
99                      if self.padre.keys()!=[]:
                            padre=self.padre.keys()
101                         self.DAO(addr[0],sr)

103                 elif opt==4:
                        if self.padre.keys()!=[]:
105                         padre=self.padre.keys()[0]
                            print "Datos de sensor recibido"
107                         sr.sendto(data,(padre,self.UPPORT))
                        else:
109                         break
                    else:
111                     break

113
    #Cracion y envio de mensajes DAO
115 def DAO(self,addr,sk):
        print "ENVIANDO DAO..."
117     Option=pack('b',3)
        DAO=Option+str(self.down)
119     sk.sendto(DAO,(addr,self.UPPORT))

121 #Funcion para obtener rango del nodo
    def OF(self, lista):
123     vecinos=lista
        for i in range (0,len(vecinos)):
```

```
125             comando='ping -W 1 -c 3'+' '+vecinos[i]
                cmd = Popen(comando.split(' '), stdout=PIPE)
127             output = cmd.communicate()[0]
                print output
129             match = re.search('(\d+\.\d+)\/(\d+\.\d+)\/(\d+\.\d+)\/(\d+\.\d+)\s+ms',
    output)
                print match
131             if not (match is None):
                    #print float(match.group(1))
133                 print "Rango que se adicionara %0.3f" % float(match.group(1))
                    return float(match.group(1))
135             else:
                    print "Failure"
137
    #Creacion y envio de mensajes DIO
139     def DIO_message(self,addr,sk):
            Option=pack('b',2)
141         Rank=pack('H',self.rank)
            DIO=Option+Rank
143         print "ENVIANDO DIO a:",addr
            sk.sendto(DIO,(addr,self.UPPORT))
145
HOST='10.42.0.45'#poner tu ip
147 rpl_nodo=RPL_nodo(HOST)
rpl_nodo.config()
149
while True:
151     time.sleep(1)
        pass
```

códigos 6.2: Código empleado para un nodo

6.3. Código nodo sensor

```
1  import socket as s
   from thread import start_new_thread
3  from struct import*
   import time
5  from subprocess import Popen, PIPE
   import re
7  import json
   import sys
9  import Adafruit_DHT

11 class RPL_nodo:
       UPPORT=6646
13     TCPPORT=7646
       broadcast_ip='10.42.0.255'
15
       def __init__(self,mi_ip):
17         self.mi_ip=mi_ip
           self.padre={}
19         self.rank=0
           #self.version=0
21         #self.id=0
           #self.DODAGID=1
23         self.down={}
           self.vecinos=[]
25
       def config(self):
27         sock=s.socket(s.AF_INET,s.SOCK_DGRAM)
           sock.setsockopt(s.SOL_SOCKET,s.SO_REUSEADDR,1)
29         sock.setsockopt(s.SOL_SOCKET,s.SO_BROADCAST,1)
           sock.bind(('',self.UPPORT))
31         time.sleep(3)
           start_new_thread(self.recibir_udp,(sock,))
33         time.sleep(15)
           start_new_thread( self.DIS_message , ( sock , ) )
35         time.sleep(15)
```

```
        start_new_thread( self.sensar , (sock, ) )
37      time.sleep(15)

39  #Creacion y envio de mensajes DIS
    def DIS_message(self,st):
41      while True:
            opcion=1
43          Option=pack('b',opcion)
            mensaje=Option
45          print "####################ENVIANDO DIS#########################"
            st.sendto(mensaje,(self.broadcast_ip,self.UPPORT))
47          time.sleep(10)

49  def recibir_udp(self,sr):
        while True:
51          data,addr=sr.recvfrom(1024)
            vecino=[addr[0]]
53
            if addr[0]!=self.mi_ip:
55              if addr[0] not in self.vecinos:
                    self.vecinos.append(addr[0])
57
                opt=unpack('b',data[0:1])[0]
59              if opt==1 and addr[0]!=self.mi_ip:
                    print "DIS recibido de:",addr[0]
61                  if self.rank==0:
                        print "No formo parte de la red RPL:"
63                  else:
                        self.DIO_message(addr[0],sr)
65
                elif opt==2:
67                  print"DIO recibido de:",addr[0]
                    rango=unpack('H',data[1:3])[0]
69                  print "Rango de DIO recibido:",rango

