UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y REDES DE COMUNICACIÓN

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERÍA EN ELECTRÓNICA Y REDES DE COMUNICACIÓN

TEMA:

"IMPLEMENTACIÓN DE UN CANAL DE COMUNICACIÓN CIFRADO ADAPTADO A UNA ARQUITECTURA SDN PARA REDES DE SENSORES INALÁMBRICAS 6LOWPAN PARA MEJORAR LA SEGURIDAD DE LA INFORMACIÓN"

AUTOR: ANTHONY RODRIGO GUZMÁN ABALCO

DIRECTOR: MSc. CARLOS ALBERTO VÁSQUEZ AYALA

IBARRA-ECUADOR

2022

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DEL CONTACTO			
Cédula de Identidad	172796109-4		
Apellidos y Nombres	Guzmán Abalco Anthony Rodrigo		
Dirección	Cayambe, calle Colombia y Rumiñahui		
E-mail	arguzmana@utn.edu.ec		
Teléfono Fijo	N.A.	**Teléfono Móvil**	0994220479
DATOS DE LA OBRA			
Título	"IMPLEMENTACIÓN DE UN CANAL DE COMUNICACIÓN CIFRADO ADAPTADO A UNA ARQUITECTURA SDN PARA REDES DE SENSORES INALÁMBRICAS 6LOWPAN PARA MEJORAR LA SEGURIDAD DE LA INFORMACIÓN"		
Autor	Anthony Rodrigo Guzmán Abalco		
Fecha	08 de junio de 2022		
SOLO PARA TRABAJOS DE GRADO			
Programa	■ Pregrado	☐ Posgrado	
Título por el que se aspira:	Ingeniero en Electrónica y Redes de Comunicación		
Director	Msc. Carlos Alberto Vásquez Ayala		

CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de esta y saldrá en defensa de la Universidad Técnica del Norte en caso de reclamación por parte de terceros.

Ibarra, a los 8 días del mes de junio de 2022

EL AUTOR:

.....

Anthony Rodrigo Guzmán Abalco

C.I.:172796109-4

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

INGENIERÍA EN ELECTRÓNICA Y REDES DE COMUNICACIÓN

CERTIFICACIÓN

MsC. CARLOS ALBERTO VÁSQUEZ AYALA, DIRECTOR DEL PRESENTE TRABAJO DE TITULACIÓN

CERTIFICA:

Que, el presente trabajo de Titulación: **"IMPLEMENTACIÓN DE UN CANAL DE COMUNICACIÓN CIFRADO ADAPTADO A UNA ARQUITECTURA SDN PARA REDES DE SENSORES INALÁMBRICAS 6LOWPAN PARA MEJORAR LA SEGURIDAD DE LA INFORMACIÓN",** ha sido desarrollado por el señor Anthony Rodrigo Guzmán Abalco bajo mi supervisión.

Es todo en cuanto puedo certificar en honor a la verdad.

.....

Ing. Carlos Alberto Vásquez.

CI: 1002424982

DIRECTOR

AGRADECIMIENTOS

Primeramente, quiero agradecer a Dios por permitirme llegar a este momento a pesar de todas las dificultades; a mis padres por el esfuerzo y sacrificio que realizaron a lo largo de mi vida académica para que yo pueda llegar hasta este punto de mi vida que, aunque no fue fácil ellos estuvieron ahí para darme las fuerzas y animo necesario para continuar; a mis hermanos por ser uno de las motivaciones para culminar mi carrera y poder dar el ejemplo de que la perseverancia y paciencia al final tiene su recompensa; a mi tíos y abuelos que, de una u otra forma me ayudaron en todo el transcurso de esta etapa.

También debo agradecer a mi grupo de amigos (Diego, Santiago, El Brayan, Edwin y Deivid) del cual forme parte, por haber estado junto a mí en la universidad y hacer de la vida universitaria más entretenida en cualquiera lugar en el que nos hayamos encontrado. Debido a su ayuda, compañía y gracia es que estoy culminando una de mis etapas académica más importantes en mi vida por lo cual les tengo mucho cariño, aprecio y sobre todo respeto.

Por último, pero no menos importantes, a los docentes que me ayudaron a formar como profesional en mi paso por la carrera. A mi director el ing. Carlos Vázquez por su sabiduría, conocimientos y paciencia aportados durante todo este largo proceso que, aunque con varios tropiezos en el camino, supo orientarme y velar por mi progreso.

Al terminar esta etapa de la cual tengo tantos recuerdos (tanto buenos como malos) puedo decir que es un logro más en mi vida y no tengo nada más que gratitud con todos aquellos que se atravesaron en mi camino y aportar su granito de arena para que yo pueda llegar a este punto.

Anthony Rodrigo Guzmán Abalco

DEDICATORIA

El presente trabajo de titulación quiero dedicar a Dios, a mis padres y hermanos por apoyarme a lo largo de todos estos años y hasta el final de esta etapa educacional, pero sobre todo a mi madre que ha estado siempre pendiente de mí y de todo este proceso que, sin su ayuda y dedicación esto no podría ser posible.

A mis hermanos para que tengan el ejemplo y la motivación necesaria y persigan sus objetivos y metas personales a lo largo de toda su vida tanto académica como personal, pero sobre todo a mí mismo por tener perseverancia y paciencia para afrontar cada reto que se cruzó en mi camino y luchar hasta el final.

Anthony Rodrigo Guzmán Abalco

RESUMEN

El presente trabajo de titulación consiste en la implementación de un mecanismo de encriptación en el envío de información de un sistema de una red de sensores inalámbricos (WSN) y las redes definidas por software (SDN) mediante un ambiente de pruebas por simulación y bajo el protocolo de comunicación 6LoWPAN conocido como SD6WSN. El sistema propuesto consta de un controlador de red y ocho nodos sensores que se ejecutan en el simulador de redes Cooja para evidenciar el funcionamiento y comportamiento de los nodos inalámbricos y el controlador.

El ambiente de pruebas, el mecanismo criptográfico y el sistema operativo ligero se escogieron adecuadamente en base a la norma ISO/IEC/IEEE 29148 para garantizar la compatibilidad y operatividad del sistema. Para recrear un ambiente realista, se diseñó, creó y ejecutó una topología en malla de ocho nodos y un controlador de red con direccionamiento IPv6, en los cuales se implementó la función de encriptación de envió de datos, que posteriormente fueron capturados y analizados mediante un sniffer de red para evidenciar el funcionamiento del sistema.

El rendimiento y funcionalidad del sistema se evalúa a través de varias pruebas realizadas mediante el ambiente de pruebas. Para verificar que existe una mejora en la seguridad, las pruebas son ejecutadas en dos escenarios: previa y posterior implementación de la encriptación. El sistema se evaluó mediante: la captura de tráfico y análisis de la carga útil, latencia de la red y tasa de recepción de paquetes, consumo de energía y consumo de memorias RAM y ROM.

ABSTRACT

This degree work consists of the implementation of an encryption mechanism for the send of information on a system of Wireless Sensor Network (WSN) and Software Defined Networking (SDN) through testing environment and under the 6LoWPAN communication protocol known as SD6WSN. The proposed system consists of a network controller and eight sensor nodes that are executed in the Cooja network simulator to show the operation and behavior of the wireless nodes and controller.

The testing environment, encryption mechanism and light operating system were chosen based on the ISO/IEC/IEEE 29148 to guarantee the compatibility and operativity of the system. To recreate a realistic environment, an eight-node mesh topology and a network controller with IPv6 addressability were designed, built, and implemented, which were later captured and analyzed by means of a network sniffer to evidence the operation of the system

The performance and functionality of the system is evaluated through various tests carried out through the test environment. To verify that there is a improve in the security, the tests are executed in two scenarios: before and after implementation of the encryption. The system is evaluated by: traffic capture and payload analysis, network latency and packet reception rate, power consumption, and RAM and ROM memory consumption.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

CAPÍTULO I: INTRODUCCIÓN

1.1 PROBLEMA

Actualmente, el mundo de las redes de comunicación ha ido evolucionando a pasos grades, esto debido al constante desarrollo de nuevas tecnologías y productos tecnológicos; pero todos ellos al estar conectados a la red necesitan poseer un direccionamiento para enviar y recibir información. El agotamiento del direccionamiento en versión IPv4 fue inminente debido al constante incremento de dispositivos, esta noción viene aumentando en parte al Internet de las Cosas (IOT) y las redes 5G que se han venido desarrollando con los años y cada vez tienen mayor fuerza, por ello que el contar con un direccionamiento es imprescindible (Lin, Wang, Cheng, & Liu, 2017). El protocolo 6LoWPAN resuelve en parte este problema de direccionamiento y trae consigo algunas otras dificultades como la latencia, velocidad de transferencia, perdida de paquetes y la seguridad. En lo que se refiere a este último punto, hay que tener en cuenta que, al ser dispositivos inalámbricos, la información en tránsito se encuentra vulnerable ante posibles ataques (Castillo Merchán, 2016).

Por lo mencionado anteriormente, una de las principales causas de estas dificultades es la naturaleza inalámbrica de comunicación que poseen, lo que la hace sensible a intrusos. Con la ayuda del protocolo IPv6 y las mejoras que trae consigo, se refuerza su seguridad, pero al tomar en cuenta que se encuentra en una red de múltiples nodos y canales de comunicación, los datos en ocasiones se vuelven vulnerables. El surgimiento de Las Redes Definidas por Software se dio por la necesidad de mejorar la arquitectura de las redes tradicionales, prometiendo una mejora en cuanto a su escalabilidad y flexibilidad en la gestión de la red (Carrasco, 2019), son un nuevo paradigma en el mundo de las redes informáticas permitiendo aumentar nuevas características.

La nueva propuesta que ha salido al mercado es la integración de las redes SDN para solucionar problemas de IoT y WSN (Miguel, Jamhour, Pellenz, & Penna, 2018). Por ello, mediante simulación se plantea el uso de redes SDN para mejorar la seguridad de los canales de comunicación mediante la implementación de un mecanismo de seguridad como el cifrado del canal. El cifrado es uno de los varios métodos de encriptación de la información que existen, pero estos no están diseñados para las comunicaciones WSN y menos para redes SDN. Toda la seguridad WSN se obtiene del mismo protocolo 6LoWPAN, lo cual no es suficiente. Habiendo varios métodos de cifrado actualmente en la red es necesario buscar alguno que ofrezca la seguridad que necesitan los entornos WSN y adecuarlo para estas redes inalámbricas de forma que pueda funcionar sobre entornos SDN para 6LoWPAN y ofrezca la seguridad requerida. El controlador SDN será quien maneje la comunicación entre los nodos, permitiendo aumentar este nuevo mecanismo de seguridad. Con ello se aumentará la confiabilidad de los datos que se envían de un nodo a otro que se levantará sobre una plataforma de software libre.

Con la ayuda del protocolo de internet IPv6 se tiene mejoras significativas, pero en las redes de sensores aún queda varios parámetros que mejorar y que ya se lo ha ido haciendo(Baddeley, Nejabati, Oikonomou, Sooriyabandara, & Simeonidou, 2018). La información que se maneja a través de las redes de comunicación es muy importante dependiendo la finalidad, pero eso no impide que este expuestas a riesgos de ataques o falsificación de información y esto algunos estándares no logran resolver con facilidad. Los métodos de seguridad actuales todavía no se encuentran adaptados para entornos SDN en su totalidad para solucionar problemas de seguridad en las WSN por ser una tecnología reciente y no se conoce todo su potencial. El uso de redes SDN para mejorar la seguridad entre los canales de comunicación es un paso más a lo que puede ofrecer la virtualización y la centralización de las funciones.

1.2 OBJETIVOS

1.2.1 OBJETIVO GENERAL

- Implementar un canal de comunicación cifrado adaptado a una arquitectura SDN para redes de sensores inalámbricas 6LoWPAN para mejorar la seguridad de la información

1.2.2 OBJETIVOS ESPECÍFICOS

- Recopilar información de la base bibliográfica, documental y del estado del arte de las redes SDN, las WSN, el protocolo IPv6 para IoT y el protocolo IEEE 802.15.4.
- Determinar los tipos de mecanismos de cifrados de datos que existen, y cuales se acoplan mejor con las redes de sensores Inalámbricos para soluciones en IoT para dispositivos de bajo rendimiento.
- Diseñar e integrar la arquitectura que contenga el controlador SD6WSN, los nodos y, los canales de comunicación con una red IoT-SDN con dispositivos basados en software libre mediante simulación.
- Configurar el controlador SD6WSN dentro de la arquitectura diseñada con los elementos de red y los nodos para la convergencia de internet e implementar el mecanismo de seguridad cifrado sobre el controlador SDN para la comunicación entre los nodos.
- Ejecutar pruebas experimentales de la comunicación de los nodos de comunicación WSN para evaluar confiabilidad de los datos al implementar el cifrado acondicionado para SDN.

1.3 JUSTIFICACIÓN

Al aumentar las infraestructuras de red como servicio y el continuo avance de la tecnología 5G se hace necesario que se cuente con un plan de direccionamiento para los dispositivos que cada día salen al mercado y de igual forma para la tecnología IOT que cuenta con el protocolo 6LoWPAN que es el protocolo IPv6 a redes de bajo consumo (Lin et al., 2017). El protocolo 6LoWPAN resuelve varios de los obstáculos que presentan los dispositivos de bajo consumo como con las que se cuenta en las WSN, pero aún quedan presentes algunos otros. La red definida por software (SDN) ofrece una arquitectura flexible y escalable separa la toma de decisiones de los dispositivos y proporciona una plataforma de red programable. Las redes de Internet de las cosas (IoT) de baja potencia, donde las arquitecturas de múltiples inquilinos y de múltiples aplicaciones requieren soluciones escalables y configurables, se encuentran en una posición ideal para su estudio (Baddeley et al., 2018).

De igual forma, al encontrarnos en un ambiente en donde las redes avanzan y cambian constantemente, las amenazas evolucionan de igual forma, y de una u otra forma se busca vulnerar algún tipo de seguridad para filtrar, modificar o extraer algún tipo de información. Las redes tradicionales se vuelven rígidas lo que no permiten realizar cambios de forma sencilla cuando y solo dependen del hardware o de los protocolos intrínsecos que vienen por defecto en algunos programas. Las SDN presentan un nuevo paradigma del Networking (Transaction & Traffic, 2014) permitiendo adaptarse a cualquier red y centralizando las funciones de control pudiendo añadir o descartar nuevas funciones para adaptarse a las necesidades de la red.

En este ámbito se ha avanzado en otros campos tal como menciona (Miguel et al., 2018), en su trabajo se implementa una red SDN donde se centraliza la funciones de red haciendo disponible una vista unificada lógica de la topología en los controladores con que puede resolver de una forma práctica las dificultades implícitas de los sensores inalámbricos, como lo son sus enlaces poco confiables, alta latencia, pérdida de paquetes, su poca potencia de procesamiento, su

limitada memoria y su flexibilidad con respecto a cambios de política. Se observa que el trabajo investigativo está enfocado a resolver los problemas más comunes de los sensores WSN mediante las características que ofrece una red SDN gracias a la separación de sus planos de control y datos, pero, aspectos como la seguridad y la privacidad son tratados con menor importancia. (Carrasco, 2019) utiliza la implementación de redes SDN para IoT con el fin de manifestar los múltiples aportes y ventajas que se puede tener en varios aspectos como: la administración y gestión, la vista unificada, la heterogeneidad tanto de red como de dispositivos, control de flujo, movilidad y su big data. Por otra parte, (Baddeley et al., 2018) muestra los desafíos que se enfrenta al implementar una red SDN sobre las limitaciones que presentan las redes inalámbricas y que el tráfico del controlador no solo está sujeto a fluctuaciones debido a enlaces poco confiables y contención de red, sino que la sobrecarga generada por SDN puede afectar severamente el rendimiento de otro tráfico y presentan μSDN (SDN liviano) y la interoperabilidad del protocolo de enrutamiento subyacente, así como la optimización de una serie de elementos dentro de la arquitectura SDN para reducir la sobrecarga de control a niveles prácticos.

Estos trabajos investigativos resuelven los problemas de una red IoT o WSN típica y la abordan desde un punto de vista técnico, ya que, si bien resuelven los principales obstáculos mejorando una red WSN típica, aún dejan abierto varias posibilidades que no se han abordado con prioridad en otras investigaciones como la seguridad y privacidad, su escalabilidad, su interoperabilidad y su programabilidad.

Por tal razón, debido a que la seguridad en este tipo de redes ha sido tratada en profundidad, en el presente anteproyecto de tesis, se busca incursionar en este campo. Con un enfoque de estudio de seguridad mediante el diseño y la implementación de un mecanismo de cifrado de comunicación de canal para una arquitectura SDN para 6LoWPAN en un ambiente virtual, que contará con nodos de comunicación, un controlador que centralizará las funciones del plano de

control y sobre el cual se implementará el mecanismo de cifrado teniendo un sistema de comunicación sobre el cual se realice un envío de datos seguro y confiable sobre un canal.

1.4 ALCANCE

Mediante el estudio de los diferentes métodos de cifrado que existen se plantea seleccionar y adaptar uno de estos métodos de seguridad a SDN para las Redes Inalámbricas de Sensores (WSN) bajo el estándar IEEE 802.15.4 o como se lo conoce 6LoWPAN, para garantizar la fiabilidad de la información que se genera en los nodos bajo posibles ataques.

En primera instancia se realizará un estudio bibliográfico para establecer bases teóricas en las que se sustente el proyecto a desarrollarse en cuanto se refiere a la importancia de las SDN y características, las WSN, el protocolo IPv6 y el estándar IEEE 802.15.4.

Se realizará el estudio de los distintos mecanismos de cifrado de seguridad desarrollados para los ambientes IoT. Se evaluará en sus características, funcionamiento y los tipos que existen, como resuelven los problemas de privacidad, confiabilidad de los datos y cual se puede adaptar mejor con las redes SDN para su modificación e implementación en la arquitectura con varios nodos de comunicación. Este mecanismo de cifrado se implementará posteriormente en el controlador de la red.

Se procederá a la evaluación de las herramientas de simulación que existen en el mercado para escenarios virtuales que incorporen dispositivos IoT. Posteriormente, se procede al diseño de la red SDN en que se incorpore una red de nodos comunicados entre sí teniendo como nodo central el controlador, que centralizará la comunicación y controlará el enrutamiento de los paquetes de datos. Esta SDN contará con una arquitectura típica con una capa de control compuesta por el controlador y una capa de reenvío compuesta por un conjunto de dispositivos de conmutación. Las aplicaciones de red se conectan al controlador para implementar algoritmos de red. Los nodos son dispositivos WSN típicos que se extienden con las siguientes funciones de comunicación: enrutamiento de borde, reenvío de plano de datos y enrutamiento de plano de control. Las funciones de comunicación y las aplicaciones de detección están

coordinadas por el agente SD6WSN, que interactúa con el coordinador del controlador a través del protocolo del plano de control. Posteriormente se configura el controlador al contar con la arquitectura levantada, se procede a verificar la comunicación entre los distintos nodos, para a continuación diseñar el mecanismo de seguridad cifrado para las WSN mediante SDN e implementarlo sobre el controlador para contar con encriptación en la comunicación los cuales contarán con un sistema de autenticación de certificados entre nodos de extremo a extremo. El cifrado se levantará sobre el canal de comunicación entre el nodo central y los nodos secundarios de comunicación sobre una red inalámbrica que pueda ser aplicado a cualquier tipo de red inalámbrica basada 6LoWPAN.

Finalmente, se realizará la integración de todos los elementos que componen la red WSN y se procederá a las pruebas de funcionamiento en un ambiente simulado, realizando la comunicación entre nodos mediante el envío de información que pasará por el controlador central previamente a implementar el mecanismo de seguridad cifrado para comprobar el estado actual del funcionamiento y su comunicación. Posteriormente se implementará y del mecanismo de seguridad seleccionado para la comunicación de los canales entre los nodos y se evaluará el tipo de cifrado del canal mediante la comparación de la comunicación antes y después de implementar la seguridad en el controlador central.

CAPITULO II: MARCO TEÓRICO

En este capítulo se establece la base conceptual de planteamientos que se desarrollaran a lo largo del proyecto, profundizando en temas referidos a: Redes de Sensores Inalámbricas (WSN), 6LoWPAN, Redes Definidas por Software (SDN) y mecanismos criptográficos.

En la primera sección, se explica el concepto de redes IoT, arquitecturas, tecnologías de comunicación y seguridad. Se describe las tecnologías de acceso inalámbrico más utilizadas en soluciones IoT, detallando a IEEE 802.15.4 y sus distintas versiones que, según (Hanes, Salgueiro, Grossetete, Barton, & Henry, 2017): "son la base de la tecnología de comunicación inalámbrica restringidas", que posteriormente se adaptan con otros protocolos hasta llegar a 6LoWPAN. Se detalla las características principales de 6LoWPAN; la capacidad de las cabeceras de compresión, fragmentación y direccionamiento que mejoran su funcionamiento para redes restringidas y las amenazas a la seguridad más comunes que enfrenta. Por último, se explica y compara los sistemas operativos (SO) existentes que operan con dispositivos restringidos de la tecnología IEEE 802.15.4.

En la siguiente sección se explica el concepto, finalidad, funcionamiento y ventajas de SDN, respecto a las redes tradicionales; también se toma en cuenta que la tecnología SDN basa su funcionamiento en controladores virtuales, los cuales se analizan aquellos de código abierto; de esta manera se establece que, el cambio de paradigma entre tecnologías ofrece nuevas soluciones que pueden ser aplicadas a áreas como las redes IoT y WSN.

El objetivo de este capítulo es proporcionar un análisis técnico o teórico del funcionamiento de los elementos que intervienen en el proyecto, el cual integra una solución de comunicación inalámbrica segura basada en un sistema 6LoWPAN con SDN.

2.1 Internet de las Cosas (IoT)

La premisa básica y la principal meta de IoT es “conectar lo desconectado”, es decir, los objetos que aún no están conectados a una red como internet, sean conectadas y puedan comunicarse entre sí e interactuar con los usuarios y con otros objetos. IoT es una transición tecnológica donde los dispositivos permiten detectar y controlar el entorno a su alrededor al hacer los objetos “inteligentes” y conectarlos a través de una red (Hanes et al., 2017).

Cuando los objetos o máquinas pueden ser detectadas y controladas remotamente a través de una red, la integración entre el mundo físico y computadoras permite mejorar áreas de eficiencia, precisión, automatización y aplicaciones avanzadas. El mundo IoT es extenso, multifacético y que rápidamente puede tornarse en algo complicado ya que cuenta con dispositivos de distintos fabricantes, diferentes proveedores de servicios y con protocolos de comunicación diferentes, logrando así crear una red heterogénea. También, hay que tener en cuenta que estos dispositivos cumplen tareas específicas dentro de este concepto, por ejemplo: sensores, actuadores, microcontroladores, transceptores, etiquetas RFID y dispositivos de comunicación inalámbrica conectados a internet. Su función principal es tomar señales del entorno o interactuar con el mismo (Carrasco, 2019).

La razón principal de interconectar todo es usar la información que recoja cada elemento y procesarla, de acuerdo a lo mencionado por (Jiménes & Berrueco, 2016) “se logra así, obtener una retroalimentación que pueda ayudar a mejorar la sociedad en varios aspectos como: sanidad, seguridad, medioambiente, economía e incluso política”. En la actualidad, internet está diseñado para dar soporte a dispositivos como computadores, smartphones, tablets, etc, siendo los principales terminales de los usuarios para acceder a internet, aumentando considerablemente la cantidad de estos dispositivos en la última década (Otto, 2017).

Puesto que la cantidad de dispositivos que IoT alberga y su diversidad es amplia, es comprensible la razón de no contar con un estándar, protocolo o arquitectura estrictamente definido, debido a que dispone de una amplia variedad con múltiples soluciones (Rose, Eldridge, & Chapin, 2015).

El impacto que puede llegar a tener esta tecnología según (Hanes et al., 2017) "es que el mundo del internet debe estar listo para afrontar estos nuevos desafíos que se vienen para los próximos años". Son varias las adversidades a distintos niveles de complejidad que debe superar IoT, los cuales se describe en la Tabla 1.

Tabla 1

Desafíos que IoT afronta a futuro

Desafío	Descripción
Escalabilidad	Las empresas ahora trabajan con centenares de nodos de red, estos pueden multiplicarse rápidamente con la integración de soluciones inteligentes, teniendo impacto directo sobre la administración de la red que debe soportar este incremento y que pueden o no dificultar la comunicación entre sí.
Seguridad	Al igual que aumenta el número de "cosas" conectadas también aumenta el riesgo para IoT, aumentan las amenazas. Si un nodo es comprometido puede servir de punto de partida para atacar todo el sistema. La seguridad en IoT debe ser generalizada hacia cada faceta.
Privacidad	Uno de los dispositivos más comunes es el sensor que está presente en la vida recolectando datos, desde información acerca de la salud hasta transacciones comerciales. El manejo de esta información debe ser una las prioridades.
Big Data y Análisis de datos	La gran cantidad de sensores que IoT contiene, desencadena en una afluencia masiva de datos. El reto que afronta es, evaluar cantidades masivas de datos que llegan de diferentes fuentes en varias formas y hacerlo de forma oportuna.

Interoperabilidad	Así como otras tecnologías y protocolos, estos compiten por el mismo mercado y estandarización dentro de IoT. Algunos de estos estándares y protocolos son privados y otros son abiertos. IoT está ayudando a minimizar este problema, pero hay varios protocolos e implementaciones disponibles para IoT.

Fuente: (Hanes et al., 2017)

2.1.1 Arquitectura IoT

En el mundo de las redes de comunicación ya existen arquitecturas diseñadas para un propósito especifico, a estas redes se las conocen como redes tradicionales y funcionan para las Tecnologías de la Información (IT). Los sistemas IT están mayormente preocupados por la confiabilidad de los datos y el continuo soporte hacia las aplicaciones de negocios como los son correos electrónicos, páginas web, base de datos, etc. Los complejos requisitos que demanda IoT hace que los sistemas IT no sean adecuados para llevar a cabo algunas características propias de IoT. Debido a que IoT está relacionado con los datos generados por los sensores y como esos datos son usados, la esencia de las arquitecturas IoT radica en cómo los datos son recolectados, transportados, analizados y finalmente ejecutados (García García, 2015).

A continuación, se realiza un acercamiento a las diferencias entre las redes IT e IoT, con un enfoque en los requerimientos de cambios de arquitectura manejados por IoT.

Escalabilidad: la escala típica de una red IT es de los varios miles de dispositivos que comúnmente son: impresoras, dispositivos móviles inalámbricos, laptops, servidores, etc. El modelo tradicional de campus de tres capas: acceso, distribución y núcleo es entendido, pero ahora se debe considerar lo que pasa cuando la escala de red pasa de un par de miles a millones de dispositivos. Una red IT no está preparada para enrutar millones de puntos finales. Si se toma los requerimientos de escalabilidad, IPv6 es la base para la capa de red IoT (Hanes et al., 2017).

Seguridad: proteger los datos empresariales de intrusiones, es una de las principales funciones de un departamento de IT así lo señala Hanes et al. (2017), pero a pesar de todos los esfuerzos, los hackers aún encuentran la manera de vulnerar redes de confianza. En IT la primera línea de defensa es el firewall; sería impensable colocar puntos críticos fuera del firewall. Sin embargo, en IoT los puntos finales con frecuencia están en una red de sensores inalámbricos que usan un espectro de banda no licenciado y no solo son visibles para el mundo, además son físicamente accesibles.

Los modelos tradicionales de IT no están diseñados para nuevos vectores de ataques introducidos por la gran dispersión de sistemas IoT. Los sistemas IoT requieren mecanismos de autenticación, cifrado, técnicas de prevención de intrusión que entiendan el comportamiento de los protocolos y puedan responder a ataques sobre la infraestructura crítica (Hanes et al., 2017). Para la seguridad IoT los sistemas deben contar con los siguientes puntos:

- Debe permitir identificar y autenticar a todas las entidades involucradas en el servicio de IoT
- Asegurarse que toda la información de usuario que sea compartido entre puntos finales y back-end este encriptada
- Utilizar una plataforma de conectividad IoT y establecer reglas basadas en políticas de seguridad que puedan tomar acciones inmediatas si se detecta comportamiento anómalo de los dispositivos conectados

Redes y Dispositivos Restringidos: la mayoría de los dispositivos IoT, están diseñados para trabajos simples, inteligentes y comúnmente son baratos. Esto significa que tienen limitaciones de batería, CPU, memoria y deben transmitir solo cuando es necesario debido a que en los ambientes impredecibles donde usualmente son desplegados, las redes que proveen la conectividad también tienden a perder y soportar baja tasa de datos. Al contrario de las redes

IT que cuenta con velocidad de conexiones en gigabits y sus puntos finales son potentes CPU's que en caso de tener problemas de conectividad la solución sería simplemente mejorar la velocidad de la red y si hubiese demasiados dispositivos la solución sería escalar en las VLAN's tanto como sea necesario. En IoT estos enfoques no servirían debido a su misma naturaleza, requiriendo de una nueva variedad de tecnología de acceso que puedan atender las limitaciones de conectividad y cantidad (Hanes et al., 2017).

Datos: los dispositivos IoT generan una cantidad enorme de datos. Al contrario de los sistemas IT que no se preocupan tanto por la información generada por sus dispositivos, en IoT estos datos son muy valiosos para los negocios permitiendo entregar nuevos servicios IoT que mejoran la experiencia del cliente, reduciendo costos y abriendo nuevas oportunidades. Aunque la mayoría de los datos generados son caóticos, la visión de estos proviene a través del análisis (Hanes et al., 2017).

También se menciona que los retos IoT han impulsado una disciplina completamente nueva de arquitectura de red. En los últimos años las arquitecturas de red han emergido para afrontar los retos de diseños de red IoT a escala masiva. El concepto fundamental de estas arquitecturas es soportar todos los datos, procesos y funciones de los dispositivos finales realizan. A continuación, se describe dos arquitecturas que han surgido para las redes IoT:

La Arquitectura Estandarizada IoT World Forum (IoTWF): el modelo presentado por el Foro Mundial de IoT ofrece una perspectiva limpia y simplificada sobre IoT e incluye computación de vanguardia, almacenamiento de datos y acceso. Proporciona una forma precisa de visualizar IoT desde una perspectiva técnica. Cada una de las siete capas se divide en funciones específicas y la seguridad abarca todo el modelo como se observa en la Figura 1.

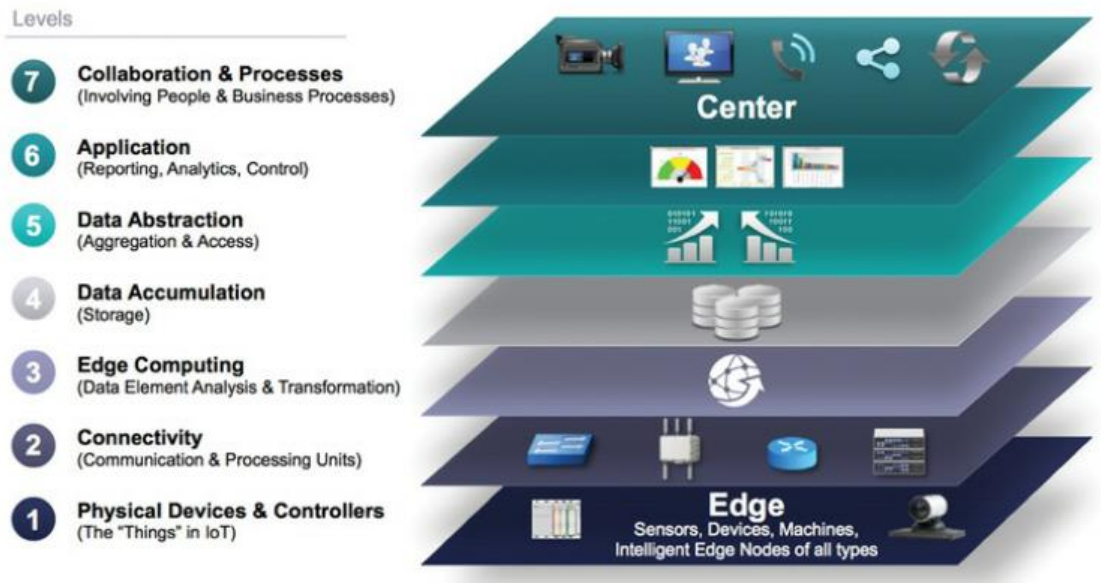


Figura 1. Modelo referencial del IoTWF.

Fuente: (Hanes et al., 2017)

A continuación, se describe la función de cada uno de los siete niveles del IoTWF.

- **Layer 1:** la primera capa del modelo de referencia de IoT es la capa de controladores y dispositivos físicos. Su principal función es generar los datos y ser consultado o controlado a través de una red.
- **Layer 2:** la segunda capa está enfocada está en la conectividad. La función más importante de esta capa de IoT es la transmisión confiable y oportuna de datos.
- **Layer 3:** la tercera capa hace énfasis en la reducción de datos y la conversión de los flujos de datos en información que esté preparada para ser procesada por las capas superiores. Otra de sus funciones es la evaluación de los datos para verificar si se pueden filtrar o agregar antes de enviarlos a una capa superior; admitiendo que los datos se vuelvan a formatear o decodificar, logrando que el procesamiento por otros sistemas sea más fácil.
- **Layer 4-7:** las capas superiores manejan y procesan los datos de IoT generados por la capa inferior.

La Arquitectura Estandarizada OneM2M IoT: el objetivo de oneM2M es crear una capa de servicios común, que pueda integrarse fácilmente en dispositivos de campo permitiendo la comunicación con los servidores de aplicaciones. De acuerdo a lo mencionado por (Ojo, Adami, & Giordano, 2016) "desde el punto de vista arquitectónico, varios componentes deben trabajar juntos para que una red IoT esté operativa". Estos componentes están divididos en una arquitectura de tres capas y se puede apreciar en la Figura 2. El núcleo de un stack funcional de IoT está comprendido por la capa de red, la capa de servicios y la capa de aplicaciones.

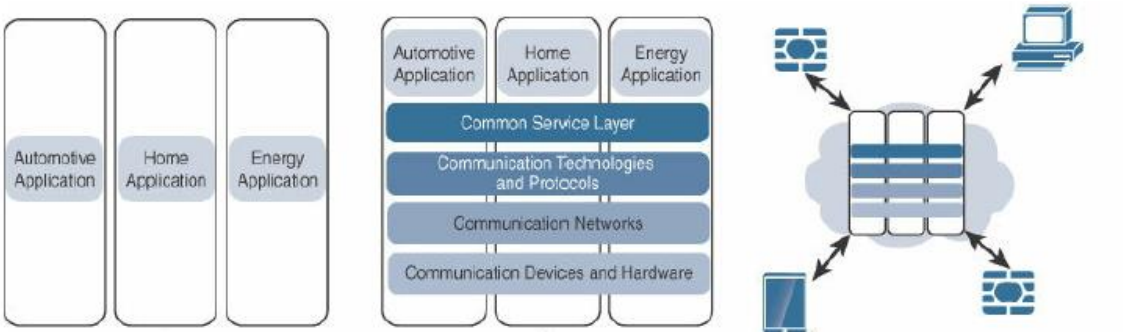


Figura 2. Arquitectura IoT oneM2M.

Fuente: (Hanes et al., 2017)

El marco de trabajo de oneM2M se centra en los servicios, aplicaciones y plataformas de IoT. Los mayores retos que supone IoT es la interoperabilidad de dispositivos heterogéneos; al M2M desarrollar una arquitectura horizontal permite una comunicación en todos los niveles de IoT. Esta arquitectura divide las funciones de IoT en tres capas:

- **Capa Aplicación:** la arquitectura M2M da un gran cuidado a la conectividad entre dispositivos, sus aplicaciones y servicios demandados por el usuario. Las aplicaciones tienden a ser específicas de la industria con sus propios conjuntos de modelos de datos, que se muestran como entidades verticales. También incluye servidores e infraestructura en la nube para servicios en tiempo real.

- **Capa de Servicios:** en esta capa, los módulos horizontales incluyen la red física donde se ejecutan las aplicaciones IoT, los protocolos de administración subyacentes y el hardware. Esta capa conceptual agrega API y middleware que admiten servicios y aplicaciones de terceros. Según (Ojo et al., 2016) menciona, "la puerta de enlace de IoT es uno de los componentes más importantes en la arquitectura de IoT que actúa como un puente entre la capa aplicación y la red central".
- **Capa de Red:** este es el dominio de comunicación para los dispositivos IoT y puntos finales como sensores, actuadores que recolectan, transmiten o realizan alguna acción especifica. En esta capa, se encuentra la infraestructura de comunicaciones e incluye tecnologías inalámbricas como: IEEE 802.15.4, IEEE 801.11ah.

2.1.2 Redes de Sensores

Una red de sensores/actuadores SANET como se las conoce, es capaz de medir u operar en su ambiente. Son capaces de comunicarse y cooperar entre sí de forma efectiva. Estas redes pueden conectarse de manera cableada o inalámbrica, como resalta (Hanes et al., 2017) "el hecho que los SANET se encuentren comúnmente en el 'mundo real' significa que tiene un cierto nivel de flexibilidad para su implementación". En la Tabla 2, se presenta las ventajas y desventajas que ofrece una red de sensores inalámbricos.

Tabla 2

Ventajas y desventajas de una SANET

Ventajas	Desventajas
Flexibilidad de despliegue	Menor seguridad
Fácil aumento de número de nodos.	Disminución de velocidades de transmisión
Implementación de bajo costo	Alto impacto sobre el entorno de despliegue.
Fácil mantenimiento a largo plazo	

Menor esfuerzo para aumentar un nuevo nodo
Adaptable a cambios de topología.

(Hanes et al., 2017)

2.1.2.1 Redes de Sensores Inalámbricos (WSN)

Una red de sensores inalámbricos (WSN) hace referencia a una cantidad de pequeños dispositivos inteligentes y autónomos, los cuales se conocen como nodos sensores distribuidos físicamente alrededor de una locación para monitorear, recolectar, almacenar y enviar información dentro de una red inalámbrica (Cheng & Ye, 2007).

Son dispositivos que constan de un microcontrolador, una fuente de energía (en su mayoría una batería), un transceptor con una antena interna y un sensor. Los componentes que forman una WSN cumplen con funcionalidades especificas las cuales se detallan a continuación:

- Sensores: son de distintos tipos y tecnologías usados para recolectar información del entorno y los transforman en señales eléctricas.
- Nodo de sensor: toman los datos del sensor y los envían a las estaciones base.
- Gateway: es la interconexión entre una red de sensores y una red TCP/IP
- Estación Base: recolector de datos
- Red Inalámbrica: basada en el estándar 802.15.4 (Cheng & Ye, 2007)

Una WSN tiene varias características tales como: tolerancia a fallos, los nodos tienen la habilidad de hacer frente a los fallos; heterogeneidad de nodos, todos los nodos pueden ser de diferentes fabricantes; energy harvesting, los nodos con baterías tienen restricción en el uso de energía; durabilidad, resistencia a las condiciones ambientales; escalabilidad, despliegue de grandes redes de nodos.

Debido a la limitada vida de sus baterías, estos dispositivos por lo general se construyen con una característica que les permite conservar energía cuando no están trabajando que se conoce como modo descanso o sleep (Cheng & Ye, 2007). Es en base a estas limitaciones que las WSN son diseñadas, desplegadas y utilizadas.

Hanes et al. (2017) señala que introducir un gran número de dispositivos permite la adopción de jerarquías dentro de la red. Una de las ventajas de este tipo de organización es el poder agregar la lectura de otros nodos que estén cercanos uno del otro. Si bien existen ciertos casos en que los nodos envían datos constantemente, la mayoría de los nodos inteligentes conectados inalámbricamente generalmente se atienen a estos dos casos:

- **Periodicidad:** la transmisión de información ocurre solo en intervalos periódicos.
- **Eventualidad**: la transmisión solo ocurre cuando el sensor detecta una actividad en particular o supera un umbral determinado.

La decisión de cual esquema de comunicación es el más adecuada depende de la aplicación. En la Figura 3, se describe las aplicaciones en que se usa las WSN.

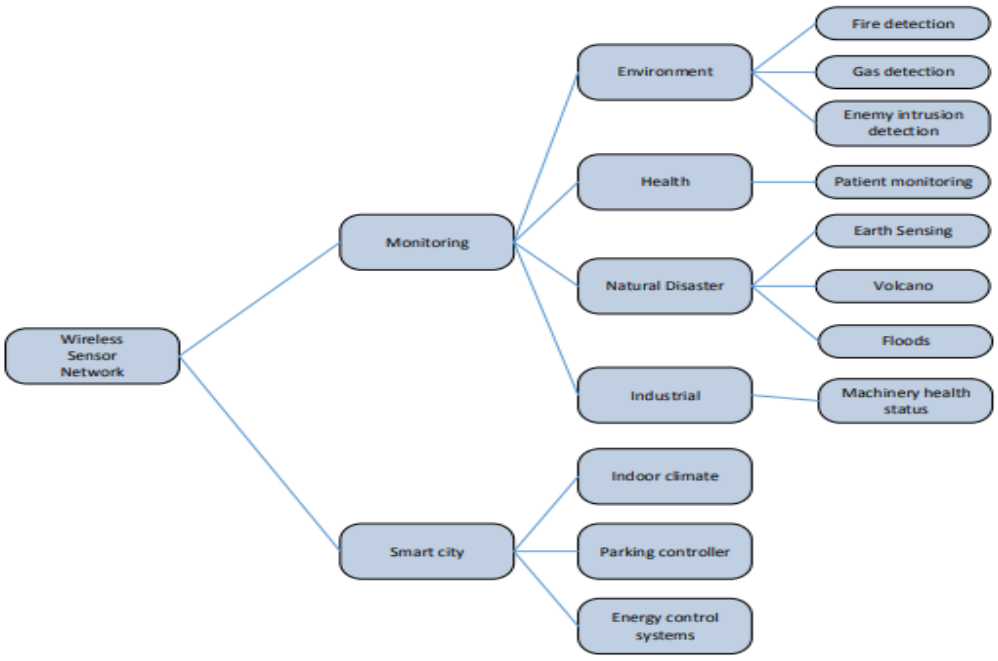


Figura 3. Aplicaciones de las WSN.

Fuente: (Fernández, Villalba, & Kim, 2018)

La mayoría de las aplicaciones de las redes de sensores inalámbricos es para detectar fenómenos en áreas específicas y requieren ser monitoreadas. Un ejemplo de esto son los sensores de gases, fuego u otras sustancias químicas. Otra de las aplicaciones son las Smart Cities en donde se las usan para registrar la cantidad de energía usada o controlar acciones en una habitación como temperatura, luces, puertas, etc. (Fernández et al., 2018).

Una de las más grandes aplicaciones es en el sector de la salud, donde, pueden ayudar a los pacientes. Este tipo de dispositivos se dividen en dos grupos: implantados y utilizables. Los implantados son aquellos que tienen implementados en el cuerpo para monitorear su salud periódicamente. Sin embargo, los utilizables no son solo para controlar problemas de salud, también es manejado en el estado físico, que es un área en crecimiento por el interés de las personas en los deportes (Fernández et al., 2018). Estos dispositivos son usados en la superficie del cuerpo humano o cerca de los órganos vitales.

En la investigación de Fernández et al. (2018) muestra que, la principal ventaja de los sensores inalámbricos frente a otros es su versatilidad para capturar datos en diferentes locaciones o diferentes puntos de una ciudad. Una WSN también puede ser ubicada en un bosque para detectar prematuramente los inicios de un incendio antes que sea aún más peligroso. En el sector industrial las WSN son muy importantes para el monitoreo.

Retos de las Redes de Sensores Inalámbricos

Los principales retos que afrontan estas tecnologías se los describe a continuación:

- En WSN el descubridor de red es un problema, debido a que la topología de red tiene que ser construida y actualizada en tiempo real y cada vez que se añade algún nuevo nodo o algún nodo falle.

- Controlar la red en función de cada nodo, debido a que estos cambian dinámicamente dependiendo de su energía, ancho de banda, etc.
- El enrutamiento es otro reto que tienen que afrontar las WSN si se lo compara con IP routing.
- Las señales colaborativas y el procesamiento de señales requieren una comunicación entre nodos y al tiempo no perder ninguna señal durante la transmisión.
- Finalmente, uno de los principales retos es la seguridad en su red debido a que una WSN es desplegada comúnmente en ambientes hostiles por tanto su exposición a ataques físicos.

De acuerdo con (Fernández et al., 2018), menciona que para que un sistema WSN sea seguro todos los componentes que lo conforman tienen que ser seguros y en caso que uno no lo sea, ese se convierte en un punto crítico de ataque. Para que un sistema sea seguro, este debe contemplar varios aspectos a cumplir: autenticación, privacidad, comunicación robusta, enrutamiento seguro, establecimiento de confianza, etc. En cuanto a la implementación de una clave compartida, es de las soluciones más sencillas, pero en caso de que la clave sea interceptada toda la comunicación se ve comprometida. La solución más optima es utilizar seguridad entre nodos mediante una llave publica encriptada. Así, tanto emisores como receptores dentro de una red WSN tienen que autenticarse primero en la red para poder enviar o recibir información.

La seguridad de la información es uno de los principales focos de atención en la comunicación, debido a que se maneja información de toda clase, que va desde información irrelevante como la fecha o temperatura ambiental hasta información muy sensible que puede contener datos personales, ubicación exacta, etc.

2.1.2.2 Topología de una red WSN

La cantidad de datos a transportar durante un periodo de tiempo junto con el consumo de energía, determinan las limitaciones de movilidad y alcance de una estructura de celda inalámbrica. Alcances similares no significan topologías similares, esto depende de la tecnología utilizada, la estructura de conectividad de la red puede ser flexible para extender las posibilidades de comunicación (Carrasco, 2019). Se cuentan con dos modelos de topología.

- **Topologías punto a punto**: esta topología permite a un punto comunicarse con otro punto. Esto haría que cada punto pueda comunicarse solo con una única puerta de enlace. Pero hay ciertos casos en que punto a punto se refiere a la estructura de comunicación que a la topología física en sí.
- **Topologías Punto-Multipunto:** esta topología hace referencia a que un nodo puede comunicarse con más de un punto. La mayoría de la tecnología IoT que cuenta con más de una puerta de enlace entra en esta categoría. Una de las características de las redes IoT, dependiendo del tipo de nodo, pueden funcionar de forma diferente; unos nodos sensores puede recolectar y enviar datos, mientras que otros nodos puertas de enlace recopilan información y pueden ordenar al sensor para realizar acciones específicas o interactuar con otras puertas de enlace.

Las tecnologías categorizan a los nodos dependiendo de las funciones de implementación. Un ejemplo de categorización es la tecnología 802.15.4 que se basa en sensores móviles pequeños que recopilan datos y los envían para formar una red de igual a igual. Sin embargo, en ocasiones hay dispositivos que recolectan más datos que los demás. En ese caso, el punto de control está en el centro de la red y la red forma una topología en estrella, con el punto de control en el centro y los sensores en los radios. En esta configuración el punto central está a cargo de la coordinación general de la red y las transmisiones de cada sensor. En el estándar IEEE 802.15.4 este punto central se denomina coordinador; el objetivo es que cada sensor se

comunique con el coordinador creando una relación maestra/esclavo. El sensor puede implementar un subconjunto de funciones de protocolo para realizar solo una parte especializada (comunicación con el coordinador). Tal dispositivo se llama Dispositivo de Función Reducida (RFD). Un RFD no puede ser un coordinador, tampoco puede implementar directamente comunicaciones a otra RFD (Hanes et al., 2017).

El coordinador de función completa se llama Dispositivo de Función Completa (FFD) que puede comunicarse directamente con una o más FFD formando múltiples conexiones punto a punto. Como se puede observar en la Figura 4 y de acuerdo a (Hanes et al., 2017):

> "Las topologías en que cada FFD tiene una ruta única a otra FFD se denominan topologías de árbol de clúster. Los FFD en el árbol de clúster pueden tener RFD, dando como resultado una topología en estrella de clúster"

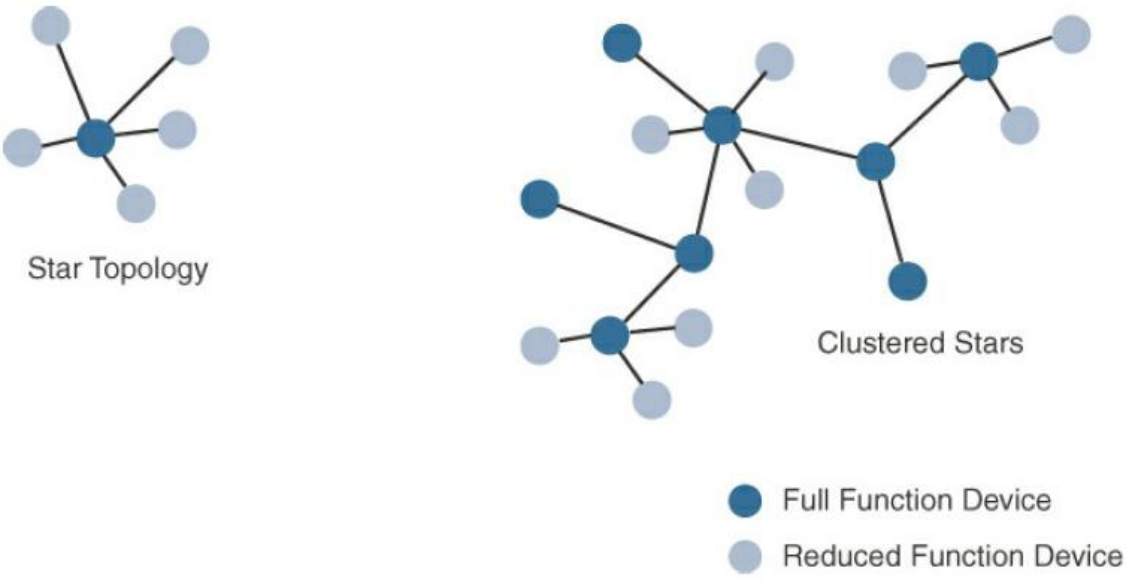


Figura 4. Topologías en estrella y en clúster.