71                  if self.rank==0:
                        self.rank=self.rank+rango+int(100*self.OF(vecino))
73                      print "Mi rango es:",self.rank
                    #else:
75                  #    if self.padre.keys()==[]:
                        print "Yo soy tu padre:",addr[0]
77                      self.padre.update({addr[0]:rango})
                        padre=self.padre.keys()
79                      self.sensar(padre[0],sr)

81                  elif self.padre.values()>[rango] and addr[0] not in self.padre.
    keys():
                        print "Yo soy tu nuevo padre:",addr[0]
83                      self.padre.clear()
                        self.padre.update({addr[0]:rango})
85                      if self.down=={}:
                            print "No hay rutas de bajada"
87                      else:
                            #ENVIAR DAO
89                          self.DAO(addr[0],sr)
                    elif self.rank<rango and addr[0] != self.mi_ip:
91                      print "Es ruta de bajada:",addr[0]
                        self.down.update({addr[0]:rango})
93                      print "Mis rutas de bajada",self.down
                        self.DAO(addr[0],sr)
95
                elif opt==3:
97                  print "DAO recibido de:",addr[0]
                    DAO=data[1:len(data)]
99                  rutas=eval(DAO)
                    print "ACULIZANDO RUTAS DE BAJADA..."
101                 self.down.update(rutas)

103             elif self.padre.keys()!=[]:
                    padre=self.padre.keys()
105                 self.DAO(addr[0],sr)
                    self.sensar(padre[0],sr)
107             else:
```

```
                    break
109
    #Creacion y envio de mensajes DAO
111 def DAO(self,addr,sk):
        print "ENVIANDO DAO..."
113     Option=pack('b',3)
        DAO=Option+str(self.down)
115     sk.sendto(DAO,(addr,self.UPPORT))

117 #Funcion para obtener rango del nodo
    def OF(self, lista):
119     vecinos=lista
        for i in range (0,len(vecinos)):
121         comando='ping -W 1 -c 3'+' '+vecinos[i]
            cmd = Popen(comando.split(' '), stdout=PIPE)
123         output = cmd.communicate()[0]
            print output
125         match = re.search('(\d+\.\d+)\/(\d+\.\d+)\/(\d+\.\d+)\/(\d+\.\d+)\s+ms',
    output)
            print match
127         if not (match is None):
                #print float(match.group(1))
129             print "Rango que se adicionara %0.3f"%float(match.group(1))
                return float(match.group(1))
131         else:
                print "Failure"
133
    #Creacion y envio de mensajes DIO
135 def DIO_message(self,addr,sk):
        Option=pack('b',2)
137     Rank=pack('H',self.rank)
        DIO=Option+Rank
139     print "ENIVIANDO DIO a:",addr
        sk.sendto(DIO,(addr,self.UPPORT))
141
    #Funcion para sensar y enviar al padre
143 def sensar(self, padre, sk):
        while True:
145         print "Tomando muestras de temperatura y humedad"
            humidity, temperature = Adafruit_DHT.read_retry(11,4)
147         Option=pack('b',4)
            Temp=pack('f',temperature)
149         Humedad=pack('f',humidity)
            Ip=pack('I',unpack("!I",s.inet_aton(self.mi_ip))[0])
151         datos_sensados=Option+Temp+Humedad+Ip
            print "Enviando muestras a mi padre"
153         sk.sendto(datos_sensados,(padre,self.UPPORT))

155 HOST='10.42.0.50'#poner tu ip
    rpl_nodo=RPL_nodo(HOST)
157 rpl_nodo.config()

159 while True:
        time.sleep(1)
161     pass
```

códigos 6.3: Código empleado para el nodo sensor

6.4. Código sinkhole

```
1  import socket as s
   from thread import start_new_thread
3  from struct import*
   from subprocess import Popen, PIPE
5  import re
   import time
7
   class RPL_border:
9      UPPORT=6646
```

```
    TCPPORT=7646
11  broadcast_ip='10.42.0.255'

13  def __init__(self,mi_ip):
        self.mi_ip=mi_ip
15      self.vecinos=[]
        self.rank=-1
17      self.id=0
        self.down={}
19
    def config(self):
21      sock=s.socket(s.AF_INET,s.SOCK_DGRAM)
        sock.setsockopt(s.SOL_SOCKET,s.SO_REUSEADDR,1)
23      sock.setsockopt(s.SOL_SOCKET,s.SO_BROADCAST,1)
        sock.bind(('',self.UPPORT))
25      time.sleep(3)
        start_new_thread(self.recibir_udp,(sock,))
27      time.sleep(15)
        start_new_thread( self.DIS_message , ( sock , ) )
29      time.sleep(15)

31  #Creacion y envio de mensajes DIS
    def DIS_message(self,st):
33      while True:
            opcion=1
35          Option=pack('b',opcion)
            mensaje=Option
37          print "####################ENVIANDO DIS#########################"
            print "Mi rango",self.rank
39          st.sendto(mensaje,(self.broadcast_ip,self.UPPORT))
            time.sleep(10)
41


43  def recibir_udp(self,sr):
        while True:
45          data, addr=sr.recvfrom(1024)
            opt=unpack('b',data[0:1])[0]
47
            if addr[0]!=self.mi_ip:
49              if addr[0] not in self.vecinos:
                    self.vecinos.append(addr[0])
51
            if opt==1 and addr[0]!=self.mi_ip:
53                  print "DIS recibido de:",addr[0]
                    if self.rank==-1:
55                      print "No formo parte de la red RPL:"
                    else:
57                      print "Enviando DIO a:", addr[0]
                        self.DIO_message(addr[0],sr)
59                      if self.down=={}:
                            if self.rank>0:
61                              self.rank=0
                            else:
63                              break
                        else:
65                          print "Enviando DIO a:", addr[0]
                            self.DIO_message(addr[0],sr)
67          elif opt==2:
                print "DIO recibido de:", addr[0]
69              rango=unpack('H',data[1:3])[0]
                print "Rango de DIO recibido:", rango
71              print "Autoasignando rango optimo ..."
                self.rank=0
73          print "Mi rango ahora es",self.rank

75          elif opt==3:
                print "DAO recibido de:",addr[0]
77              DAO=data[1:len(data)]
                rutas=eval(DAO)
79              print "ACULIZANDO RUTAS DE BAJADA..."
                self.down.update(rutas)
81              print "Rutas de bajada:",self.down
```

```
83              elif opt==4:
                    ip_fuente=s.inet_ntoa(pack("!I",unpack("I",data[9:13])[0]))
85                  temperatura=unpack('f',data[1:5])[0]
                    humedad=unpack('f',data[5:9])[0]
87                  print "DATOS SENSADOS RECIBIDOS DE:",ip_fuente
                    print "Temperatura: {0:0.1f} C Humedad: {1:0.1f}%".format(temperatura,
    humedad)
89
    #Creacion y envio de mensajes DIO
91  def DIO_message(self,addr,sk):
        Option=pack('b',2)
93      Rank=pack('H',self.rank)
        DIO=Option+Rank
95      print "ENIVIANDO DIO a:",addr
        sk.sendto(DIO,(addr,self.UPPORT))
97
99
101 HOST='10.42.0.59'
    rpl_border=RPL_border(HOST)
103 rpl_border.config()