Fuente: (Hanes et al., 2017, p. 106)

2.1.2.3 Seguridad en una red WSN

Las vulnerabilidades en la seguridad son fallas básicas en el desarrollo e implementación, como plantea (Castillo Merchán, 2016) y han ido creciendo principalmente por la exposición

de información de los sistemas informáticos. Los ataques digitales representan más del 10%. El objetivo de los atacantes es vulnerar tres elementos fundamentales de la seguridad: confidencialidad, integridad y disponibilidad. En consecuencia, los atacantes intentan comprometer una red para encontrar vulnerabilidades que puedan aprovechar.

Para preservar la seguridad en la información hay varios parámetros a tomar en consideración, para ello, hay que comprender que el sistema es una función en donde actúan un transmisor y un receptor (Tanenbaum & Wetherall, 2012). A continuación, se presenta una breve introducción a los requisitos que se debe considerar necesarios para que una red pueda considerarse segura:

Confidencialidad: para que una red sea considerada confidencial, la información que posea un sistema computacional solo debe ser accedida por el sistema o sistemas autorizados para su lectura o modificación. Que un sistema no autorizado pueda acceder o interceptar la información, como se aprecia en la Figura 5, es una amenaza a la confidencialidad, aumentando la probabilidad que la red sea comprometida dependiendo del medio físico usado para la comunicación o los elementos intermedios.

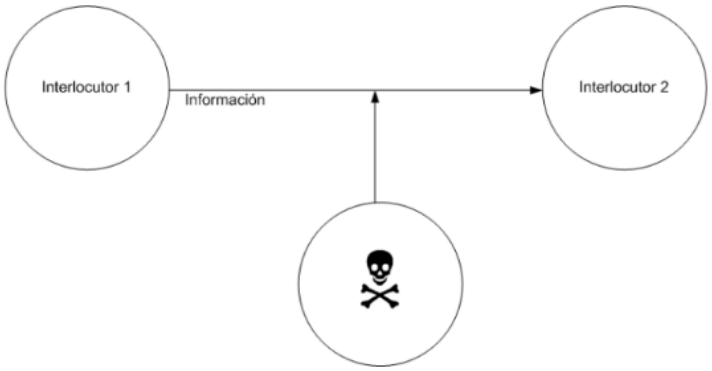


Figura 5. Amenaza a la confidencialidad.

Fuente: (Cañete, 2015)

Integridad: todos los elementos del sistema deben ser gestionados por agentes autorizados. Es muy importante que el mensaje sea recibido exactamente como se lo envió de forma tal que su contenido no sea alterado en camino al destino. La amenaza que compromete la integridad son modificaciones a los mensajes, que pueden ser realizados por terceros sin autorización a la información en tránsito. Como se muestra en la figura 6, esta amenaza depende que el atacante tenga fácil acceso al canal de comunicación.

Figura 6. Amenaza a la integridad de la información.

Fuente: (Cañete, 2015)

Autenticidad: esto permite identificar y asegurar que los agentes que participan en la comunicación sepan que el agente que transmite sea quien dicen ser y que el receptor tenga la certeza que la información que recibe es legítima y no esta adulterada de alguna forma. La Figura 7, ilustra de forma gráfica este concepto.

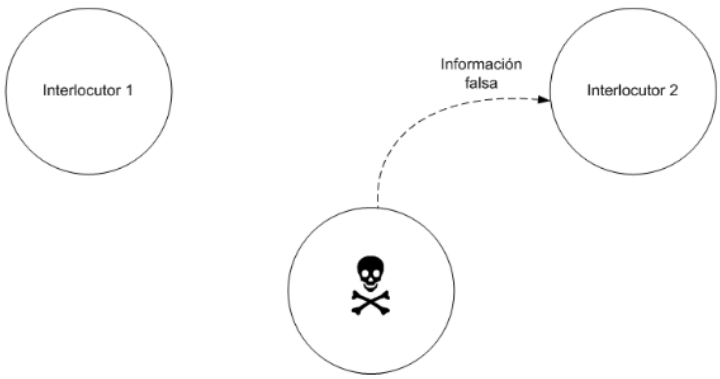


Figura 7. Amenaza a la autenticidad.

Fuente: (Cañete, 2015)

Disponibilidad: garantiza que los nodos de comunicaciones se encuentren disponibles para los grupos autorizados en los momentos que sean necesarios, es decir, se encuentren funcionales aún después de sufrir un ataque de denegación de servicios (DoS). Este requisito es amenazado por la interrupción de las comunicaciones que puede ser sobre el medio, sobre los agentes o los elementos intermedios, tal y como se aprecia en la Figura 8. En los nodos sensores pueden tener diferentes objetivos en las diferentes capas.

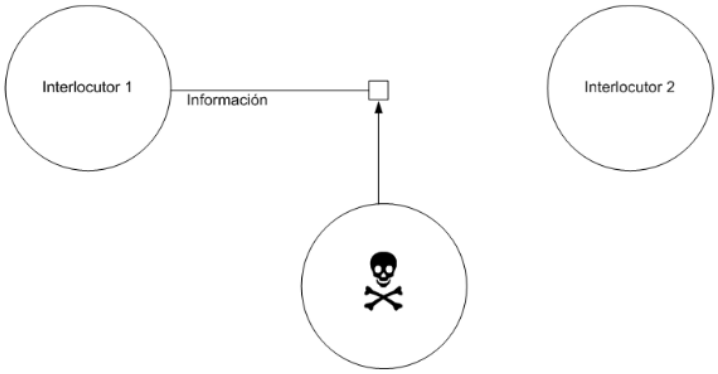


Figura 8. Amenaza a la disponibilidad.

Fuente: (Cañete, 2015)

Algoritmos de Cifrado para WSN

Los algoritmos de cifrado se dividen en varias categorías que siguen los problemas de la seguridad principales como lo son la confidencialidad, autenticación y la integridad. Tal y como menciona (Stephen Evanczuk, 2020). A continuación, se describe brevemente los grandes grupos de codificación y su funcionamiento en las comunicaciones.

Algoritmo de clave simétrica: el algoritmo o el cifrado utiliza la misma clave para cifrar el mensaje legible por humanos (texto sin formato) en una versión protegida (texto cifrado) y luego descifrar el texto cifrado de nuevo a texto sin formato (Tanenbaum & Wetherall, 2012).

La criptografía de clave simétrica se suele utilizar para garantizar la confidencialidad. Los algoritmos de cifrado simétrico más comunes incluyen:

- Triple DES (Data Encryption Standard), también conocido como 3DES, o como oficialmente es llamado por el Instituto Nacional de Estándares y Tecnología de los Estados Unidos (NIST): el Triple Algoritmo de Cifrado de Datos (TDEA).
- Algoritmos Estándar de Cifrado Avanzado (AES), como el AES-256, que utiliza claves de 256 bits (Stephen Evanczuk, 2020).

Algoritmos de clave asimétrica: donde el cifrado utiliza un conjunto emparejado de claves privadas y públicas para cifrar y descifrar un mensaje que, por lo general, forman parte de protocolos de seguridad más amplios para el acuerdo de claves y las firmas digitales. Los cifrados asimétricos suelen utilizarse para garantizar la confidencialidad, la autenticación o el no repudio (Tanenbaum & Wetherall, 2012). Los algoritmos de cifrado más comunes son:

- Algoritmo de firma digital del Estándar Federal de Procesamiento de Información (FIPS),
- Algoritmo de RSA (Rivest-Shamir-Adleman)
- Algoritmo de firma digital de curva elíptica (ECDSA).

Algoritmos avanzados de cifrado ligero para dispositivos IoT: de acuerdo con el NIST, la criptografía ligera es una subcategoría de la criptografía que tiene como fin proveer soluciones para el rápido crecimiento de aplicaciones que en general son empleados por nodos restringidos.

Un sistema de criptografía convencional puede funcionar bien en computadores, servidores y hasta en teléfonos móviles. Pero, dispositivos como etiquetas RFID, dispositivos de detección, redes de sensores y hasta sistemas embebidos no cuentan con los mismos recursos. Estos dispositivos y redes requieren plataformas con algoritmos de criptografías ligeras las

cuales pueden ser aplicables a WSN, RFID, Wireless Body Area Network (WBAN), IoT, tarjetas inteligentes, etc. (Singh, Sharma, Moon, & Park, 2017). Existen diferentes tipos de criptografías ligeras primitivas para nodos restringidos y cada una se basa en las características de su funcionamiento como su tamaño, bloque, longitud, numero de rondas y estructuras.

Cifrado de bloque ligero: el cifrado de bloques ligero es un cifrado simétrico en el que todo el bloque se procesa a la vez. Los cifrados en bloque se utilizan para diseñar funciones hash y códigos de autenticación de mensajes (MAC). Se basan en dos tipos de estructuras: Red de permutación alternativa (SPN) y Feistel. La estructura de Feistel usa su función redonda solo en la mitad de los estados. Ayuda a diseñar el mismo circuito de cifrado y descifrado con una sobrecarga mínima. Por lo tanto, la principal ventaja de la estructura feistel es que los procesos de cifrado y descifrado utilizan el mismo código de programa. Conduce a un bajo uso de memoria (Singh et al., 2017). El marco de Feistel no es adecuado para un diseño de baja latencia. SPN será más rápido, pero no se requiere programación clave. La falta de programas clave lo hará vulnerable a los ataques Los principales parámetros para evaluar cifrados de bloques ligeros son el tamaño de la clave, el tamaño del bloque, el tipo de estructura y el número de rondas (Dhanda, Singh, & Jindal, 2020).

AES: El Estándar de Encriptación Avanzada (AES) es uno de los más famosos y fue optimizado para hacerlo más liviano y que funciona con una estructura SPN (Red de Permutación Sustituida). El tamaño de bloque de cifrado es de 128 bits, permitiendo opciones de tres tamaños de clave, siendo el tamaño de clave quien determine el número de rondas del algoritmo. Así, un tamaño de clave de 128 necesita 10 rondas, un tamaño de clave 192 requiere 12 rondas y un tamaño de clave de 256 requiere 14 rondas (Dhanda et al., 2020). Debido a que el aumentar el tamaño de la clave implica aumentar el procesamiento y almacenamiento en el dispositivo, el tamaño de clave de 128 bits sería el más adecuado para un dispositivo de bajos recursos y el cual será considerado en el alcance del presente trabajo.

SPECK: fue propuesto por (Beaulieu et al., 2015), en la familia de cifrados ligeros por bloques, diseñado con flexibilidad, permitiendo diferentes tamaños de bloques y claves. Fue desarrollado para operar mejor sobre microcontroladores de forma más eficientemente. SPECK está basado en la estructura de cifrado por bloques Feistel, la cual cuenta con la ventaja de usar el mismo algoritmo para cifrar y descifrar la información reduciendo la carga general al dispositivo. Tiene dos ramas, cada una de las cuales ha sido desarrollada en ambos sentidos, con importantes cambios en ambos sentidos, adición modular y XOR bitwise. La función de ronda de SPECK depende de la estructura Feistel y la siguiente formula, Rk (X, Y), la cual se presenta en la Ec. 1 .

$$\boldsymbol{Rk}\,(\boldsymbol{X},\boldsymbol{Y}) \;=\; [(\boldsymbol{S^{-a}}\,\boldsymbol{x}+\boldsymbol{y}) \,\oplus\, \boldsymbol{k},\, \boldsymbol{S^{-b}}\,\boldsymbol{y} \,\oplus\, (\boldsymbol{S^{-a}}\,\boldsymbol{x} \,+\, \boldsymbol{y}) \,\oplus\, \boldsymbol{k}] \qquad \text{Ec1}$$

Los parámetros de valor de RK (X, Y) se ven afectados por a y b, si n=16 entonces a=7 y b es 2, de lo contrario a es 8 y b es 3. Observe que, al describir la función de ronda SPECK, como el SPECK2n/mn se puede describir como SPECK con el tamaño de bloque de 2*n bits y m*n bit es el tamaño de la clave. p.ej. SPECK32/64, el de 16 bits es el tamaño de la palabra que se llama (n), el de 32 bits es el 2*n y el número de palabras es 4 que se indica como m. Como resultado, la clave total es m*n, que equivale a 64. Para obtener más detalles sobre SPECK, se remite al lector a la referencia (Beaulieu et al., 2015).

Speck cuenta con varias características y configuraciones que se pueden implementar para no generar una sobrecarga sobre los dispositivos. Se debe notar que, Speck tiene varias configuraciones (Tabla 3) que dependen de las capacidades del hardware y software sobre el que trabaja.

Tabla 3

Diferencias de parámetro del algoritmo Speck

Tamaño de bloque (bits)	Tamaño de clave (bits)	Numero de rondas	Tipo de estructura	α	β
32	64	22	Feistel	7	2
48	27	22	Feistel	8	3
	96	23	Feistel	8	3
64	96	26	Feistel	8	3
	128	27	Feistel	8	3
96	96	28	Feistel	8	3
	144	29	Feistel	8	3
128	96	32	Feistel	8	3
	128	33	Feistel	8	3
	255	34	Feistel	8	3

Fuente: Autoría

Como muestra en la tabla anterior, Speck varia en los tamaños de bloque, clave y de ronda, de hecho, Speck generalmente se denota por *Speck2n/mn* donde *2n* es el tamaño del bloque, $n \in 16, 24, 32, 48, 64$ y *mn* se asemeja al tamaño de la clave utilizada donde $m \in 2, 3, 4$ dependiendo de la seguridad deseada. Speck procesa la entrada como dos palabras. Como menciona (Farzaneh, List, Lucks, & Wenzel, 2013), al inicio de cada ronda la palabra de la izquierda del estado es rotada por α bits a la izquierda, antes de que la palabra de la derecha se agregue al módulo 2^n. Luego, a una clave de ronda K^i se le realiza un proceso XOR a la mitad izquierda. A continuación, la palabra derecha es rotada por β bits a la derecha, antes de que la palabra izquierda se haga XOR a la derecha. El procedimiento es descrito visualmente por la Figura 9.

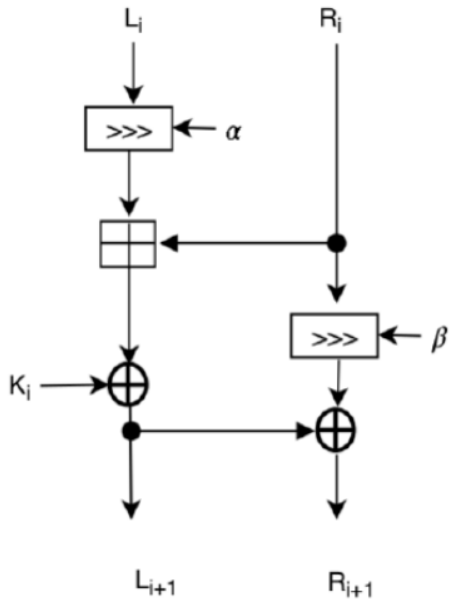


Figura 9. Representación general de la ronda del cifrado Speck.

Fuente: (Farzaneh et al., 2013)

Es de considerar que mientras más grande los tamaños de bloque, mayor procesamiento requiere su implementación. La Figura 10 presenta un diagrama visual del funcionamiento del cifrado de Speck 64/96, donde sigue el proceso anteriormente descrito repitiéndose 26 veces lo que equivale al número de rondas.

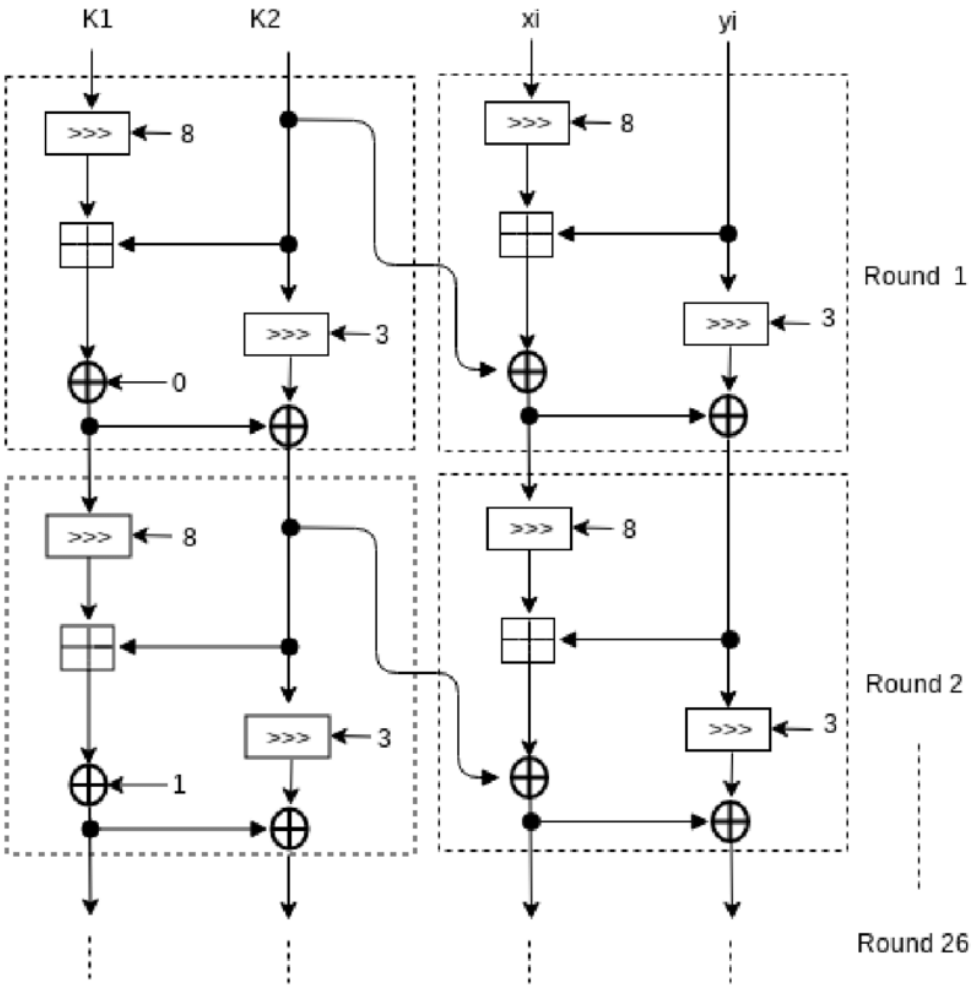


Figura 10. Diagrama de cifrado Speck 64/96.

Obtenido de: (Lama Sleem & Raphaël Couturier, n.d.)

SIMON: Pertenece a la misma familia de SPECK que funciona con la estructura Feistel diseñado por la agencia de seguridad nacional (NSA). Sus operaciones de cifrado usan simples funciones de ronda bit a bit de AND, OR y desplazamientos circulares a la izquierda. Esta desarrollado para mejorar el comportamiento en hardware y dar buenos resultados de cómputo criptográficos. Su función de ronda depende de dos etapas del mapa Feistel como se muestra en la Ec.2 a continuación:

$$\boldsymbol{Rk(X,Y)} \;=\; (\boldsymbol{y} \oplus \; \boldsymbol{f(x)} \; \oplus \; \boldsymbol{k,x}) \qquad \text{Ec.2}$$

Donde *k* es la ronda de clave y la Ec.3 muestra f(x):

$$f(x) = (Sx \& S^8 x) \oplus S^2 x \quad \text{Ec.3}$$

Funciones hash ligeras*:* las funciones hash ligeras son otra forma de brindar seguridad. Crean un "resumen de mensaje" de longitud fija a partir de un mensaje de longitud arbitraria. Este "resumen del mensaje" se utiliza para garantizar la integridad de los datos transmitidos. La huella de una función hash está determinada por el número de bits de estado, el tamaño de la lógica funcional y de control utilizada en la función redonda. Se caracteriza por:

- Tamaño de salida más pequeño: la gran cantidad de tamaño es fundamental para aplicaciones que requieren resistencia a colisiones con función hash. Aplicaciones en que no se necesite resistencia a colisiones puede usarse tamaños de interior y equilibrio, pero en los casos donde son necesarias las funciones hash, esta tarea debe tener una seguridad similar contra ataques de preimagen, segunda imagen y de impacto.
- Tamaño de mensaje más corto: la capacidad hash tradicional se utiliza para mejorar la contribución del enorme tamaño de aproximadamente 264 bits. Para la mayoría de las reglas objetivas con respecto a las funciones hash delgadas, el tamaño normal de la información es mucho más pequeño (256 bits como máximo). De esta forma, para aplicaciones ligeras, una función hash mejorada para mensajes cortos puede ser más razonable (Singh et al., 2017).

Sistema de alto rendimiento: el sistema elite utiliza un motor encriptado específico para cumplir tres necesidades: rendimiento, adaptabilidad y la seguridad. Otras limitaciones (como el área y la potencia) se consideran en menor medida. A continuación, se analizará algunos de los requisitos para lograr un sistema de alto rendimiento.

- CPU personalizada: los procesadores de cifrado (como CPU y ALU de cifrado) utilizan CPU optimizadas para ejecutar algoritmos de cifrado. La arquitectura del conjunto de instrucciones (ISA) generalmente integra instrucciones orientadas a contraseñas.

Debido a la diversidad de algoritmos de cifrado, es difícil elegir este tipo de instrucciones. Para usar la nueva declaración, el programa del sistema debe actualizarse a algo similar a un compilador

- Coprocesador criptográfico: la actualización de la velocidad de cifrado se completa con el módulo de equipo, que está dedicado al coprocesador de cifrado, el negocio de cifrado y está controlado por el procesador principal. El procesamiento de información regular hacia y desde el procesador c afecta la ejecución regular de datos.
- Matriz de Cifrado: utilizando la computación paralela, se desarrolló una matriz criptográfica de elementos de procesamiento y un procesador criptográfico de múltiples núcleos para mejorar aún más el rendimiento. Los arreglos cifrados están cerca de las tareas algorítmicas y requieren una topología de enrutamiento para mover datos entre los elementos de procesamiento y la memoria.
- Criptografía multinúcleo: por otro lado, los procesadores criptográficos de varios núcleos no se basan en algoritmos. Proporciona una velocidad de datos altamente encriptada o utiliza una encriptación diferente al mismo tiempo (Grand et al., 2011).

Cifrado de flujo ligero: Otra categoría de cifrado ligero es el de flujo. En un cifrado de flujo, los bits "r" se cifran y descifran a la vez. cifrados de flujo también fomentan el uso de la situación original en situaciones forzadas. El concurso eSTREAM, resuelto por la Red europea de excelencia en criptografía, tiene como objetivo identificar nuevos números de tráfico que pueden ser adecuados para una adopción ilimitada. Los cifrados de flujo ligeros importantes son Grain, Mickey y Trivium, estos son finalistas de la fase 3 del proyecto e-STREAM.

Dispositivos de bajos recursos y métricas de rendimiento: en el algoritmo criptográfico ligero, al considerar las métricas de rendimiento en dispositivos de bajos recursos, existe una interposición entre el rendimiento y el recurso para alcanzar los mismos niveles de seguridad. El rendimiento de la transmisión puede basarse en el consumo de energía, el tiempo de espera

o el tiempo de espera y el rendimiento. En esta sección, hay dos categorías para implementar el cifrado en dispositivos de bajos recursos, como se describe a continuación.

- Implementación y métricas específicas de software: la implementación del software se logró ejecutando un código criptográfico en el procesador. El código puede ser dependiente de la máquina (ensamblado) o independiente (java). Por ejemplo, si el código está escrito en lenguaje C, puede implementarse en un procesador específico. Generalmente, un sistema funciona con microcontroladores de bajo costo de 8 o 16 bits. Para un dispositivo de baja restricción, las implementaciones de software se centran en la cantidad de energía, velocidad y memoria utilizada. Las métricas específicas del software se centran en el número requerido de registros en RAM y ROM.
- Implementación y métricas específicas de hardware: **l**os recursos necesarios para el diseño y la implementación del hardware se expresan generalmente en términos de área de puerta y utilizan ASIC personalizado completo o FPGA. En FPGA, el diseño proporciona beneficios como minimizar los costos de desarrollo y aumentar la flexibilidad (Singh et al., 2017).

2.1.2.4 Protocolos de Comunicación

De acuerdo con Hanes et al. (2017) indica que las redes de sensores actualmente utilizan varios protocolos de comunicación para transmitir la información entre sus nodos. La elección de uno u otro protocolo depende de la función y las características de su aplicación. En unos casos se necesita mayor ahorro energético, en otros la fiabilidad de la comunicación, en otros que priorice la mayor cantidad de paquetes enviados, etc.

Entre todos los protocolos existentes que se utilizan con frecuencia para WSN se destacan los protocolos de bajo consumo como: ZigBee/6LoWPAN (IEEE 802.15.4), bluetooth (IEEE 802.15.1), Wi-Fi (IEEE 802.11) y Z-Wave. Ahora para aplicaciones determinadas, se está

propagando el uso de protocolos IP con conexión a internet los cuales necesitan mayores prestaciones en cuanto a consumo de energía y eficacia en la comunicación. Realizar una comparativa más profunda de protocolos de comunicación para medios inalámbricos es útil para elegir un protocolo para una aplicación determinada (Lee, Su, & Shen, 2007).

A continuación, se presenta una breve descripción de los protocolos mencionados:

- **Bluetooth Low Energy (BLE):** es una tecnología inalámbrica de corto alcance que opera en la capa física del modelo OSI y posee una velocidad de transmisión alta de datos y un mínimo de consumo de energía. Su arquitectura funciona de maestro/esclavo, y su limitante es el poder conectar pocos dispositivos. Permite interoperar pequeños dispositivos desarrollados para usar bluetooth y destinados a enviar pequeños paquetes de datos reducidos y opera en la banda libre de 2.4 GHz.
- **Wireless Fidelity (Wi-Fi):** protocolo de la capa enlace que tiene un consumo de energía muy alto y está basado en el estándar 802.11 que cuenta con varias versiones como la 802.11a, 802.11b, 802.11g, etc., que son incompatibles entre ellas y utiliza las bandas 2.4 GHz, 5 GHz y 60 GHz para tasas de transferencia en el orden de decenas o cientos de Mbps.
- **Z-Wave:** protocolo de baja potencia de la capa de enlace que cuenta con tecnología basada en RF y fue desarrollado para entornos de domótica a nivel doméstico. Tiene capacidad escalable y admite la topología en malla. Z-Wave permite transmisiones seguras de baja latencia y bajas tasas de velocidad de pequeños paquetes de datos hasta 100 kbps. Los dispositivos mantienen una distancia máxima de 100 metros a diferencia de otras redes como Wi-Fi y opera en el rango de frecuencia de 868 MHz y no tiene interferencia de otras tecnologías.
- **ZigBee:** es un protocolo de la capa de enlace que fue diseñado para aplicaciones a nivel doméstico e industrial orientado al control y monitoreo admitiendo varias topologías.

Se basa en el estándar IEEE 802.15.4 que opera en bandas de 2,4 GHz con tasas de transferencia de 255 Kbps y en 915 MHz con tasas de transferencia menores. La distancia adyacente puede ser de hasta 10 metros en aplicaciones de mínimo consumo de energía. Se enfoca en velocidades de transmisión limitadas de bajo coste y consumo de energía reducido.

- **6LoWPAN:** es el acrónimo de IPv6 over Low Wireless Personal Area Networks. Es la adaptación del protocolo IPv6 a redes de bajo consumo 802.15.4 y sus diferentes características cuyo objetivo es proveer una dirección IP versión 6 a todo dispositivo inalámbrico pudiendo direccionarse y accederse directamente. Debido a que 802.15.4 tiene una longitud de MTU limitado a 127 bytes, diferentes longitudes y bajo ancho de banda, fue necesario implementar una capa de adaptación que ajustara los paquetes IPv6 a las especificaciones 802.15.4 (García García, 2015).

Uno de los objetivos más recientes en lo referido a protocolos de comunicación es el incorporar el protocolo IP en redes de sensores de bajo consumo energético. Como consecuencia el IETF desarrollo el estándar 6LoWPAN, que permite incluir el protocolo IPv6 de internet en dispositivos basados en IEEE 802.15.4. Tal como menciona (Jiménes & Berrueco, 2016), frente a ZigBee, 6LoWPAN posibilita a los dispositivos comunicarse no solo con aparatos que dispongan del mismo protocolo sino también con dispositivos que dispongan de otros protocolos IP como Ethernet o Wi-Fi.

Comparado con Wi-Fi u otros protocolos basados en IP, 6LoWPAN tiene un consumo muy inferior. Como resultante, la integración de este estándar permite incluir el protocolo IP en dispositivos de bajo consumo y con ello, las WSN está dando un paso más hacia la configuración de una red global de Internet, es decir, Internet de las Cosas.

En la Tabla 4, se presenta un resumen de las características de cada protocolo que se ha mencionado resaltando la frecuencia y la velocidad de transferencia de los datos.

Tabla 4

Comparación de protocolos para IoT.

Protocolo	Frecuencia	Velocidad de datos	Seguridad	Uso Previo
Bluetooth (baja energía)	2,4GHz	10 kbps	Encriptado	Productos portátiles
Wi-Fi	2,4GHz/ 5GHz	54 Mbps	Opcional	Productos independientes para negocio u hogar.
Z-wave	915MHz	40 kbps	Encriptado	Producto para el hogar.
ZigBee	2,4GHz/ 915MHz	250 kbps 40 kbps	Encriptado	Productos de bajo costo para el hogar.
6LoWPAN	2,4GHz	250 kbps	Opcional	Producto dirigido a los hogares y negocios con necesidad de conexión con otros sistemas.

Fuente: (Carrasco, 2019)

Estándar IEEE 802.15.4

IEEE 802.15.4 es una tecnología de acceso inalámbrico de bajo costo y de baja tasa trasferencia de datos que funcionan principalmente por baterías. Permite una fácil instalación gracias a su pila de protocolos que la hace compacta, flexible y simple.

La incertidumbre del uso de 802.15.4 es por su confiabilidad, latencia ilimitada, susceptibilidad hacia interferencias y desvanecimiento en trayectos múltiples. Estos aspectos negativos son debido a su algoritmo de CSMA/CA, el cual es un método de acceso donde un dispositivo escucha para asegurarse que ningún otro dispositivo este transmitiendo antes que realice su propia transmisión. Si otro dispositivo está transmitiendo, se produce un tiempo de

espera que suele ser aleatorio antes que vuelva a escuchar. Esta interferencia ocurre porque con 802.15.4 no cuenta con una técnica de salto de frecuencia (que las variantes 802.15.4e y 802.15.4g ya abordan y posteriormente se detallan) (García García, 2015).

Desde 2003 se han publicado múltiples iteraciones de 802.15.4 y cada una con su etiqueta de año de publicación como: 802.15.4-2003, 802.15.4-2006, 802.15.4-2011, 802.15.4-2015.

Las capas de 802.15.4 PHY y MAC son la base de varias pilas de protocolos de red. Estas pilas de protocolos utilizan 802.15.4 en las capas física y de enlace, pero en las capas superiores son distintas, se promueven por separado a través de distintas organizaciones. Algunas de las tecnologías que se basan en 802.15.4 son: ZigBee; 6LoWPAN; ZigBee IP que adopta las capas adaptación 6LoWPAN (García García, 2015).

Capa Física 802.15.4

El estándar 802.15.4 soporta varias opciones de PHY (physical layer) que van desde los 2.4 GHz hasta sub-frecuencias de banda ISM. En el estándar IEEE 802.15.4-2003 se especifican solo tres opciones de PHY que se basan en la Modulación de Espectro Ensanchado de Secuencia Directa (DSSS) que es una técnica que propaga la señal a través del dominio de la frecuencia que da un mayor ancho de banda (Hanes et al., 2017). Las opciones de transmisión de la capa física originales eran: 2.4 GHz, 16 canales, con velocidad de datos de 250 kbps, que opera en todo el mundo; 915 MHz, 10 canales, con velocidad de datos de 40 kbps, que opera en América del norte y del sur; 868 MHz, 1 canal, con velocidad de datos de 20 kbps, que opera en Europa, Este medio y África.

Las versiones de 802.15.4 del 2006, 2011 y 2015 introdujeron opciones de comunicación PHY adicionales, incluidas las siguientes:

- **OQPSK:** es una técnica de modulación que utiliza cuatro valores de bits únicos que se señalan mediante cambios de fase. Una función de compensación que está presente durante los cambios de fase permite que los datos se transmitan de manera confiable.
- **BPSK PHY:** esto es DSSS PHY, que emplea modulación de codificación por desplazamiento de fase binaria (BPSK). BPSK especifica dos cambios de fase únicos como su esquema de codificación de datos.
- **ASK PHY:** esta es la PHY de espectro ensanchado de secuencia paralela (PSSS), que emplea modulación por desplazamiento de amplitud (ASK) y modulación BPSK. PSSS es un esquema de codificación avanzado que ofrece mayor rango, rendimiento, velocidades de datos e integridad de la señal en comparación con DSSS. ASK utiliza cambios de amplitud en lugar de cambios de fase para señalar diferentes valores de bits.

Estas mejoras realizadas aumentan la velocidad de datos de 915 MHz y 868 MHz a 250 Kpbs y 100 kbps respectivamente.

Capa MAC

Como expresa (García García, 2015), la capa MAC 802.15.4 administra el acceso al canal PHY precisando como los dispositivos en la misma área compartirán las frecuencias asignadas. La capa MAC 802.15.4 también realizan tareas como: beacon de red para dispositivos que actúan como coordinadores (nuevos dispositivos usan beacons para unirse a una red 802.15.4); asociación PAN y disociación por dispositivo; seguridad del dispositivo y comunicaciones de enlace confiables entre dos entidades MAC similares.

La capa MAC logra estas tareas utilizando varios tipos de tramas predefinidas:

- Data Frame**:** maneja todas las transferencias de datos
- Beacon Frame: se utiliza en la transmisión de balizas desde un coordinador PAN
- Acknowledgement Frame**:** confirma la recepción exitosa de una trama.

- MAC command Frame: responsable de la comunicación de control entre dispositivos

Todas estas tramas MAC de 802.15.4 están representadas en la Figura 11, en la cual se puede apreciar que la trama MAC es transportada dentro de la trama PHY.

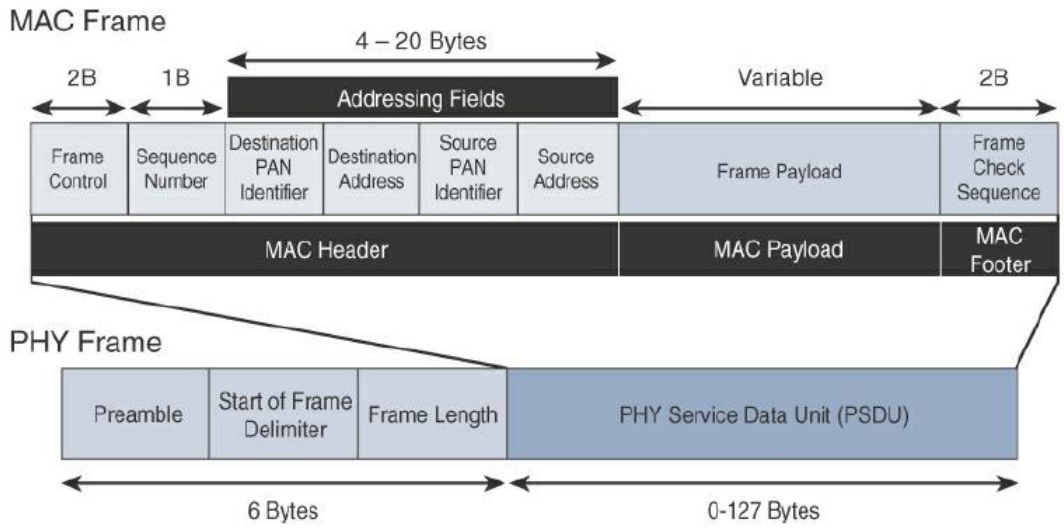


Figura 11. Formato MAC y PHY de 802.15.4.

Fuente: (Hanes et al., 2017)

El campo de encabezado MAC está compuesto por los campos Frame Control, Sequence Number y los campos Addressing. El campo frame control define atributos como el tipo de trama, los modos de direccionamiento y otros indicadores de control. El campo sequence number indica el identificador de secuencia para la trama. El campo addressing especifica los campos Identifier PAN de origen y destino, así como los campos dirección de origen y destino (Hanes et al., 2017).

Seguridad

Las especificaciones de 802.15.4 usa el Estándar de Encriptación Avanzado (AES), con una longitud de clave de 128 bits como un algoritmo de encriptación base para la seguridad de sus datos. (García García, 2015) también expresa que su uso en el sector privado ayudo a convertirse en uno de los algoritmos más populares de criptografía simétrica.

El habilitar esta función de seguridad, modifica levemente el formato del marco, consumiendo una pequeña parte de la carga útil. El usar el campo de seguridad en el control de tramas es necesario para habilitar el cifrado AES, este campo es de un solo bit que se establece en 1 para la seguridad. En la Figura 12, se puede observar que, una vez establecido el bit, se genera un campo llamado encabezado de seguridad auxiliar después del campo dirección de origen.

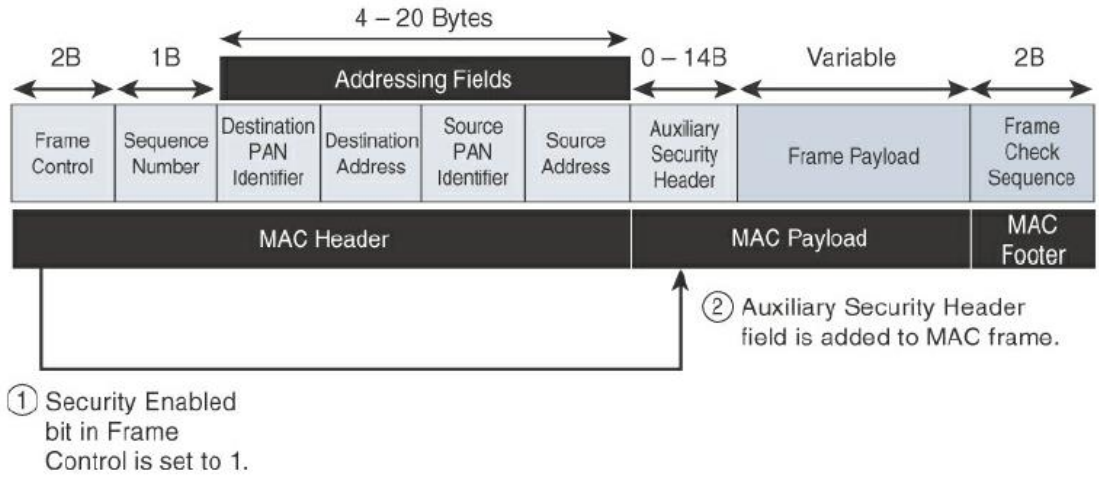


Figura 12. Formato de trama con el encabezado de seguridad auxiliar.

Fuente: (Hanes et al., 2017)

Estándar IEEE 802.15.4g y 802.15.4e

IEEE continuamente se mantiene realizando mejoras menores a las especificaciones del núcleo de 802.15.4, cuando estas mejoras son realizadas se adjuntada una letra minúscula. Dos ejemplos de esto es 802.15.4e-2012 y 802.15.4g-2012 las cuales son realmente relevantes para el tema de IoT.

La enmienda de 802.15.4e mejora varias debilidades que 802.15.4 tenía, como la confiabilidad de la MAC, la latencia ilimitada y el desvanecimiento en múltiples rutas, llegando también a ciertos dominios de aplicaciones como la automatización y la red inteligente. IEEE

802.15.4e mejoró las capacidades de la capa MAC de IEEE 802.15.4 como el formato de trama, seguridad, mecanismo de determinismo y el salto de frecuencia (García García, 2015).

La enmienda de 802.15.4g al igual que la 802.15.4e se integró en el adendum principal de 802.15.4-2015, siendo su enfoque principal la comunicación de la red de servicios públicos inteligentes. El objetivo de 802.15.4.g es optimizar las redes inalámbricas de malla para redes de campo FAN y se introducen nuevas definiciones de PHY al igual que algunas modificaciones de MAC para apoyar su implementación (García García, 2015).

Capa Física - IEEE 802.15.4g

Mientras que en IEEE 802.15.4 el tamaño de la carga útil o PSDU es de 127 bytes, en IEEE 802.15.4g aumentó a 2047, esto proporciona una mejora en el emparejamiento de paquetes mucho más grandes que se encuentran en protocolos de capa superior. Por ejemplo la configuración de MTU de IPv6 es de 1280 bytes y como explica (Hanes et al., 2017) “la fragmentación no sería necesaria en la capa 2 cuando los paquetes IPv6 se transmiten través de tramas MAC IEEE 802.15.4g. También la protección contra errores se mejoró en IEEE 802.15.4g evolucionando el CRC de 16 a 32 bits”.

IEEE 802.15.4g admite múltiples velocidades de datos en bandas que van desde los 169 MHz hasta los 2.4 GHz. Dentro de estas bandas los datos deben modularse en la frecuencia utilizando algunos de los siguientes mecanismos PHY para cumplir con 802.15.4g: Multi-Rate and Multi-Regional Frequency Shift Keying (MR-FSK), Multi-Rate and Multi-Regional Orthogonal Frequency Division Multiplexing (MR-OFDM) y Multi-Rate and Multi-Regional Offset Quadrature Phase-Shift Keying (MR-O-QPSK) (García García, 2015).

Capa MAC – IEEE 802.15.4e

Mientras que IEEE 802.15.4e no es aplicable a la capa PHY, si lo es a la capa MAC. Las enmiendas realizadas a la capa MAC la mejoran a través de varias funciones que pueden

habilitarse en función de varias implementaciones del estándar. Como plantea Hanes et al. (2017), es necesario utilizar perfiles definidos por organizaciones como Wi-SUN en casos en que la interoperabilidad es impredecible; a continuación, se describe las mejoras a la capa MAC propuestas por IEEE 802.15.4e-2012:

- **Salto de canal en intervalos de tiempo (TSCH):** su función es garantizar el acceso al medio con una diversidad de canales. El salto de frecuencia usa diferentes canales en distintos lapsos de tiempo para la transmisión. TSCH divide el tiempo en "intervalos de tiempo" que garantice ancho de banda y latencia predecible. En un intervalo de tiempo se puede transmitir un paquete y su acuse de recibo, mejorando la capacidad de la red evitando que los nodos quieran comunicarse en el mismo intervalo de tiempo, dando vigorosidad a la red para entornos ruidosos y coexistencia con otras tecnologías inalámbricas.
- **Elementos informativos (IEs):** permite el intercambio de información con la capa MAC de forma extensible, ya sea cómo IE de encabezado (estandarizado) o como IE de carga útil (privado). Especificando el formato de etiqueta, longitud y valor (TLV) se puede agregar metadatos adicionales a las tramas para soportar los servicios de la capa MAC.
- **Beacons Mejoradas (EBs):** los EBs extienden la flexibilidad de los beacons 802.15.4 dando paso a la construcción de beacons específicas de la aplicación. Esto se logra mediante la inclusión de IEs relevantes en los marcos EBs. Algunas de las características de estas beacons incluyen métricas de red, horario de transmisión con saltos de frecuencias e información de la PAN.
- **Solicitudes de Beacons Mejoradas (EBRs):** al igual que las EBs los EBRs también aprovechan las IEs en los EBRs dando posibilidad al remitente de solicitar información específica.

- **Reconocimiento Mejorado:** permite la integración de un contador de tramas para las tramas que se reconocen, que sirven como ayuda para proteger contra ciertos ataques que ocurren cuando se falsifica el marco de reconocimiento.

Protocolo de Internet (IP)

Una de las formas de garantizar la vida útil de una red por más de 30 años, es usando una arquitectura en capas como la arquitectura IP, que ha demostrado ser capaz adaptarse a pequeños y grandes cambios a lo largo del tiempo y aún mantener operaciones para sus dispositivos y usuarios (Tanenbaum & Wetherall, 2012). A continuación, se provee las ventajas de IP para IoT:

- **Abierta y basada en estándares:** internet de las cosas crea un nuevo sistema donde los dispositivos, aplicaciones y usuarios aprovechan diversas funcionalidades mientras garantizan intercambiabilidad, interoperabilidad, seguridad y administración. Esto requiere la implementación, validación y despliegue de soluciones abiertas basadas en estándares.
- **Versátil:** existe una variedad de tecnologías de acceso de última milla para la conectividad de las cosas. El hecho que las tecnologías de comunicación avancen a un ritmo acelerado, logra que su vida útil estimada sea de 10 a 20 años. Pero con la tecnología IP en capas puede hacer frente a estos cambios, como tal, estas capas usan varios protocolos que se pueden usar a lo largo de su duración sin requerir cambios en toda su arquitectura.
- **Ubicua:** todas las versiones de sistemas operativos de uso general vienen con una pila de IP dual (IPv4 e IPv6) que se puede mejorar posteriormente. Si bien las actualizaciones de IP son principalmente en IPv4, recientes esfuerzos de estandarización han incluido IPv6. El protocolo IP es el más divulgado al observar que admite múltiples soluciones en IoT.

- **Escalable:** IP ha probado ampliamente su escalabilidad de forma sólida. Millares de nodos con infraestructuras tanto públicas como privadas han estado operativas durante varios años incluso dando un buen soporte a aquellas redes que no están asociadas a IP. Mostrando que la escalabilidad uno de sus puntos fuertes.
- **Administrable y altamente seguro:** una infraestructura de red requiere administración y seguridad para funcionar. IP al estar más de 30 años en actividad cuenta con gestión de redes, protocolos y mecanismos que se encuentran disponibles. No obstante, aún enfrenta varios desafíos en áreas como seguridad en nodos restringidos, protocolos OT heredados y escalamiento de operaciones.
- **Estable y resistente:** IP ha existido desde hace más de 30 años y tiene una base de conocimientos extensa y establecida que se ha sido usada en infraestructuras criticas como redes financieras y de defensa. Además, su resistencia y estabilidad es beneficiada por su entorno que cuenta con profesionales de IT que pueden diseñar, desarrollar y operara soluciones IP.
- **El factor de innovación:** en las últimas décadas se ha establecido a IP como el factor de innovación debido a que es el protocolo detrás de la transferencia de archivos, correo electrónico, comercio electrónico redes sociales, etc. e IoT puede aprovechar estas innovaciones.

Pero, con el creciente uso de internet, se presentan inconvenientes. La expansión genero un obstáculo ineludible: el agotamiento del direccionamiento por parte del estándar utilizado. A raíz de ello, se desarrolló una nueva versión del protocolo IP que contempla el aumento de dispositivos sin el riesgo de agotar el direccionamiento. Así es como nació el nuevo estándar IPv6 que busca eliminar la limitante de direcciones asignables que se tenía con su antecesor y mejorar algunos aspectos tales como la optimización de recursos, máscaras y sub-máscaras para la identificación de usuarios de origen y de fin (Jiménes & Berrueco, 2016).

La expansión de capacidades que tiene IPv6 es aumentar de 32 bits a 128 bits, que supone una mayor cobertura para cualquier dispositivo: $2^{128}=3x10^{38}$ posibles direcciones utilizables. IPv6 tiene una mayor capacidad de direcciones IP disponibles. Se compone por ocho campos y cada uno de 16 bits unidos por dos puntos, a cada digito de estos grupos son números hexadecimales. Su formato se presenta en la Figura 13, que muestra cómo es su direccionamiento.

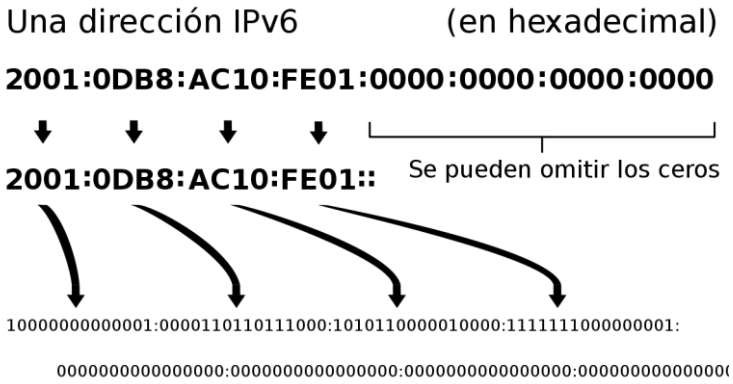


Figura 13. Formato IPv6.

Obtenido de: (Smetana, 2012)

Como se ha descrito, IPv6 es el nuevo estándar de internet que progresivamente reemplazara a IPv4. Permitiendo una conexión total de todos los dispositivos con acceso a internet siendo crucial para que IoT se desarrolle por completo logrando que cualquier "cosa" se conecte a la red (Jiménes & Berrueco, 2016).