105 while True:
        time.sleep(1)
107     pass
```

códigos 6.4: Código empleado para el ataque sinkhole

Bibliografía

[1] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks." RFC 6550, Mar. 2012.

[2] L. Wallgren, S. Raza, and T. Voigt, "Routing attacks and countermeasures in the rpl-based internet of things," *International Journal of Distributed Sensor Networks*, vol. 9, no. 8, p. 794326, 2013.

[3] P. Pongle and G. Chavan, "A survey: Attacks on rpl and 6lowpan in iot," in *2015 International Conference on Pervasive Computing (ICPC)*, pp. 1–6, Jan 2015.

[4] P. O. Kamgueu, E. Nataf, and T. D. Ndie, "Survey on rpl enhancements: A focus on topology, security and mobility," *Computer Communications*, vol. 120, pp. 10–21, 2018.

[5] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003.*, pp. 113–127, May 2003.

[6] L. Wallgren, S. Raza, and T. Voigt, "Svelte: Real-time intrusion detection in the internet of things," in *Ad Hoc Networks*, vol. 11, pp. 2661–2674, Nov 2013.

[7] K. Weekly and K. Pister, "Evaluating sinkhole defense techniques in rpl networks," in *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pp. 1–6, Oct 2012.

[8] A. Dvir, T. Holczer, and L. Buttyan, "Vera - version number and rank authentication in rpl," in *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, pp. 709–714, Oct 2011.

[9] H. Perrey, M. Landsmann, O. Ugus, M. Wählisch, and T. C. Schmidt, "Trail: Topology authentication in rpl," in *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, EWSN '16, (USA), p. 59–64, Junction Publishing, 2016.

[10] J. Deogirikar and A. Vidhate, "Security attacks in iot: A survey," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pp. 32–37, Feb 2017.

[11] A. Mayzaud, R. Badonnel, and I. Chrisment, "A taxonomy of attacks in rpl-based internet of things," 2016.

[12] D. Kaur and P. Singh, "Various osi layer attacks and countermeasure to enhance the performance of wsns during wormhole attack," *International Journal on Network Security*, vol. 5, no. 1, p. 62, 2014.

[13] A. Wahid and P. Kumar, "A survey on attacks, challenges and security mechanisms in wireless sensor network," *International Journal for Innovative Research in Science and Technology*, vol. 1, no. 8, pp. 189–196, 2015.

[14] I. Andrea, C. Chrysostomou, and G. Hadjichristofi, "Internet of things: Security vulnerabilities and challenges," in *2015 IEEE Symposium on Computers and Communication (ISCC)*, pp. 180–187, IEEE, 2015.

[15] M. I. Abdullah, M. M. Rahman, M. C. Roy, *et al.*, "Detecting sinkhole attacks in wireless sensor network using hop count," *Int. J. Comput. Netw. Inf. Secur*, vol. 3, pp. 50–56, 2015.

[16] I. Kenji, T. Matsunaga, K. Toyoda, and I. Sasase, "Secure parent node selection scheme in route construction to exclude attacking nodes from rpl network," *IEICE Communications Express*, vol. 4, no. 11, pp. 340–345, 2015.

[17] C. Cervantes, D. Poplade, M. Nogueira, and A. Santos, "Detection of sinkhole attacks for supporting secure routing on 6lowpan for internet of things," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 606–611, 2015.

[18] A. Le, J. Loo, K. Chai, and M. Aiash, "A specification-based ids for detecting attacks on rpl-based network topology," *Information*, vol. 7, p. 25, May 2016.

[19] O. Arana, F. Garcia, and J. Gomez, "Analysis of the effectiveness of transmission power control as a location privacy technique," *Computer Networks*, vol. 163, p. 106880, 2019.

[20] M. Nawir, A. Amir, N. Yaakob, and O. B. Lynn, "Internet of things (iot): Taxonomy of security attacks," in *2016 3rd International Conference on Electronic Design (ICED)*, pp. 321–326, Aug 2016.