Adaptación de IP para IoT

Mientras el protocolo IP es la clave para el éxito de IoT; nodos y redes restringidas demandan de varias optimizaciones de capas y sobre múltiples protocolos de la arquitectura IP. La Figura 14 presenta las capas donde la optimización fue aplicada por la IETF.

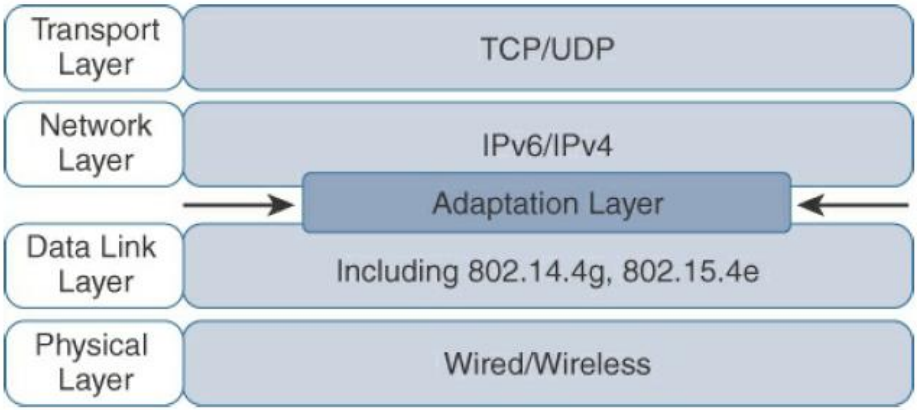


Figura 14. Optimización del Protocolo IP para IoT.

Fuente: (Hanes et al., 2017)

6LoWPAN

6LoWPAN aprovecha las mejoras realizadas en los estándares IEEE 802.15.4g y IEEE 802.15.4e que sientan las bases para una transmisión sobre nodos y redes restringidas adoptando IPv6 y aprovechando la comunicación inalámbrica en malla al máximo. En la arquitectura IP para transportar un paquete se lo realiza a través de las capas 1 (PHY) y 2 (MAC) que primero se deben definir y documentar (Castillo Merchán, 2016).

Como lo hace notar Hanes et al. (2017), el modelo para empaquetar IP en las capas inferiores es conocido como una capa de adaptación y, a menos que la tecnología sea propietaria, la capa de adaptación IP es comúnmente definida por el grupo IETF en los RFC el cual define como se comporta un estándar de internet. Los protocolos relacionados a IoT siguen un proceso similar. La principal diferencia es que una capa de adaptación está diseñada para que IoT pueda incluir varias optimizaciones que le ayude a lidiar con redes y nodos restringidos.

Uno de los principales trabajos de adaptación fue realizado por el grupo de trabajo 6LoWPAN que hizo varias publicaciones con un enfoque en la transmisión de paquetes IPv6 a través de redes restringidas. Pero la más importante fue el RFC 4994 que define las

capacidades de las cabeceras de compresión, fragmentación y direccionamiento de malla (García García, 2015). Dependiendo de la implementación, se pueden habilitar todas, ninguna o cualquier combinación de estas capacidades y sus encabezados correspondientes. La Figura 15 presenta las cabeceras dentro de 6LoWPAN.

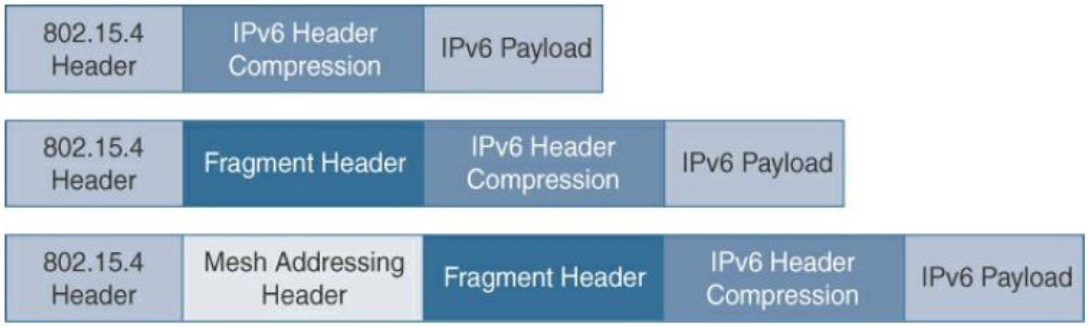


Figura 15. Cabeceras de 6LoWPAN**.**

Fuente: (Hanes et al., 2017)

Header Compression: la cabecera de compresión de IPv6 para 6LoWPAN fue definida en el RFC 4944 y actualizada en el RFC 6282. La compresión de cabecera ayuda a reducir el tamaño de las cabeceras de 40 bytes de IPv6 y 8 bytes de UDP, hasta 6 bytes combinándolas en algunos casos. La compresión de encabezado no tiene estado y no son complicadas, pero cuenta con algunos factores que afectan la cantidad de compresión. Para entender su función e impacto, se proporciona un ejemplo de la compresión de 6LoWPAN: en alto nivel 6LoWPAN aprovecha la información conocida compartida por el resto de los nodos en una red local y omite varios campos que asume que son comunes (Hanes et al., 2017). En la Figura 16, se observa la cantidad de compresión de 6LoWPAN.

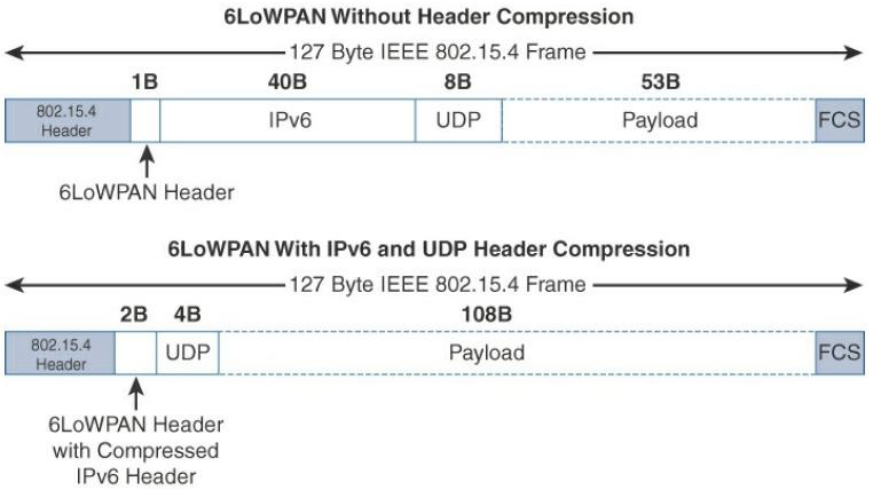


Figura 16. Cabecera de Compresión.

Fuente: (Hanes et al., 2017)

En la parte superior de la Figura 16, la trama 6LoWPAN no cuenta con ninguna compresión de encabezado, por ende, el encabezado IPv6 completo es de 40 bytes, el encabezado UDP es 8 bytes y el encabezado 6LoWPAN es 1 solo byte. Teniendo en cuenta lo anterior, la carga útil es de solo 53 bytes de los 127 que comprende la trama. Por otro lado, en la parte inferior de la Figura 16, se aprecia una compresión ideal de encabezados. El encabezado 6LoWPAN ha aumentado a 2 bytes para acomodar la compresión del encabezado IPv6, mientras el encabezado UDP se ha reducido a 4 bytes. Lo importante de la compresión de encabezados es que ha permitido aumentar el tamaño de la carga útil a más del doble, es decir, 108 bytes de los 127 que comprende la trama (Hanes et al., 2017).

Fragment Header: la unidad de transmisión máxima (MTU) para una red IPv6 debe ser de al menos 1280 bytes. Actualmente el MTU definido por IEEE 802.15.4 es de 127 bytes. No obstante, como el MTU de IPv6 es más grande y se debe transportar dentro de una trama 802.15.4 más pequeña los paquetes IPv6 deben fragmentarse en varias tramas 802.15.4 en la capa 2.

La cabecera de fragmentación se compone de tres campos: datagram size, datagram tag y datagrama offset. El campo datagram size es de 1 byte y especifica el tamaño de la carga útil sin fragmentar, el datagram tag identifica el conjunto de fragmentos de la carga útil y el datagram offset define que tan lejos en la carga útil ocurre un fragmento en particular. Se puede apreciar mejor el encabezado de fragmentación en la Figura 17.

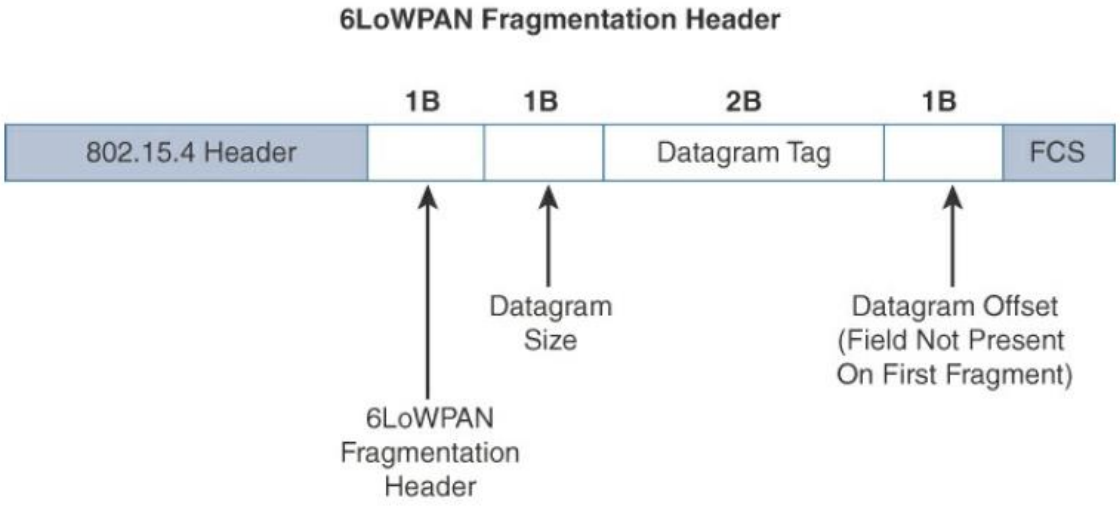


Figura 17. Cabecera de Fragmentación.

Fuente: (Hanes et al., 2017)

En la Figura 17, el encabezado de fragmentación de 6LoWPAN usa en sí mismo un valor de bit único para identificar que los campos de subsiguientes son campos de fragmento opuesto a sus capacidades. Por otra parte, en el primer fragmento su campo Offset no se encontraría debido a que su valor se fijaría en 0, resultando que la primera cabecera de fragmentación para una carga útil IPv6 solo cuente con 4 bytes de longitud. El resto de los fragmentos tendrán un campo de encabezado de 5 bytes para especificar el desplazamiento apropiado.

Mesh Addressing Header: la función principal de la cabecera de direccionamiento de malla 6LoWPAN es reenviar los paquetes por medio de múltiples saltos definiendo tres campos para esta cabecera: hop count, source address y destination address. El campo límite de saltos es similar al datagrama en IPv6 y proporciona un límite superior sobre cuantas veces se puede

reenviar una trama; cada vez que el paquete es reenviado su valor se reduce en 1 y al llegar a 0 se elimina dejando de reenviarse (Hanes et al., 2017). Los campos de dirección origen y destino indican el punto inicial y final respectivamente de un salto IP y son direcciones 802.15.4, estos campos pueden ser apreciados en la Figura 18.

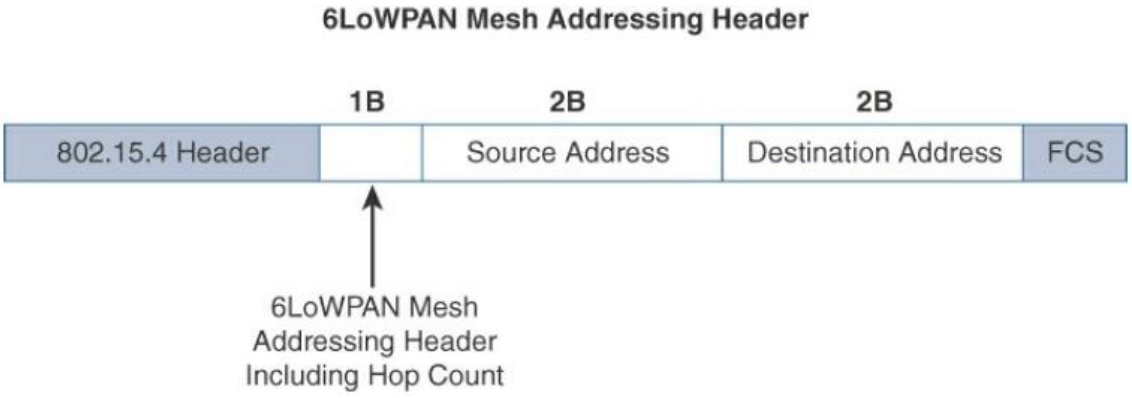


Figura 18. Campos del encabezado de direccionamiento de malla.

Fuente: (Hanes et al., 2017)

Configuración 6LoWPAN

El estándar 6LoWPAN se compone normalmente de varios nodos conectados simultáneamente en una topología de malla que es capaz de comunicarse con redes externas y por consecuencia conectarse a internet, como se muestra en la Figura 19, una red con 6LoWPAN conectada a distintos terminales en una red local (Jiménes & Berrueco, 2016).

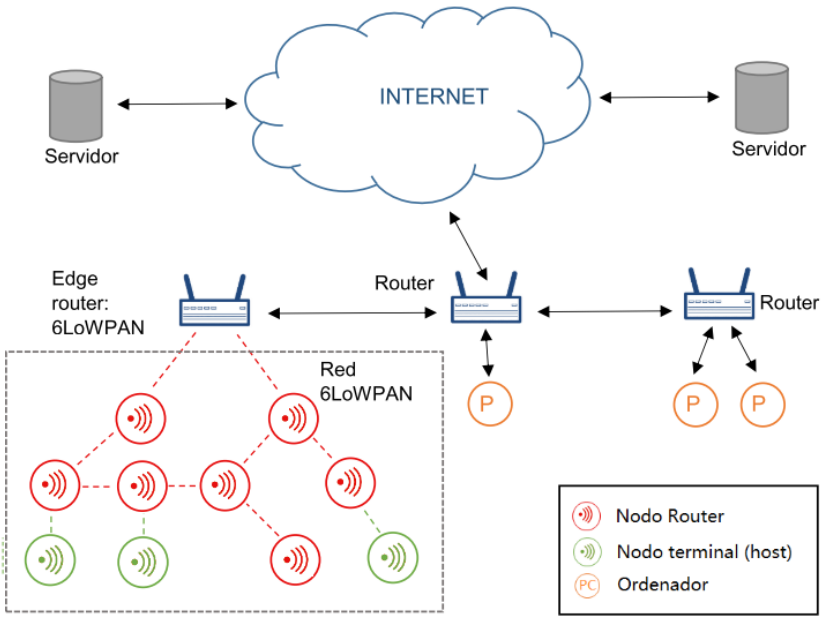


Figura 19. Estructura de una red 6LoWPAN conectada a una red.

Fuente: (Jiménes & Berrueco, 2016)

Las aplicaciones que se ejecutan en los nodos envían sus paquetes IP por los router de frontera (router edge) el cual también es el encargado de la comunicación entre los dispositivos 6LoWPAN y redes IPv6 o realiza una comunicación directa con internet. Así mismo es el encargado de direccionar los paquetes a nivel de capa de red y deben tener la capacidad de convertir paquetes IPv4 en IPv6 y viceversa debido a que se basan en el estándar IEEE 802.15.4 y no se pueden comunicar directamente con redes IPv4 (Jiménes & Berrueco, 2016).

Estas redes LoWPAN no tienen conocimiento de otro tipo de redes, no llevan tráfico, tienen comunicación por medio de router de frontera. Este tipo de redes se las conoce como stub (talón) debido a que solo pueden enviar o recibir paquetes. Están formadas por dos tipos de nodos: los nodos de rutado los cuales permiten la comunicación entre dispositivos LoWPAN y al router de frontera 6LoWPAN; y los nodos terminales no son capaces de enviar información dentro de la red y solo receptan información cada cierto tiempo, consultando al nodo conectado

cada cierto intervalo de tiempo si tiene pendiente información por enviar, logrando así un ahorro de energía ya que no está pendiente todo el tiempo (Carrasco, 2019).

Las redes 6LoWPAN cuentan con tres configuraciones que se pueden trabajar, permitiendo una conectividad general e interoperabilidad con dispositivos heterogéneos para redes inalámbricas de baja potencia que comúnmente se utilizan en IoT:

- Simple LoWPAN: trabaja con un único enrutador (router de frontera) para conectarse a otra red.
- Extended LoWPAN: trabajan con varios enrutadores de frontera conectados entre sí en una red troncal comúnmente ethernet. Cuenta con un servidor para administrarlos siendo utilizada para redes de gran tamaño.
- Ad-Hoc LoWPAN: esta red opera sin conexión a internet dentro de una red exclusiva que no cuenta con ningún router (Carrasco, 2019).

Para los enlaces de comunicación estas redes por defecto utilizan el protocolo de enrutamiento para redes inalámbricas de bajo consumo y susceptible a perdidas RPL, el cual es un protocolo basado en vectores distancia que opera sobre IEEE 802.15.4.

RPL se basa en el concepto de gráfico acíclico dirigido (DAG), el cual es un grafo donde no existe ciclos. Un proceso RPL involucra crear un gráfico acíclico dirigido orientado al destino DODAG que básicamente es un DAG arraigado a un destino. RPL cuenta con un rango o “rank” asignado a cada nodo en la red, el cual aumenta a medida que el dispositivo se alega del dispositivo raíz (Cama & Cama, 2012). Dentro de este concepto se definen tres tipos de paquetes:

DIS (Solicitud de información del DODAG): son mensajes de solicitud de información del DODAG más cercanos que se usa para describir redes existentes.

DIO (Objeto de información del DAG): son mensajes que manejan los cambios o descubrimientos de ruta en la red y que envían información de los DAG en respuesta a los mensajes DIS. Los mensajes DIO determinan a los “padres” y el mejor camino hacia la raíz DODAG.

DAO (Objeto de actualización del destino): son mensajes que se envían hacia el DODAG y que permiten informar a los “padres” de su presencia y de su accesibilidad a los descendientes.

Amenazas de seguridad

Los ataques que puede sufrir la seguridad de los datos en 6LoWPAN pueden llegar a ser muy destructivos y perjudiciales. Las amenazas al protocolo se encuentran en todas las capas del modelo OSI.

6LoWPAN es vulnerable contra ataques físicos o amenazas debido a la reubicación, destrucción o enmascaramiento (Cañete, 2015). Usando este tipo de ataque, los nodos pueden ser suprimidos y perder información de forma irreversible. Un ataque físico puede modificar la circuitería y cambiar en su programación el roll al que estaba destinado de forma que actúe como un nodo maligno que puede tomar el control.

En 6LoWPAN se pueden llevar a cabo varios ataques de tipo DoS en las diferentes capas, por ejemplo, en la capa física la denegación de servicio puede darse por interferencias electromagnéticas.

Capa MAC: los Ataques que se pueden dar hacia la Medium Access Control (MAC), pueden ser variados debido a que ser dispositivos con recursos limitados, estos buscan ahorrar lo mayor posible su batería entrando en modo reposo. Con estas características su trataría de atacar directamente a su energía agotándola, enviando paquetes inútiles con la finalidad que el nodo no entre en estado de ahorro. Estos ataques pueden ir dirigidos a cualquier nodo, pero el

coordinador del PAN puede ser el objetivo final; manteniendo ocupado este nodo, se amenaza la disponibilidad completa de la red (Cañete, 2015).

Capa de Red: en esta capa, se presentan dos clases de ataques:

- **Falsificado o alterado de información:** este tipo de ataque usa un nodo malicioso para modificar o crear información falsa que se reenvía entre los nodos para crear bucles en el enrutamiento, atraer o repeler tráfico de red, aumentando las rutas y generando errores generales en la red.
- **Ataque a la fragmentación:** como se había mencionado con anterioridad, para que IPv6 funcione sobre el estándar 802.15.4 se usa el mecanismo de fragmentación que se encuentra en la capa adaptación. Al ser dispositivos con "características limitadas" no cuenta con un método de autenticación en la capa 6LoWPAN, entonces nodos malignos pueden enviar fragmentos duplicados que los nodos a donde estén destinado estos fragmentos no pueden identificar.

Los nodos encargados del empaquetamiento almacenan todas las partes de un paquete con el fin de armar un paquete completo, por ello cuentan con un tiempo de espera para descartar paquetes que no estén completos; pero, al solo enviar paquetes incompletos durante periodos prolongados, su escasa memoria se llena por completo (Cañete, 2015).

2.1.2.5 Sistemas Operativos IoT

Como señala (Carrasco, 2019), existen dispositivos IoT de gama alta como Raspherry PI que pueden usar sistemas operativos como Linux y de igual manera dispositivos de gama baja con muchos menos recursos que requieren de otro sistema operativo. La mayoría de los dispositivos son de gama baja. A continuación, se presenta los sistemas operativos IoT de código abierto:

Contiki: es un sistema operativo de código abierto que se ejecuta en pequeños microcontroladores de bajo consumo y de bajo costo, que son integrados sobre microcontroladores y nodos de redes de sensores de 8 y 16 bits. Su configuración básica consta de 2KB de RAM y 40 KB de ROM. Tiene una pila ligera de TCP/IP incluida que está diseñada para comunicaciones inalámbricas de baja potencia y soporta protocolos como IPv6 al igual que el protocolo de enrutamiento para redes de bajos recursos y perdidas RPL y 6LoWPAN. Contiki es escrito sobre un lenguaje de programación C y usa una licencia tipo BSD (Contiki, 2014).

RIOT: es compatible con la mayoría de los dispositivos IoT de bajo consumo y con arquitecturas de microcontroladores de 32, 16 y 8 bits. Además, es un sistema de código abierto que se basa en arquitectura de microkernel, cuenta con capacidades multiproceso en tiempo real y es compatible con IPv6, 6LoWPAN, RPL y CoaP (Riot, 2016).

Tiny OS: es una plataforma y sistema operativo integrado que se basa en componentes inalámbricos de bajo consumo como los usados en WSN. Soporta plataformas HW de 8 y 16 bits. Implementa una pila completa de 6LoWPAN, RPL y IPv6 (TinyOS Alliance, 2012).

Tiny Core: no es considerado un sistema operativo para IoT por sí mismo. Mas bien es un sistema operativo que se basa en el kernel Linux 2.6 y tener un consumo de solo 10 MB. Se puede instalar en un CD o USB que funcionan como emulador para dispositivos IoT en ambientes virtuales (Shingledecker, n.d.).

Zephyr: es un sistema operativo en tiempo real para dispositivos conectados que cuentan con recursos limitados e integrados (microcontroladores) soportando múltiples arquitecturas. Posee los componentes necesarios para desarrollar aplicaciones integradas al igual que un conjunto de protocolos como IPv4, IPv6, 802.15.4, Bluetooth Low Energy.

SDN-WISE: es una solución de Redes Definidas por Software para Redes de Sensores Inalámbricos sistema operativo basado en SDN. SDN-WISE se basa en las capas MAC y PHY de 802.15.4. Los elementos de red se pueden distinguir por nodos y sinks (sumideros). Cuenta con tablas de flujo flexibles que permite la creación de reglas para la coincidencia de paquetes que son particularmente útiles en protocolos de red como 6LoWPAN y ZigBee (Galluccio, Milardo, Morabito, & Palazzo, 2015).

En la Tabla 5, se presenta un resumen de los sistemas operativos para IoT descritos. Los recursos que necesitan y los protocolos que manejan para las comunicaciones.

Tabla 5

Comparación de SO para IoT.

Sistema Operativo	Lenguaje de Programación	RAM Requerida (KB)	Protocolos Soportados	Simulador disponible
Contiki	C	2	RPL, IPv6, 6LoWPAN	Si
RIOT	C/C++	1.5	IPv6, 6LoWPAN, RPL y CoaP	No
Tiny OS	NesC	1	6LoWPAN, RPL y IPv6	No
Zephyr	C	8	IPv4, IPv6, 802.15.4, Bluetooth Low Energy	
SDN-WISE	Java	10	6LoWPAN, ZigBee	Si

Fuente: (Khan et al., 2016)

2.2 Redes Definidas por Software (SDN)

Las redes definidas por software SDN como se las conoce comúnmente, hace referencia a la separación física del plano de control y del plano de datos en los elementos de red. Por lo tanto, los conmutadores y enrutadores están bajo el dominio del plano de control el cual puede gestionar varios dispositivos. Este cambio de paradigma que ofrece SDN según (Lobato, 2013), promete brindar una mayor velocidad, una infraestructura ágil, mejores costos de TI y múltiples dinamismos de las aplicaciones existentes.

Estas redes están basadas en estándares abiertos y no cuentan con un proveedor propietario como la mayoría de los elementos de red, logrando simplificar las operaciones de diseño, siendo que las decisiones son tomadas por el controlador, mediante el protocolo de comunicación OpenFlow.

El contar con una separación de los planos reduce el tiempo de enrutamiento logrando mejorar los tiempos de entrega de los datos entre dispositivos y contando con una mejor administración de los dispositivos respecto a las redes tradicionales.

2.2.1 Arquitectura SDN

Las redes tradicionales y las redes SDN funcionan mediante dos planos que realizan la función de intercambio de información y envío de datos, estos planos se los conoce como plano de control y plano de datos.

Plano de control: contiene los mecanismos avanzados de capa tres para lograr el intercambio de información de enrutamiento como: OSPF, EIGRP, RIP, BGP, etc.

Plano de datos: es quien realiza los envíos de la información basado en la dirección destino, es un mecanismo sencillo de envío.

El funcionamiento de los dispositivos de una red TI normal frente a un dispositivo SDN se puede observar en la Figura 20. Donde en el dispositivo tradicional tanto el plano de control

como el de datos funcionan conjuntamente y dependiendo del protocolo de enrutamiento implementado se enrutan los datos. Una de las limitaciones que posee esta arquitectura es la administración que deben tener de cada uno de los dispositivos, puesto que se debe configurar y compilar todos los elementos de una infraestructura de red.

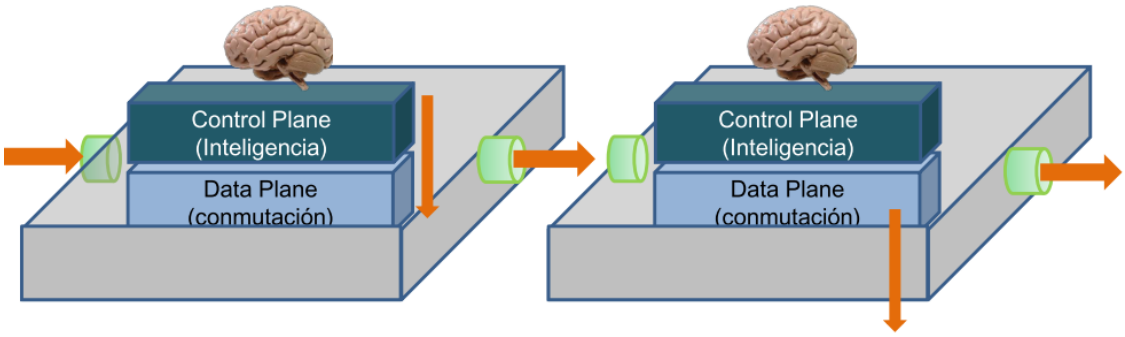


Figura 20. Funcionamiento de los dispositivos de red.

Fuente: (Lobato, 2013)

Una red SDN (a diferencia de las redes tradicionales) tiene un funcionamiento diferente, no obstante, aún conserva su esencia básica de los planos de trabajo, aunque en diferentes dispositivos. El plano de datos aún funciona sobre el dispositivo de enrutamiento, pero el plano de control funciona sobre un dispositivo centralizado apartado del dispositivo de enrutamiento conocido como controlador SDN. Este concepto se ilustra en la Figura 21.

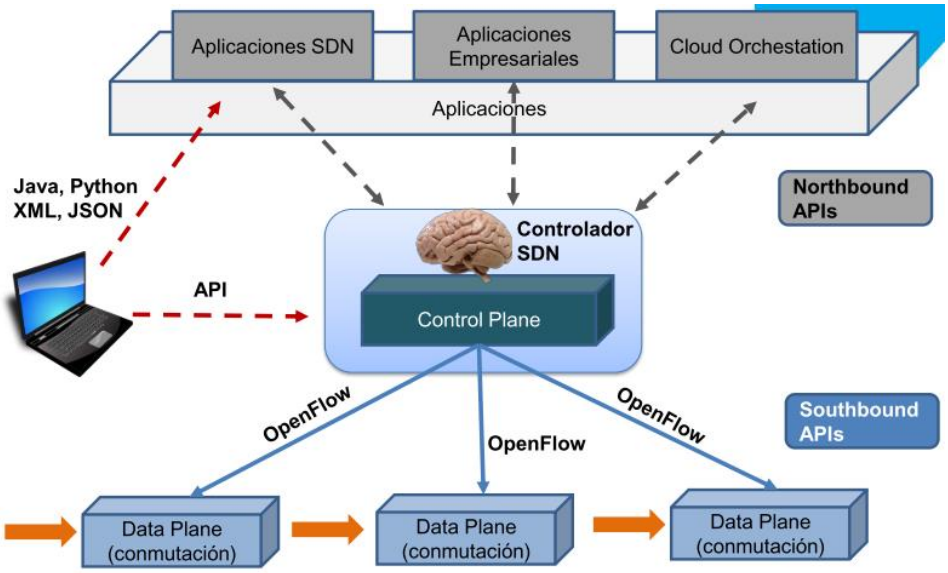


Figura 21. Arquitectura SDN

Obtenido de: (Lobato, 2013)

En síntesis, la arquitectura SDN separa los principales planos, donde el plano de control se encuentra sobre un dispositivo central que se comunica (a través del protocolo de comunicación OpenFlow) con los elementos de red y con las aplicaciones de los usuarios.

En esta arquitectura las aplicaciones funcionan mediante APIS que son otorgadas por el controlador y debido a que las aplicaciones no son generalizadas, al pasar a otro controlador, se los debe adaptar. Según indica (Triana, 2017), el controlador otorga la funcionalidad de gestión como rendimiento, fallas a través de SNMP y otros protocolos estándar. Por lo general se usa la gestión de configuración de dispositivos OpenFlow con el fin de proporcionar una topología de la red, el reenvió, QoS, y de gestión de enlace.

Igualmente, este modelo de gestión tiene una visión más completa de la red debido a que intercambia información de forma regular entre las distintas capas que componen su arquitectura: aplicación, control e infraestructura.

2.2.2 Capas SDN

Al modelo planteado por SDN se lo pudo conceptualizar dividiéndolo en tres capas lógicas que trabajan en conjunto para que el modelo funcione. A continuación, en la Figura 22, se expresa de forma gráfica cómo interactúan estas capas.

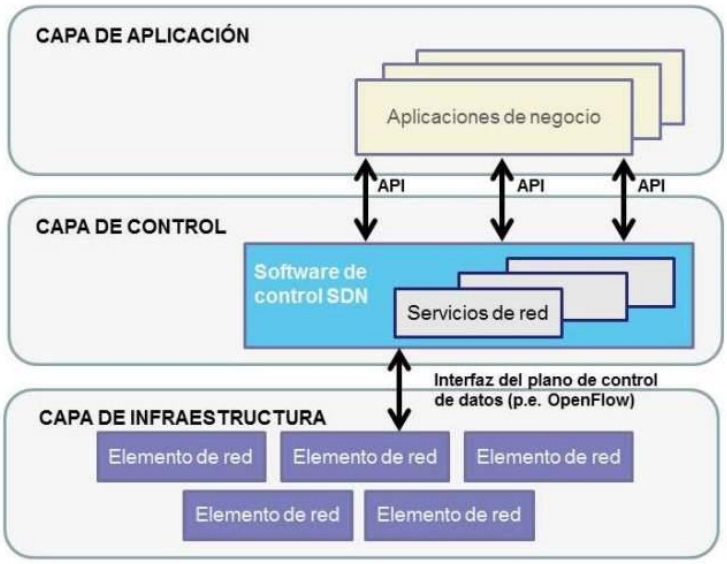


Figura 22. Capas de la arquitectura SDN.

Obtenido de: (Millán Tejedor, 2014)

Capa de Infraestructura: esta capa lo componen los elementos de hardware de red que permiten la conectividad física del entorno: host, conmutadores, enrutadores y medios de transmisión. Es el equivalente a la capa física del modelo OSI. Toman algunas decisiones en la transferencia de datos para procesar el tráfico o para reenviarlo y ejecutan dos tareas de acuerdo con sus componentes lógicos.

Control: obtiene información del estado de la red como su topología o el estado de la red quien así mismo señala las reglas para el reenvío de paquetes.

Datos: el procesador de red envía los paquetes coincidentes que el dispositivo identifica y que son tomadas por el plano de control. Al igual que en las redes tradicionales, el proceso se

realiza en base a las direcciones IP o MAC, pero SDN también se puede basar en puertos TCP/UDP, etiquetas, VLAN´s, o puerto de entrada del conmutador.

Capa de Control: es la capa intermedia de la arquitectura donde se encuentra el controlador que da una visión global de la red y que conforma el sistema operativo de red (NOS). Mediante la comunicación con la capa inferior (infraestructura), recepta el estado de la red y de acuerdo con las aplicaciones ejecutadas toma decisiones y actualiza las tablas de flujo en caso de darse cambios. A su vez, se comunica con la capa superior (aplicación) mediante traducción, teniendo varias opciones de lenguajes de programación de alto nivel (Phyton, C++, Java, entre otros).

En esta capa el controlador es el principal componente que debe ser capaz de comunicarse con otros controladores dentro de la red. Al contar con un solo controlador, este se convierte en un punto crítico que, en caso de fallar, toda la red puede colapsar al no contar con un respaldo. Debido a que no se tiene un controlador SDN estandarizado, dentro de una red amplia puede incluir varios y de distintas opciones que pueden realizar transferencia de información entre sí.

Dependiendo de la comunicación que se realiza entre capas desde de la capa de control, como muestra la Figura 23, existen dos tipos de interfaces de programación:

- Southbound APIs: la función que tienen estas API’s es básicamente comunicar al controlador con los elementos de conmutación que tiene la red y realizar cambios en función de las necesidades y demandas que se presenten.
- Northbound APIs: **l**a función que tienen estas API’s son las más críticas en la red, ya que tiene que soportar una variedad de servicios y aplicaciones. Se conectan directamente con el controlador y este se comunica con los dispositivos.

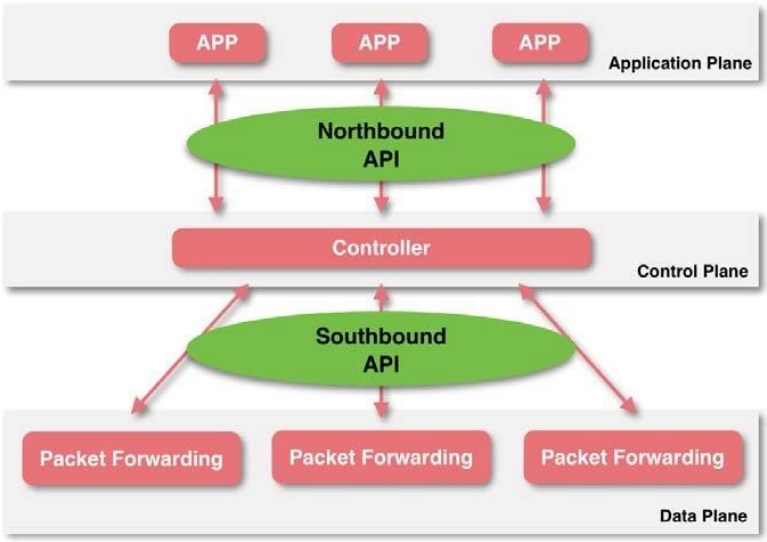


Figura 23. Función de las APIs dentro de la comunicación SDN.

Obtenido de: (Wu, Huang, Kong, Tang, & Huang, 2015)

Capa Aplicación: en esta capa es donde cambia el funcionamiento con respecto a las redes tradicionales. Las aplicaciones que contenga la red comunican sus requisitos al controlador mediante el Northbound para complacer las necesidades de los usuarios.

Entre las aplicaciones que se pueden ejecutar se cuenta con: enrutamiento adaptativo, itinerancia sin interrupciones, mantenimiento, seguridad de red, virtualización y cloud computing (Serrano, 2015).

2.2.3 Protocolo OpenFlow

OpenFlow es el protocolo de comunicación de las redes SDN que fue desarrollado con estándares abiertos para la comunicación entre el controlador SDN y los dispositivos de conmutación. En síntesis, es la comunicación entre la capa de control y la capa de infraestructura de red. Este protocolo está basado en el envío de mensajes para establecer la conexión con los dispositivos, como ilustra la Figura 24, que posteriormente realizan las acciones correspondientes (Blandón Gómez, 2013).

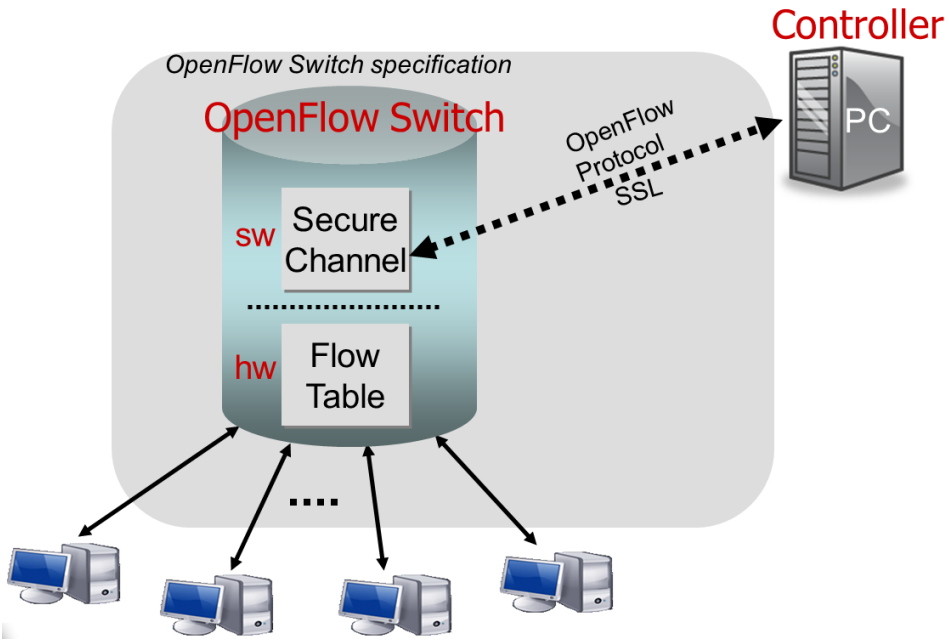


Figura 24. Funcionamiento de un OpenFlow Switch.

Obtenido de: (McKeown et al., 2008)

La comunicación se establece con protocolo SSL a nivel de software, mientras que a nivel de hardware lo hace la tabla de flujos. El protocolo OpenFlow hace referencia a un estándar con el cual se especifica las entradas de una tabla de flujos que posteriormente se actualiza en los conmutadores los cuales tienen que seguir las instrucciones asignadas. El hecho que se pueda asignar a un controlador la tarea de control hace posible que se pueda programar de manera que se puede administrar el cómo van a ser procesados los diferentes flujos.

Switch OpenFlow: un switch OpenFlow es un conmutador basado en el protocolo OpenFlow que emplea técnicas SDN para reenviar paquetes en una red y consiste en tres partes:

- Tabla de Flujos: acción asociada a cada entrada que determina como debe procesar el flujo de datos
- SSL: capa de conexión segura, protocolo usado para el controlador y los elementos de conmutación.
- OFP: protocolo de comunicación abierto entre el controlador y los dispositivos.

El switch OpenFlow se dividen en dos categorías: switch OpenFlow-Only dedicado que no admiten el procesamiento en el reenvío de tráfico de Capa 2 y Capa 3; y OpenFlow-enable switch que se usan tanto para operaciones OpenFlow, así como para la conmutación tradicional. Tienen incluido la arquitectura OpenFlow que añaden flujos de tabla, canal seguro y el protocolo OpenFlow.

El protocolo OpenFlow funciona de igual forma para los ambientes con redes de sensores, debido a que mantiene su principal característica de una red SDN que es la separación de los planos de control y datos, pero en dispositivos de tamaño reducido y debido a ello se los adapta a las nuevas necesidades.

2.2.4 Controlador

Como se mencionó anteriormente, el controlador es el elemento más importante dentro de una red SDN, y de acuerdo con (Eissa, Bozed, & Younis, 2019) "es quien toma las decisiones, implementa las reglas de la red, ejecuta las instrucciones proporcionadas por las diferentes aplicaciones y las distribuye a los diferentes dispositivos de capa física de la red". Un controlador ofrece una interfaz de programación para los conmutadores de forma que las aplicaciones de gestión pueden realizar su función y ofrecer nuevas funcionalidades. Un controlador SDN es un software que ofrece características tales como:

- Gestión del estado de la red en una base de datos que sirve como repositorio de elementos gestionados como estado de la red, información de configuración e información de la topología de red.
- Un mecanismo de descubrimiento de dispositivo, topología, sistemas de cálculo de ruta.
- Un protocolo basado en estándares OpenFlow para obtener el estado de la red impulsados por aplicaciones de elementos en la red.

- Un conjunto de API´s que generalmente son RESTFul, en los cuales se exponen los servicios del controlador a las aplicaciones de gestión.

2.2.4.1 Controladores SDN

Ryu: es un componente de las redes definidas por software. Adicionalmente, proporciona componentes software con API´s bien definidas que hace más sencillo para desarrolladores crear nuevas aplicaciones de administración y control de red. Soporta varios protocolos para la administración de dispositivos como OpenFlow, Netconf, OF-config, etc. Incluye las versiones 1.0, 1.2, 1.3, 1.4, 1.5 de OpenFlow (RYU SDN Project Team, 2016). Una de las ventajas de este controlador, es que existe la documentación necesaria para crear aplicaciones para el controlador y los elementos.

Floodlight: FloodLight es un controlador de red desarrollado por una comunidad abierta que se utiliza con el protocolo de comunicación OpenFlow para orquestar los flujos de tráfico en un entorno SDN. Esta desarrollado en lenguaje de programación de Java y ofrece la capacidad de adaptar fácilmente software y desarrollar aplicaciones. Es compatible con el protocolo OpenFlow v1.0 (Project Floodlight, n.d.).

OpenDayLigth (ODL): es un proyecto colaborativo de código abierto escrito en lenguaje de programación Java, cuyo objetivo es promover las redes definidas por software (SDN) y la virtualización de funciones de red (NFV). Debido a la naturaleza de código abierto del controlador permite al usuario minimizar la complejidad de las operaciones y en consecuencia alarga la vida de la infraestructura existente. El controlador ODL actúa como un software completo que puede ser iniciado en cualquier sistema operativo, así como también en JVM (Java Virtual Machine). Varios protocolos pueden ser operados en ODL en su SouthBound como por ejemplo OpenFlow 1.0, OpenFlow 1.3, BGP-LS, etc. (LF Projects, 2019).

ONOS: el controlador ONOS fue diseñado para satisfacer las necesidades de los desarrolladores que desean construir soluciones donde se aproveche el hardware al mismo tiempo que ofrece la flexibilidad de crear e implementar nuevos servicios de red dinámicos con interfaces programables simplificadas. ONOS admite la configuración y el control en tiempo real. También se encarga de la escalabilidad y alta disponibilidad. Fue diseñado y construido para tener un alto rendimiento para operaciones de red escaladas; ONOS escala según sea necesario agregando nuevas instancias cuando se necesita más capacidad del plano de control (Open Network Foundation, 2020).

ONOS soporta múltiples protocolos en su interfaz Southbound que provee la comunicación con los diferentes dispositivos y explotar la API correcta en Northbound a fin de satisfacer las necesidades de los proveedores de servicio y desarrolladores.

IT-SDN: Es un controlador de red que viene integrado con la herramienta de simulación IT-SDN que puede ser ejecutado en una PC. Un nodo contiki es usado como intermediario entre la red de sensores y el software del controlador.

Basado en los paquetes de informes vecinos recibidos de los nodos de la red, el software para PC mantiene una vista global centralizada de la red almacenada en forma gráfica. Como explica Alves et al. (2017), esta representación se utiliza para calcular la solicitud de flujo del nodo y actualizar la tabla de flujo de acuerdo con el algoritmo de ruta más corta de Dijkstra. La caché de la tabla de flujo mantiene una copia de las rutas configuradas previamente en el nodo. El mensaje de informe de vecino puede cambiar la vista de red del controlador y activar un nuevo cálculo de ruta.

µSDN: µSDN se ha desarrollado para proporcionar una plataforma de código abierto para entregar SDN en redes 6LoWPAN IEEE 802.15.4-2012. Además de µSDN, proporcionamos un controlador SDN integrado, Atom, así como un generador de flujo para fines de prueba,

Multiflow. Para obtener más detalles sobre µSDN, se remite al lector a la referencia (Baddeley et al., 2018).

SD6WSN: presenta una solución para el control del comportamiento del tráfico de datos 6LoWPAN con un enfoque SDN, tomando en cuenta todas las limitaciones específicas de los nodos WSN como la alta latencia, baja transferencia de datos, perdida de paquetes, etc., aprovechando la flexibilidad que brinda un reenvío basado en flujo presentando una infraestructura de pruebas de medición avanzada y la sobrecarga de los mensajes de control.

Presenta una solución muy completa en cuanto a soluciones de redes de sensores SDN y como se detalla en (Miguel, Jamhour, Pellenz, & Penna, 2018) y presenta un manejo simplificado de la creación de flujos.

Existe una variedad de controladores en el mercado con distintas características y se ha descrito cinco de los más conocidos en soluciones SDN. En la Tabla 6, se realiza un breve resumen de sus características, versiones y formatos.

Tabla 6

Comparación de Controladores SDN.

	Soporte OpenFlow	**Lenguaje de Desarrollo**	**Interfaz Gráfica**	**Soporte de Plataformas**	**Código Abierto**	**API Rest**
RYU	OF v1.0, v1.2, v1.3	Phyton	Web	Linux	Si	Si
FloodLigth	OF v1.0	Java	Web	Linux, Mac OS, Windows	Si	Si

OpenDayLight	OF v1.0	Java	Web	Linux, Mac OS, Windows	Si	Si
ONOS	OF v1.0, OF v1.3	Java	Web	Linux	Si	Si
IT-SDN	No	C	QT5	Linux, Windows, Mac oS	Si	Si
µSDN	No	C	No	Linux	Si	Si
SD6WSN	No	C	No	Linux	Si	Si

Fuente: (Centeno, Manuel, Vergel, & Calderón, 2014)

2.2.5 Mecanismos de Cifrado SDN

Existen varias investigaciones en que se implementa un mecanismo de cifrado a las comunicaciones en conjunto con SDN, los cuales varían en función de su objetivo. A continuación, se presenta los más relevantes y se centran en la encriptación como enfoque principal.

Cifrado de extremo a extremo en SDN: de acuerdo con el modelo diseñado por (Amulothu, Huralikupp, Kapur, & Shukla, 2012), propone un sistema donde las comunicaciones cuenten con un cifrado extremo a extremo entre los diferentes componentes de la red y dispongan de conjunto de políticas, varios algoritmos de encriptación que serán negociadas en base a los dispositivos de red. Se plantea un sistema SDN, en que el controlador está a cargo de descubrir los componentes de red, sus políticas y los algoritmos de cifrado. También se presenta un ejemplo donde en caso de descubrir un elemento en la red, este envía su primera información de identificación de algoritmo cifrado soportado al controlador. Luego si este primer elemento de la red quiere comunicarse con un segundo elemento, el controlador primero envía las políticas y algoritmos de codificación, y, en base a una política, se selecciona el algoritmo de cifrado soportado y funcional para la ruta de comunicación entre los dos.

Este modelo de comunicación se repite en función del número de rutas creado entre todos los elementos presentes en la red, es decir, las políticas y el algoritmo de cifrado cambian dependiendo de su ruta y si los elementos tienen soporte para las mismas. En este modelo, todos los elementos de red (incluyendo el controlador) cifran sus comunicaciones. De esta forma, el controlador administra y redirecciona el flujo de forma dinámica mientras mantiene la seguridad en los canales de comunicación.

Codificación de red segura basada en el cifrado de transmisión en SDN: en este sistema se propone un esquema llamado codificación de red de conmutadores seguros (SSNC), el cual agregarán varios grupos de multidifusión a un grupo de multidifusión de acuerdo con los requisitos de los servicios y el estado de la red. Como lo explica (Chen, Jia, Huang, Lan, & Yan, 2016):

> "El controlador enrutará este grupo de multidifusión con codificación de red. Los conmutadores en la misma ruta obtendrán el mismo atributo. Cuando el paquete de datos ingresa al conmutador en SDN y cumple con los requisitos de recursos de la transacción, el conmutador puede descifrar los paquetes de datos recibidos y luego combinar los paquetes de acuerdo con la matriz de codificación".

Tal como expresa (Chen et al., 2016), el proceso inicia cuando el controlador ejecuta el algoritmo llamado Bro.Setup(n) en SDN. Ingresa el número de conmutadores en la red. Luego cada conmutador obtiene el sistema configuración de salida, la llave publica PK y su llave privada d_i. Luego que múltiples grupos multicast son agregados a un grupo, el controlador enruta este grupo por codificación de red para obtener un subconjunto S y el conmutador S_i=i a lo largo del camino. Ingresa la llave pública PK y el subconjunto S_i.

La salida del algoritmo Bro.Encryp (S, PK) es un par de (Hdr, K), donde Hdr es el texto broadcast cifrado y K es una llave de sesión multicast. Porque un nodo malicioso puede

falsificar un par de (Hdr, K), el controlador elije una función hash collision-resistance, y firma el valor hash de Hdr: σ=enc(h(Hdr), Private.rsa) por el algoritmo RSA. Así el mensaje (S, Hdr, σ) es difundida al conjunto S.

Cuando los conmutadores obtienen el mensaje broadcast (S, Hdr, σ), tienen que confirmar la identidad del controlador. Así, los conmutadores desencriptan σ para obtener h(Hdr)= dec(σ, Public.rsa) y calcular y revisar h(Hdr) por la función H como la entrada Hdr. Después que el mensaje pase la verificación, ingresa un subconjunto S, la id i del conmutador y la llave privada d_i para el conmutador i, una cabecera Hdr, y la llave publica PK. Luego la salida del algoritmo Bro.Decrypt (S, i, d_i, Hdr, PK) es la llave de sesión multicast. La llave puede ser luego usada para encriptar/desencriptar los datos multicast. Para obtener más detalles sobre el mecanismo de encriptación, se remite al lector a la referencia (Chen et al., 2016).

De esta forma SSNC mejora en los siguientes aspectos:

- El controlador es responsable de administrar el grupo de multidifusión que puede evitar que los atacantes envíen datos y los usuarios ilegales los reciban;
- El controlador autenticará y autorizará el conmutador permitiendo únicamente que conmutadores de confianza que cumplan con los requisitos de servicios se unan a la ruta del árbol de multidifusión;
- Los conmutadores solo envían los datos de acuerdo con la entrada de flujo

Mejora de seguridad extremo a extremo basado en SDN con SSL/TLS: este modelo propone el usar los protocolos de encriptación ya conocidos como SSL/TLS el cual pretende mejorar su funcionamiento aprovechando de las negociaciones de los mensajes handshake que intercambian las entidades involucradas. Tal y como explica (Ranjbar, Komu, Salmela, & Aura, 2016), la fase de mensajes handshake ocurre al inicio de la comunicación para aceptar parámetros y credenciales a fin de establecer una sesión cifrada. Mientras la carga útil de las

aplicaciones se encuentra encriptada, el proceso de negociación de mensajes se realiza en formato de texto plano dejándolo vulnerable.

La solución propuesta para verificar la negociación TLS se basa en la arquitectura centralizada de SDN, donde, un punto lógico centralizado como el controlador puede inspeccionar todo el tráfico de la red y aplicar políticas uniformes en todo el flujo de red. Dado que los flujos de datos en TLS están encriptados, el controlador solo verifica las credenciales y los parámetros en texto claro transmitidos durante la fase de negociación. Con esto es suficiente para detectar, certificados o versiones de protocolos desactualizadas (por ejemplo). La violación de las políticas puede generar alarmas al operador e incluso bloquear todo el flujo TLS. Luego de una negociación exitosa el controlador permite al cliente TLS y al servidor establecer una sesión encriptada extremo a extremo.

En esta implementación según manifiesta Ranjbar (2016), los conmutadores interceptan todos los mensajes de negociación (handshake) y los inyectan hacia su controlador. El controlador consulta su “policy-box”, para empatar cada paquete con las políticas de la red. Cuando la policy-box acepta el mensaje handshake, el controlador entrega directamente al destino el mensaje pospuesto temporalmente, evitando a los conmutadores del núcleo de la red. La policy-box repite este procedimiento para todo intercambio de mensajes handshake entre el cliente y el servidor hasta que todos los mensajes sean correctamente verificados. Finalmente, el controlador instala las reglas de flujo de reenvío en todos los conmutadores a lo largo del camino entre los puntos finales. Para obtener más detalles sobre el mecanismo de encriptación, se remite al lector a la referencia (Ranjbar et al., 2016).

2.2.6 Seguridad IoT con SDN

IoT hacer referencia a millones de dispositivos conectados a la red produciendo una cantidad enorme de datos en cada segundo y cuyo almacenamiento, automatización y

administración es una tarea intensiva. Estos dispositivos están contantemente bajo amenaza debido a sus características y comunicación a través de un medio de transmisión cableado o inalámbrico. En este sentido SDN se considera una herramienta eficaz para tener control centralizado de los flujos de información en la red y proporcionar una política de seguridad preventiva (Khan et al., 2016).

De igual forma, está implícito que las redes IoT la forman múltiples dispositivos en distintas ubicaciones geográficas; la administración de privacidad en cada una se transforma en una tarea desafiante, pero puede mejorar con la incorporación de un punto único de gestión SDN que, al ser flexible en su configuración, se puede manejar varios dispositivos al mismo tiempo optimizando y mejorando los recursos de la red.

Varias soluciones a la seguridad de WSN en base a SDN han sido propuestas en las cuales cada una resuelve distintos aspectos, pero aún carece de un sistema totalmente seguro y aplicable a todos los casos. En la Tabla 7 se presenta un resumen de las soluciones planteadas desde distintos enfoques y con diversas limitaciones.

Tabla 7

Soluciones IoT basadas en SDN

Enfoque	**Parámetro de Seguridad**	**Red**	**Descripción**	**Limitaciones**
Marco seguro SDN	Autenticación	Red Ad hoc	El controlador bloquea todos los puertos del conmutador al recibir un nuevo flujo e inicia la autenticación	No probar implementación o simulación, solo un marco teórico
DISFIRE	Autenticación & autorización	Red Malla	Red de clúster jerárquica con múltiples controladores SDN implementar un firewall dinámico para garantizar la autorización.	Falta evaluación del marco. El protocolo utilizado es opflex que no está prácticamente probado

Black SDN	Locación Security, Confidencialidad, Integridad, Autenticación y Privacidad.	IoT genérico/ comunicación M2M	asegurar los metadatos y la carga útil mediante encriptación en la capa de enlace y usar el controlador SDN como TTP	La escalabilidad en la red negra creará un peligro al proporcionar una seguridad completa
SDP	Autenticación	Red Ad hoc/ comunicación M2M	SDP recopila las direcciones IP de todos los dispositivos con capacidad de comunicación M2M y las almacena en una red lógica. Y autenticar sobre la base de la información almacenada	La escalabilidad encontrará rendimiento en el caso de IoE
SDIoT	Autenticación	Red genérica IoT	Utilizó el mecanismo SDSecurity que aprovecha NFV y SDP para garantizar un acceso seguro a la red mediante autenticación.	Difícil de administrar la gran red en caso de un solo elemento lógico SDSec. Falta una evaluación experimental

Fuente: (Khan et al., 2016)

Criptografía SDWSN: Software-Defined Wireless Sensor Networking o SDWSN es un modelo que combina SDN con WSN y el cual puede tomar soluciones de ambos paradigmas y ser adaptarlos para ser consideraros de implementar. Teniendo en cuenta esto, algunas soluciones de cifrado WSN puede ser implementado en SDWSN con el fin de verificar si con el plano de control y el plano de datos se puede evitar el consumo excesivo de recursos del cual carecen la mayoría de los nodos en una red WSN. Debido al nuevo modelo de funcionamiento que introduce SDN, los métodos criptográficos tradicionales pueden optimizarse para funcionar en ambientes de redes de sensores inalámbricos, como lo son el algoritmo AES (cifrado simétrico) y el algoritmo RSA (cifrado asimétrico) (Pritchard, Hancke, & Abu-Mahfouz, 2018).

En el caso del algoritmo AES, es un cifrado de bloque que es capaz de procesar entradas de bloques de datos de 128 bits usando tamaños de llaves de 128, 192 y 256 bits. Utiliza cuatro etapas para encriptar los datos de entrada: sustitución de byte usando una tabla S-box;

desplazamiento de filas de la matriz de estado; mezcla de datos dentro de columnas de la matriz de estado; y agregar una clave de ronda al estado.

Como da a conocer (Pritchard et al., 2018), para iniciar el cifrado el mensaje es copiado en una matriz de estado. Después, una clave de ronda es añadida y la matriz de estado es transformada usando una función de ronda. Una función de ronda es una función que aplica las cuatro etapas en el orden que son descritas y dependiendo del tamaño de clave, la función de ronda puede ser implementada 10, 12 o 14 veces, con la ronda final siendo diferente en que no aplica la transformación de columnas mixtas. A partir de entonces, para descifrar el texto cifrado, todas las transformaciones de cifrado se invierten en el orden inverso produciendo un cifrado inverso.

En caso del algoritmo RSA, la idea detrás de PKC es tener una clave pública de cifrado y una clave privada de descifrado, entonces, solo la clave correcta será capaz de descifrar el mensaje. Debido al hecho que las claves de encriptación son públicas, ambas claves deben ser producidas de forma que las claves de descifrado no sean fácilmente deducidas. A fin de verificar el origen del mensaje, firmas digitales son usadas en la clave de descifrado. El algoritmo utiliza dos números primos para crear un número primo grande que luego se usa para cifrar y descifrar un mensaje. Para crear una clave segura y un esquema de cifrado, el proceso requiere grandes funciones matemáticas y muchas multiplicaciones, razón por la cual el algoritmo RSA demanda de más recursos en hardware de las cuales disponen los dispositivos de una red WSN.

CAPITULO III: SOFTWARE Y MECANISMO DE SEGURIDAD A UTILIZAR

El desarrollo de este capítulo comprende la selección de las diferentes herramientas que se utilizan a lo largo del proyecto. Para ello, se describe el funcionamiento, características y parámetros de los elementos necesarios con el propósito de implementar cada uno de los diferentes tipos de software en una solución en conjunto.

Para diseñar un escenario de pruebas, es necesario elegir los diferentes componentes que conforman la red de comunicación. En el capítulo anterior se describió de forma general los elementos que intervienen en el desarrollo, por lo tanto, en este capítulo se enlista y selecciona aquellos que cumplen con los requerimientos para una simulación de redes SDWSN.

En la selección se debe considerar que el sistema de comunicación es inalámbrico y con recursos limitados el cual está conformado por: un algoritmo de encriptación ligero, un ambiente de simulación, un controlador y nodos sensores. El controlador central es quien administra las reglas de flujo para la comunicación en la red. Los elementos seleccionados deben permitir la modificación de los programas para implementar nuevas funciones.

Finalmente, en la selección de las herramientas software, se utiliza un método de selección que permite optar por la mejor opción en base a los requerimientos del proyecto. Por tal razón, para la selección de las cuatro herramientas se utiliza el modelo de la norma ISO/IEC/IEEE 29148:2018 en su enmienda del año 2018, que presenta los requisitos y parámetros que permite la selección de software para un proyecto de implementación el cual se detalla en la sección “9.6 Especificación de los Requisitos de Software (SRS)”, que se encuentra adjunta en el ANEXO A.

3.1 Selección del algoritmo de encriptación

Para la selección del mecanismo de encriptación mediante la norma ISO/IEC/IEEE 29148, el algoritmo debe cumplir con ciertos parámetros establecidos dado que es quien brinda la seguridad en el envío de información y no debe crear sobrecarga en los dispositivos. Principalmente el algoritmo seleccionado debe ser de encriptación ligera por bloques dado que este tipo de encriptación es la más apta para dispositivos con capacidades limitadas y pocos recursos como lo afirma (Dhanda et al., 2020).

En lo que respecta al "propósito" del mecanismo de encriptación, su finalidad es cifrar y descifrar la información que genera cada nodo sensor en la red SDWSN (teniendo en consideración que los nodos de una red WSN cuentan con recursos limitados) creando una comunicación cifrada de extremo a extremo entre los nodos sensores y el controlador y que no genere una sobrecarga en el funcionamiento y rendimiento del dispositivo.

Por otra parte, el alcance consiste en las funciones que debe tener el software al implementarse; en este sentido**,** el algoritmo de encriptación debe cumplir con los siguientes objetivos dentro de la red:

- Cifrar información de extremo a extremo entre los nodos sensores y el controlador de red, siendo compatible con una red SDN y WSN.
- Funcionar sobre nodos o motas con recursos limitados (batería, procesamiento, almacenamiento) y sobre el controlador de red.

Del mismo modo, en la perspectiva del producto se plantea que el software funcione sobre nodos con recursos limitados, encriptando la información de cada uno de los nodos, permitiendo una fácil integración con el protocolo 6LoWPAN y sus características.

Igualmente, las funciones del producto del mecanismo de encriptación, debe cumplir con los siguiente en la red:

- Encriptar y desencriptar la información generada por los nodos y el controlador.
- Funcionabilidad sobre una red SDWSN.
- Compatibilidad con el protocolo 6LoWPAN.
- Seguridad en los enlaces de comunicaciones entre los nodos y el controlador.
- Consumo mínimo de los recursos al ser implementado.

Así mismo, el software debe presentar características de usuarios pertenecientes a capas de nivel superior dado que los usuarios son estudiantes, docentes e investigadores universitarios en el área de las redes de comunicaciones, quienes tienen experiencia en el manejo de redes IT, pero no tienen suficiente conocimiento de redes SDWSN.

De la misma manera, las limitaciones con respecto al mecanismo de encriptación son: pocos recursos utilizables en los dispositivos restringido; el algoritmo de encriptación por debe ser por bloques ligero; el tamaño de bloque debe ser reducido; solo se podrá trabajar con el protocolo de comunicación 6LoWPAN.

En cuanto a las suposiciones y dependencia del mecanismo de encriptación, los requisitos propuestos en el proyecto pueden variar, puesto que es un proceso dinámico. Dentro del mecanismo de encriptación, se supone lo siguiente:

- El mecanismo debe es compatible con el lenguaje de programación C, debido a que la mayoría de los nodos 6LoWPAN operan con este lenguaje.
- El mecanismo puede funcionar en redes SDWSN cifrando la información.
- Su consumo no es mayor a los recursos que los sistemas actuales ya usan.
- Compatible con todos los dispositivos de la red.

Por consiguiente, para la selección del algoritmo de encriptación en la Tabla 8 se presentan los requisitos funcionales que el mecanismo criptográfico debe cumplir para ser elegido. La prioridad viene dada por valores de 1 a 3, donde 3 es el valor máximo y 1 es el valor mínimo.

Un valor de cero significa que la función no cuenta con la característica. En el último cuadro, se calcula el valor total para determinar que algoritmo de encriptación es apropiado.

Tabla 8

Requerimientos funcionales de los mecanismos de encriptación.

Nro.	Nombre	Característica	Descripción	Prioridad		
				AES	SIMON	SPECK
REQ 01	Tamaño de bloque	La cantidad de datos que es capaz de cifrar al mismo tiempo	El tamaño de bloque debe ser adecuado para no crear sobrecarga. Menor a 96 bits.	1	3	3
REQ 02	Tamaño de clave	Con el cual se cifra y descifra los datos	Debe ser lo más reducido posible para acortar almacenamiento. Menor a 128 bits.	0	3	3
REQ 03	Número de rondas	El número de rondas que deben cumplir para encriptar la información	Mientras menor sea el número de rondas, menor procesamiento consumirá. Menor a 32 rondas.	3	1	3
REQ 04	Estructura	El tipo de estructura de cifrado	La estructura Feistel usa el mismo algoritmo para cifrar y descifrar.	0	3	3
REQ 05	Seguridad	La encriptación y desencriptación es lo suficientemente segura para no ser vulnerada.	No se puede visualizar el contenido mediante sniffers de red.	2	3	3
		Total		6	12	15

Fuente: Autoría

Al igual que el proceso anterior, en la Tabla 9 se describen los requisitos no funcionales que el mecanismo de encriptación debe cumplir para ser implementado.

Tabla 9

Requisitos No Funcionales de los mecanismos criptográficos.

Nro.	Nombre	Característica	Descripción	Prioridad		
				AES	SIMON	SPECK
REQ 01	Tipo de cifrado ligero	Cifrado por bloques	Dado que el cifrado por bloques consume menos recursos, el algoritmo debe pertenecer a este grupo.	2	3	3
REQ 02	C	Lenguaje de programación	El mecanismo debe ser estar desarrollado en C.	2	3	3
REQ 03	Licencia	Licencia abierta	Debe manejar licencias de código abierto	3	3	3
REQ 04	Implementación	Disponibilidad del algoritmo	Debe existir la implementación del algoritmo en páginas oficiales.	2	3	3
REQ 05	Actualización	Documentación actualizada	Debe haber una documentación actualizada.	2	3	3
REQ 06	Desempeño		Debe tener altos niveles de seguridad.	1	2	3
		Total		12	17	18

Fuente: Autoría

De la misma manera, el mecanismo de seguridad debe cumplir con ciertos requisitos de rendimiento para funcionar correctamente en todos los dispositivos que conforman la red de comunicación inalámbrica en forma simultánea, sin afectar su rendimiento y siendo compatible con la tecnología de comunicaciones IEEE 802.15.4.

Finalmente se presenta la justificación y de acuerdo con los resultados que presentes en la Tabla 10 de la evaluación para la selección del software ISO/IEC/IEEE 29148:2018, el mecanismo de seguridad Speck de cifrado ligero por bloques presenta las mejores características de encriptación y menor impacto sobre el rendimiento de los dispositivos

adaptándose idealmente al despliegue del proyecto e igualmente como lo afirma (Dhanda et al., 2020), demostrando que, para redes de dispositivos restringidos, los algoritmos de cifrado de bloques son los más apropiados y mencionando que un bloque de cifrado por bloques debe tener un tamaño de bloque de 32–64 bits, en comparación al tradicional de 64-128 bits.

Tabla 10

Resumen de resultados de requisitos

	AES	**SIMON**	**SPECK**
Requisitos Funcionales	6	12	15
Requisitos No Funcionales	12	17	18
Total	18	29	33

Fuente: Autoría

Speck presenta mejores resultados frente a los otros métodos de encriptación por bloques, así como lo demuestra de igual forma la investigación presentada por (Alassaf & Gutub, 2019), donde realiza un benchmark de algoritmos de cifrado ligero por bloques AES, SPECK y SIMON determinando que el método de cifrado por bloques SPECK es más optimo a ser implementado en función de su seguridad, consumo de energía, consumo de almacenamiento y velocidad con respecto a SIMON y AES.

3.2 Simulador de red para el ambiente controlado

Para crear un escenario de pruebas, es preciso definir un software de simulación que tenga capacidad de integrar tanto redes WSN como SDN y al igual que las demás herramientas, tiene que ser de código abierto.

De acuerdo al estudio realizado por (Gonzalez, Flauzac, & Nolot, 2018) señala que, aunque en la actualidad existen diversas plataformas de simulación de redes que tienen soporte para SDN como Ns2 y Opnet, estas no ofrecen escenarios prácticos de escalabilidad ni adaptabilidad a posibles situaciones del mundo real. También existen otras plataformas mejor desarrolladas

como Mininet, pero como expresa Gonzales (2018), debido a su enfoque, no tiene soporte para múltiples dispositivos de redes restringidas. Todas están enfocadas a soportar principalmente SDN para redes tradicionales de IT, sin consideran a las redes WSN o IoT.

Continuando con la selección del simulador mediante la norma ISO/IEC/IEEE 29148, el propósito del simulador es contener a todos los elementos que comprenden la red WSN basados en la comunicación inalámbrica IEEE 802.15.4 los cuales son implementados y simulados, permitiendo la integración de estos mediante una interfaz gráfica permitiendo manipular el entorno en tiempo real.

En lo referido al alcance del ambiente de simulación, este debe ser capaz de contener, controlar y administrar un controlador de red, nodos de red inalámbricos con comunicación basada en 802.15.4, permitiendo una conexión de todos los dispositivos presentes en la red. Así mismo, debe ser capaz de soportar la implementación de un algoritmo de encriptación.

Se planea que el simulador tenga una perspectiva de producto que brinde las herramientas para crear un ambiente controlado de pruebas, logrando obtener datos previa y posterior implementación del mecanismo de encriptación en la red SDWSN, permitiendo observar la interacción y comportamiento de los nodos de red.

Las funciones que se proyecta que el simulador de red debe cumplir son las siguientes:

- Permitir iniciar un controlador para administrar los flujos de red.
- Permitir la integración de nodos sensores basados en un sistema operativo open-source.
- Permitir la implementación de un algoritmo de encriptación en todos los elementos de la red.
- Dar una vista global de la red y controlar las variables de simulación.
- Permitir la adición de distintos tipos de nodos de acuerdo con su función (controlador, sensor)

El simulador debe presentar características de usuarios pertenecientes a capas de nivel superior debido a que está dirigido a estudiantes, docentes e investigadores universitarios en el área de las redes de comunicaciones, quienes tienen experiencia en el manejo de redes IT, pero no de redes SDWSN.

Las limitaciones del software de simulación están dadas por sus requisitos funcionales del hardware:

- El número de nodos sensores depende de la capacidad de recursos del host.
- Debido a que es un ambiente controlado, las pruebas se tienen que realizar de forma local.
- La comunicación inalámbrica únicamente debe ser con el protocolo 6LoWPAN.

Por otra parte, las **s**uposiciones y dependencias durante el despliegue de la simulación de una red SDWSN son las siguientes:

- El software simulador es compatible con Linux debido a que es un proyecto basado en código abierto.
- El software puede cambiar la topología de la red en cualquier momento de la simulación.
- Los registros de comunicaciones realizadas durante la simulación son almacenables para posterior análisis.
- Obtener estadísticas de las comunicaciones realizadas entre los diferentes elementos.

A continuación, en la Tabla 11 presenta los requisitos funcionales para la selección del software de simulación para la red SDWSN. La prioridad está dada por los valores de 1 al 3, donde, 1 es el valor mínimo, 2 el valor medio y 3 el valor máximo. Así, se puede determinar cuál de los simuladores es el más apto para su implementación.

Tabla 11

Requisitos funcionales del simulador de red SDWSN.

Nro.	Nombre	Característica	Descripción	Prioridad		
				Mininet-iot	Netsim	Cooja
REQ 01	Conexión inalámbrica basada en 802.15.4	Soporte de 6LoWPAN	El simulador debe realizar la comunicación mediante el protocolo 6LoWPAN.	3	1	3
REQ 02	Mecanismos criptográficos	Implementación del mecanismo criptográfico Speck.	El simulador debe permitir la implementación del mecanismo de cifrado en las comunicaciones.	1	1	3
REQ 03	Controlador de red	Soporte de arquitectura SDN	Debe permitir la integración de controlador y nodos de red	3	3	3
REQ 04	Simulación de redes WSN	Despliegue de elementos de red inalámbricos	El simulador debe soportar varios elementos de red simultáneamente	3	3	3
REQ 05	Registros de actividad	Creación de archivos para posterior análisis.	Debe ser capaz de crear archivos que puedan ser analizados por sniffers de red.	1	3	3
		Total		11	11	15

Fuente: Autoría

De igual forma, en la Tabla 12 se presentan los requisitos no funcionales que el simulador deben cumplir para ser seleccionado.

Tabla 12

Requisitos No funcionales del simulador de red SDWSN.

Nro.	Nombre	Característica	Descripción	Prioridad		
				Mininet-iot	NetSim	Cooja
REQ 01	Open Source	Código abierto	El sistema debe ser gratuito para utilizar todas sus funciones	3	1	3
REQ 02	Actualizaciones	Soporte y documentación	El simulador debe estar actualizado y documentado.	2	3	2
REQ 03	Compatibilidad	Lenguaje de programación	Debe ser compatible con el lenguaje de programación C.	0	1	3
REQ 04	Interfaz gráfica	Vista global.	Debe proveer una vista grafica en tiempo real de la simulación.	3	3	3
		Total		8	8	11

Fuente: Autoría

Además de los anteriores requerimientos, también debe cumplir con requisitos de rendimiento**.** La instalación del software de simulación se realiza sobre un sistema operativo en base Linux, debiendo así, ser compatible con software libre y ser capaz de soportar la implementación varios elementos de red como un controlador para la red WSN y nodos sensores.

De esta manera, se presenta una justificación en base a las matrices de evaluación realizadas para la selección de software. En la Tabla 13 se presenta un resumen donde se evidencia que el simulador de redes que cumple con la mayoría de las características necesarias es Cooja que tiene todas las funcionalidades descritas. Una de sus particularidades que lo hace elegible es que esta desarrollado para redes inalámbricas de baja potencia e incorpora una pila de comunicación TCP/IP. Además, es un software libre cuenta con una interfaz gráfica de usuario.

A diferencia de Netsim que si soporta redes SDWSN su desventaja es que es un software de pago y no soporta el protocolo 6LoWPAN (NetSim™, n.d.) y mininet que si bien soporta SDN y 6Lowpan no cuenta con una integración entre ambas arquitecturas , Cooja ya ha sido probada y utilizada por varias investigaciones para la implementación de SDN y 6LoWPAN en distintas soluciones factibles.

Cuenta con una guía de instalación en Linux y con una documentación variada que hace que su instalación y uso sea simple para redes SDWSN debido a que Cooja está desarrollado en base al simulador de redes inalámbricas Contiki con parches adicionales para que también funcione como una red SDN.

Tabla 13

Resumen de evaluación de simuladores de redes

	Mininet	**Netsim**	**Cooja**
Requisitos Funcionales	11	11	15
Requisitos No Funcionales	8	8	11
Total	19	19	26

Fuente: Autoría

3.3 Controlador de red para redes WSN

El controlador es la pieza más importante dentro de la comunicación debido a que controla y administra los flujos de red en la SDWSN. En la capitulo anterior, se presentó los controladores más populares para redes IT que se podrían utilizar. Sin embargo, debido a varios de ellos no está adaptado para redes WSN o comunicaciones en base a IEEE 802.15.4, solo se toman en cuenta aquellos diseñado para este tipo de redes inalámbricas.

El propósito principal del controlador es administrar y controlar el flujo de la red inalámbrica SDWSN permitiendo una gestión más simple con reglas de flujo que permitan

agilizar el tráfico en la red. De igual forma, debe tener una vista global de la red para la toma de decisiones de enrutamiento.

Así mismo, el alcance que se proyecta para el controlador contempla que debe ser capaz de conectarse en la red SDWSN a los nodos sensores con el protocolo de comunicación 6LoWPAN e implementar un algoritmo que permita la encriptación de la información entre todos los elementos en la red pero que no genere una sobre carga sobre los mismos.

Igualmente, La perspectiva de producto que se espera del controlador es que sea un elemento centralizado de software libre que permita controlar y administrar las reglas de tráfico y proporcione una capa de seguridad a los dispositivos que se encuentran en la red.

Además, el controlador debe cumplir con las siguientes funciones de producto:

- Permitir la conexión entre los diferentes dispositivos en la red SDWSN.
- Permitir la implementación del mecanismo de encriptación (previamente seleccionado).
- Ser compatible con el simulador de redes inalámbricas Cooja.
- Permitir la comunicación mediante el protocolo 6LoWPAN
- Administrar las comunicaciones dentro de la red.
- Crear un mapa topológico de la red en general.

Por otra parte, el controlador debe presentar características de usuarios pertenecientes a capas de nivel superior debido a que está dirigido a estudiantes, docentes e investigadores universitarios en el área de las redes de comunicaciones, quienes tienen experiencia en el manejo de redes IT, pero no de redes SDWSN.

Las limitaciones con las que debe operar el controlador dentro del ambiente de simulación son:

- Administrar varios dispositivos al mismo tiempo.
- Debe estar basado en software libre
- Desencriptar la información procedente de todos los nodos en la red.
- Su protocolo de comunicación únicamente es 6lowpan.

Por otra parte, en lo referido a **s**uposiciones y dependencias durante la implementación del software controlador para la red SDWSN se espera:

- Compatibilidad del controlador con la herramienta de simulación Cooja y con el mecanismo criptográfico Speck.
- El mecanismo de encriptación Speck se funciona adecuadamente en el controlador.
- El controlador recibe información de los demás elementos en la red mediante encriptación.
- El controlador instala y administra las reglas de flujo para la comunicación.

Ahora bien, en la Tabla 14 se presenta los requisitos funcionales para la selección del software controlador de la red SDWSN. La prioridad se encuentra dada por valores del 0 al 3 donde, 0 es el valor mínimo, indicando que no cuenta con esa característica y 3 es el valor máximo. Al final se calcula el valor de cada elemento y conocer cuál es el más adecuado a ser implementado.

Tabla 14

Requisitos funcionales del controlador de red.

Nro.	**Nombre**	**Característica**	**Descripción**	**Prioridad**		
				SD6WSN	(ONOS) SDNWISE	IT-SDN

REQ 01	Mecanismo Criptográfico	Implementación de mecanismos criptográficos.	Debe soportar la integración del mecanismo criptográfico Speck.	3	2	3
REQ 02	API Rest	Arquitectura de desarrollo	Debe poseer una API Rest para la manipulación de la red	3	2	3
REQ 03	Sistema Operativo	Soporte de varios sistemas operativos ligeros	El controlador debe soportar a varios dispositivos al mismo tiempo.	3	3	3
REQ 04	Redes WSN	Funcionamiento con redes inalámbricas de sensores	Soporte del protocolo de comunicación 6LoWPAN	3	2	1
REQ 05	Simulador de red	Basado en el simulador de red.	Debe ser compatible con el software simulador Cooja.	3	2	3
REQ 06	Reglas de flujo	Instalación de reglas de flujo en los nodos.	Debe tener la capacidad de crear reglas de flujo.	3	1	3
		Total		18	12	16

Fuente: Autoría

Igualmente, la Tabla 15 presenta los requisitos no funcionales que el controlador de red debe cumplir para que ser seleccionado.

Tabla 15

Requisitos no funcionales del controlador de red.

Nro.	**Nombre**	**Característica**	**Descripción**	**Prioridad**		
				SD6WSN	SDN-WISE	IT-SDN
REQ 01	Soporte	Disponibilidad de documentación	Debe contar con documentación de instalación en software libre.	3	0	3
REQ 02	Registros	Disponibilidad de manuales de uso	Constancia de creación de reglas de flujo.	3	2	3

REQ 03	Mercado	Tiempo de actividad	El controlador debe contar con actualizaciones periódicas.	0	0	1
REQ 04	Funcionalidad	Ambientes de ejecución	Se debe instalar y ejecutar un ambiente básico de pruebas.	3	1	3
REQ 05	Open Source	Código abierto	El sistema debe ser gratuito para utilizar todas sus funciones	3	3	3
		Total		12	9	13

Fuente: Autoría

Además de los anteriores requerimientos, el controlador también debe cumplir con requisitos de rendimiento**.** El controlador de red debe funcionar directamente con el software de simulación Cooja siendo el principal elemento en la red y quien controla todo el flujo de comunicación. Al momento de integrarlo a la red deberá monitorear, administrar y generar reglas de seguridad para una comunicación cifrada extremo a extremo.

De esta manera se presenta la justificación de la selección. En la Tabla 16 se presenta un resumen de los resultados de la evaluación de los diferentes controladores de red para redes de bajos recursos.

Tabla 16

Resumen de evaluación de simuladores de redes

	SD6WSN	**SDN-WISE**	**IT-SDN**
Requisitos Funcionales	18	12	16
Requisitos No Funcionales	12	9	13
Total	30	21	29

Fuente: Autoría

En base a los resultados obtenidos en la evaluación realizada, se determina que la solución del controlador de red SD6WSN presentado en (Miguel et al., 2018) es el más apto para su implementación debido a que cumple con los requisitos necesarios para el proyecto

sobrepasando apenas a IT-SDN (Alves et al., 2017) debido a que este último no cuenta con el soporte para protocolo inalámbrico 6LoWPAN sino funciona sobre un protocolo más ligero (RIME) y SD6WSN. Uno de los factores por los cuales fue seleccionado, es su compatibilidad con el simulador de redes en el cual fue desarrollado SD6WSN.

De igual forma, ya ha sido utilizado anteriormente en otros proyectos de investigación de rendimiento, arrogando resultados positivos con respecto a su uso e implementación.

3.4 Sistema Operativo Ligero para simular los nodos de red

El sistema base de un nodo restringido necesariamente debe ser un SO ligero, puesto que estos sistemas fueron optimizados para funcionar sobre plataformas con recursos limitados de procesamiento, energía y almacenamiento. Por ende, en el capítulo anterior se presentó los diferentes SO para nodos de bajos recursos de código abierto.

El propósito general del sistema operativo es simular las motas o nodos de red inalámbricos dentro de una SDWSN basados en el protocolo de comunicación IEEE 802.15.4 y que permita la manipulación de su código fuente para que sea capaz de integrar un algoritmo de encriptación para la comunicación en un controlador central y en los nodos de red.

En cuanto al alcance del software, el SO ligero debe permitir el protocolo de comunicación 6LoWPAN con características de recolección o generación de datos, comunicación inalámbrica y funciones SDN.

Se espera que el SO ligero tenga una perspectiva de producto en funciones como: funcionar dentro de un ambiente de pruebas simulado y gestionar las funciones de cada nodo. También se espera que el sistema operativo también soporte mecanismos de encriptación y sea compatible con IEEE 802.15.4 para de cifrar la comunicación entre los distintos elementos que conforman la red.

El sistema operativo para las motas o sensores debe cumplir con las siguientes funciones de producto:

- Comunicarse con todos los componentes en la red.
- Tener conectividad con el controlador de red
- Soportar un algoritmo de seguridad para su comunicación
- Ser de código abierto y que permita manipular su algoritmo de funcionamiento.
- Comunicarse mediante el protocolo IEEE 802.15.4
- Enviar información a través de la red
- Cifrar la cualquier comunicación que realice con otros dispositivos.

Este software debe presentar características de usuarios pertenecientes a capas de nivel superior debido a que los usuarios son estudiantes, docentes e investigadores universitarios en el área de las redes de comunicaciones, quienes tienen experiencia en el manejo de redes IT, pero no de redes WSN.

Por otra parte, las limitaciones que se pueden dar con el software del sistema operativo de las motas están dadas debido a que: cuentan con bajos recursos, y por ello no pueden realizar tareas complejas; un número de nodos limitados, debido a que no debe exceder la capacidad del hardware; en caso de haber cambios, se deben realizar todos los nodos.

Continuando con las suposiciones y dependencias del SO, durante el despliegue de los nodos en la red se espera que tengan el siguiente comportamiento:

- Los nodos son compatibles con el sistema operativo Linux
- Permiten implementar un algoritmo de encriptación para las conexiones
- Permite la comunicación con el controlador para la implementación de reglas de flujo.

- Los datos generados son cifrados en la red.

Por otra parte, la Tabla 17 presenta los requisitos funcionales para la selección del software controlador de la red SDWSN. La prioridad se encuentra dada por valores del 0 al 3 donde, 0 es el valor mínimo, indicando que no cuenta con esa característica y 3 es el valor máximo. Al final se calcula el valor de cada elemento y conocer cuál es el más adecuado a ser implementado.

Tabla 17

Requisitos funcionales del SO ligero.

Nro.	Nombre	Característica	Descripción	Prioridad		
				Contiki	SDN-WISE	TinyOS
REQ 01	Mecanismo Criptográfico	Implementación de mecanismos criptográficos.	Debe soportar la integración del mecanismo criptográfico seleccionado.	3	3	3
REQ 02	Redes WSN	Funcionamiento con redes inalámbricas de sensores	Soporte del protocolo de comunicación 6LoWPAN	3	2	3
REQ 03	Simulador de red	Basado en el simulador de red.	Debe ser compatible con el software simulador Cooja.	3	3	2
REQ 04	Integración	Función en conjunto con el controlador	El SO debe integrarse fácilmente con el controlador de red.	3	2	2
		Total		12	10	10

Fuente: Autoría

Igualmente, en la Tabla 18 presenta los requisitos no funcionales que el controlador de red debe cumplir para que sea elegible por encima del resto.

Tabla 18

Requisitos no funcionales del SO ligero.

Nro.	Nombre	Característica	Descripción	Prioridad		
				Contiki	SDN-WISE	TinyOS
REQ 01	Compatibilidad	Soporte para simulador y nodos	Debe ser de software libre	3	1	1
REQ 02	Soporte	Disponibilidad de documentación	Debe tener una documentación y actualización del sistema.	3	1	2
REQ 03	Registros	Disponibilidad de manuales de uso.	Funcionamiento de los nodos y como cambiar sus características.	3	2	1
REQ 04	Open Source	Código abierto	El sistema debe ser gratuito para utilizar todas sus funciones	3	3	3
		Total		12	7	7

Fuente: Autoría

Además, los requisitos de rendimiento que debe cumplir el SO en los nodos de red es que interactuen dentro del software de simulación en conjunto con el controlador de red. Los nodos son quienes generan toda la información a ser transmitida en la red y por ende deben implementar el mecanismo de encriptación en sus códigos para las comunicaciones.

Finalmente, se presenta la justificación del SO con las mejores prestaciones en cuanto a rendimiento para dispositivos ligeros como se muestra en la Tabla 19.

Tabla 19

Resumen de evaluación de sistemas operativos ligeros

	Contiki	**SDN-WISE**	**TinyOS**
Requisitos Funcionales	12	10	10
Requisitos No Funcionales	12	7	7
Total	24	17	17

Fuente: Autoría

La creación de una red de sensores necesita de dispositivos con capacidad IoT y para ello el sistema operativo Contiki es una de las mejores soluciones en el mercado. Contiene una pila de comunicación para IPv6 de baja potencia y admite la comunicación inalámbrica IEEE 802.15.4 junto con protocolos RPL y 6LoWPAN que son necesarios para establecer la comunicación entre nodos.

Una de las características de ContikiOS es que viene integrado con un ambiente de simulación Cooja que permite desplegar, configurar, controlar y observar los ambientes WSN. Este SO, es uno de los más populares e intuitivos en su utilización. Esta desarrollado bajo el lenguaje de programación C. Al ser abierto, permite la modificación de su código y la aplicación de nuevas funciones a los nodos de red.

3.5 Resumen de software

Finalmente, en la Tabla 20 se presenta un resumen del software seccionado para cada uno de los componentes. implementación con las características y funciones más importantes para el proyecto que tendrán dentro de la simulación.

Tabla 20

Resumen de Software elegido

Software Seleccionado		Función	Open-source	Soporte de 6LoWPAN
Algoritmo de encriptación	Speck	El algoritmo de encriptación por bloques ligeros a ser usado para cifrar las comunicaciones.	✓	✓
Simulador de la red WSN	Cooja	El entorno de simulación que contendrá todos los elementos de red que intervienen	✓	✓
Controlador de la red SD6WSN	SD6WSN	El controlador de red a ser usado dentro de una comunicación basado en 802.15.4	✓	✓

Sistema operativo ligero	Contiki Os	El sistema operativo que funcionara dentro de los nodos de red.	✓	✓

Fuente: Autoría

El elemento más importante sobre el cual los demás elementos deben basarse es el simulador Cooja, debido a que es quien controla y administra los parámetros de simulación, en adición, cabe recalcar que es de código abierto, lo que permite adaptar a la necesidad del proyecto. Por otra parte, también tiene la capacidad de generar archivos pcap, los cuales pueden ser analizados mediante el sniffer de red Wireshark.

CAPITULO IV: DISEÑO Y CONFIGURACIÓN DE LA RED SDWSN

Este capítulo comprende el diseño e integración de los programas previamente seleccionados en una solución conjunta de encriptación y comunicación dentro de un ambiente de pruebas SD6WSN.

En primera instancia, se define la distribución de los elementos seleccionados dentro de la arquitectura de red WSN y SDN dividiéndolos en capas según su función dentro del sistema. Luego, se procede al desarrollo del ambiente de pruebas precisando las plataformas sobre las cuales funcionan, dimensionando su capacidad, diseñando e implementando la topología de despliegue y finalmente ejecutando el sistema de simulación.

A continuación, se describe el funcionamiento del controlador de red dentro del sistema y como realiza la instalación de reglas de flujo dentro de cada dispositivo para alcanzar al controlador y viceversa; de igual forma, se realiza la comprobación del funcionamiento del sistema previo a la implementación de la encriptación, con el fin de evidenciar el correcto funcionamiento del ambiente de pruebas SD6WSN.

Además, se detalla el proceso de implementación y adaptación del algoritmo de encriptación Speck dentro de la programación del controlador y los nodos con el fin de verificar su comportamiento dentro de las plataformas de simulación. Finalmente, se integra los dispositivos con función de encriptación dentro del ambiente previamente diseñado y desplegado con el propósito de verificar que los mensajes creados por los nodos son encriptados y desencriptados por el controlador.

4.1 Dimensionamiento de la arquitectura de red SDWSN

Una arquitectura para SDWSN típica contempla todos los elementos de conmutación y enrutamiento divididos por propósito. La arquitectura utilizada para la red en desarrollo está basada la arquitectura en capas del modelo SDN y es similar a la arquitectura oneM2M de IoT en la separación de sus sistemas en base a su propósito. Este modelo cuenta con tres niveles o capas donde se encuentran los distintos elementos utilizados, ubicados de acuerdo con su función, por ejemplo, la capa de control esa compuesta por el controlador, en la capa de infraestructura se encuentran los dispositivos de conmutación (también conocidos como nodos). En la Figura 25, se muestra gráficamente como estos niveles y los componentes están distribuidos para su funcionamiento y su relación para la comunicación de todo el sistema.

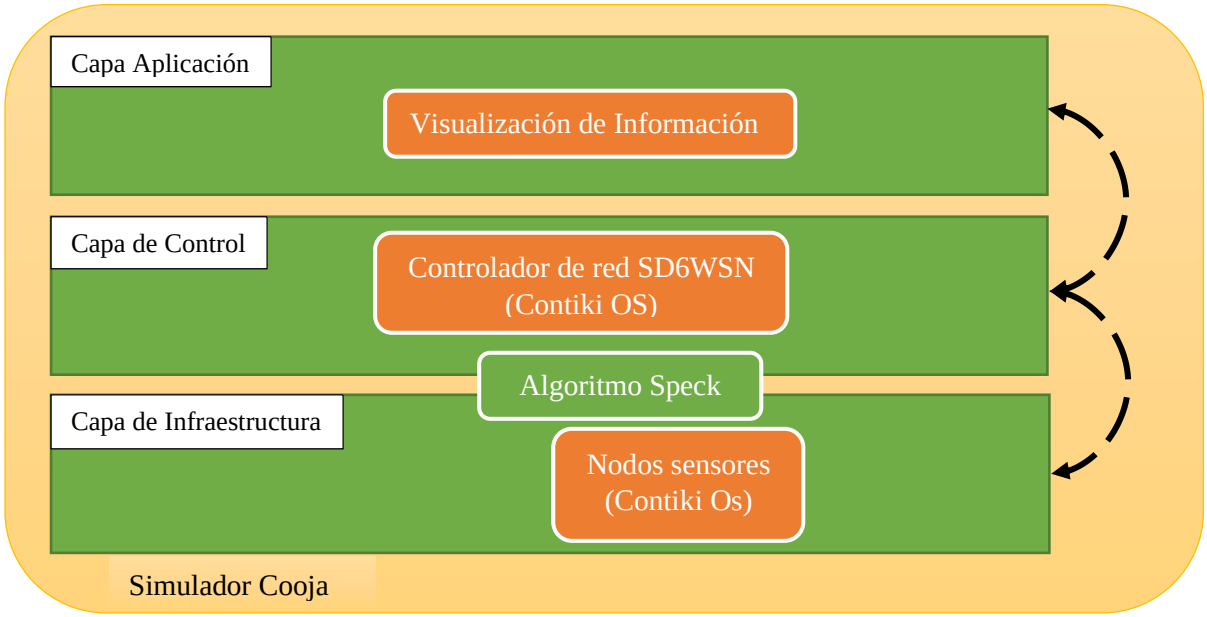


Figura 25. Arquitectura SDN-WSN basada en capas

Fuente: Autoría

En la Figura 25, se observa que el sistema establecido parte desde simulador de redes WSN Cooja como el entorno de funcionamiento y sobre el cual se ha desarrollado la arquitectura dividida en tres niveles (típicos de sistemas SDN): capa de infraestructura, capa de control y capa de aplicación.

4.1.1 Capa de infraestructura

Esta es la capa de los dispositivos terminales IoT y puntos finales tales como sensores/actuadores que realizan alguna acción especifica, recolectan información del entorno o transmiten información. En este nivel de acuerdo con la arquitectura planteada, se encuentran los nodos basados en el sistema operativo Contiki que generan la información para posteriormente enviar al controlador. De igual forma, aquí se encuentra la integración del algoritmo de encriptación SPECK.

Nodos de red: constituye uno de los elementos de red del sistema que cumplen con la función de establecer la comunicación con el controlador para enviar y recibir información. Debido a que son dispositivos con escasos recursos, estos no pueden contar con mayor procesamiento, almacenamiento y energía. Están diseñados e integrados bajo el sistema operativo para dispositivos de bajos recursos ContikiOS el cual tiene incorporado el protocolo de comunicación 6LoWPAN basados en la tecnología de acceso inalámbrico 802.15.4 con el protocolo de enrutamiento RPL. La función del nodo es enviar actualizaciones al controlador de la información recolectada y de sus vecinos inmediatos. Al ser nodo adaptados a un funcionamiento SDN cuentan con dos componentes: agente y enrutamiento del plano de control. Las funcionalidades principales del agente son: enviar información hacia el controlador; administrar las entradas en las tablas de flujo; inspeccionar las cabeceras de los paquetes del plano de datos; enviar o descartar datos de acuerdo con las coincidencias en la tabla de flujos; enviar notificaciones al controlador cuando no se encuentra coincidencia de paquetes; interactuar con el sistema operativo para cambiar parámetros de transmisión.

A continuación, se presenta una lista de las funcionalidades que cumplen los nodos de red, las cuales están acorde con las descritas en la selección de software del sistema operativo.

- Generan información

- Comunicación por protocolo 802.15.4
- Compatibilidad 6LoWPAN
- Envió de información
- Función de encriptación

Este elemento de red al estar basado en la solución SD6WSN, cuenta con características de hardware importantes para el funcionamiento del este sistema, por tal motivo, los dispositivos deben contar con las características mínimas de la solución propuesta por (Miguel et al., 2018), (dado que al implementar la encriptación estas pueden superar las utilizadas previamente) las cuales se describen a continuación:

- Comunicación inalámbrica 802.15.4
- Memoria de 16 KB SRAM
- Almacenamiento de 128 KB flash

4.1.2 Capa de control

En esta capa se encuentran los elementos de control tales como el controlador de red y las funciones que vienen integradas en el sistema operativo. Además, para realizar la comunicación con la capa de infraestructura se usa el mecanismo criptográfico Speck para el intercambio de datos entre los nodos y el controlador.

Controlador: es el elemento principal del sistema, que tiene como función la convergencia y establecimiento de la comunicación dentro del sistema SDWSN. El controlador tiene las mismas características que los nodos debido a que está basado en el sistema operativo 6LoWPAN, pero con funcionalidades diferentes. Dado que establece la comunicación dentro de la red, este también cumple la función de comunicarse con redes externas, es decir, tiene la función de enrutador de borde para direccionar información dentro y fuera de la red.

Para la conectividad de la red el controlador cuenta con tres módulos: coordinador, descubrimiento y gestión de topología y control de flujo.

- **Coordinador:** realiza la integración con las aplicaciones de red.
- **Descubrimiento y gestión de topología**: para tener una vista global de la red, el controlador identifica los nodos activos y la calidad de los enlaces los cuales están indicados por las estimaciones de transmisión (ETX). El protocolo SD6WSN, utiliza la información enviada en notificaciones para incluir o excluir nodos de la topología. En caso de inclusión el controlador revisa si el nodo es parte de la tabla de direcciones IPv6 precargada enviando al nodo el mensaje info-get/nbr-etx para obtener sus vecinos y la respectiva calidad de enlaces (ETX). Las decisiones de enrutamiento del controlador son realizadas por el protocolo de enrutamiento para redes de baja potencia y pérdidas (RPL, por sus siglas en ingles), que se encuentran en la capa 3 al igual que el direccionamiento IPv6.
- **Control de flujo:** Los mensajes flow-mod SD6WSNP añaden o remueven entradas en las tablas de flujo y si una entrada ya existe, se sobrescribe. Cuando un paquete llega a un nodo, SD6WSN verifica los campos del encabezado IP de acuerdo con las reglas de coincidencia en la tabla de flujo, buscando un flujo existente. Si se produce una coincidencia, el nodo realiza la acción asociada, es decir, reenvía el paquete en el plano de datos o lo descarta.

De igual forma, el controlador cumple con los parámetros que fueron tomados en cuenta previamente en la selección de software y que se resumen a continuación:

- Instalación de reglas de flujo dentro de los nodos
- Comunicación 6LoWPAN
- Recepción de información de los nodos

- Función de desencriptación

4.1.3 Capa de aplicación

Se encuentran las aplicaciones necesarias para el usuario tales como un visualizador de la información generada por los nodos de red. De acuerdo con la función del sistema se pueden implementar aplicaciones específicas las cuales se deben programar y enlazar.

Visualizador de información: es una aplicación básica que no constituye en sí un elemento de la red, pero es parte importante del sistema para la verificación del correcto funcionamiento de los elementos, tanto en la generación y envió de datos como en la recepción de la información.

4.2 Dimensionamiento del sistema

Para establecer los valores de simulación de los nodos y controlador red, como primer paso se debe conocer las características del hardware sobre el cual se llevará a cabo el entorno de pruebas Cooja, es decir, las características de la PC sobre la cual va a funcionar la simulación, que se encuentran detalladas en la Tabla 21.

Tabla 21

Características de la PC de simulación.

Fabricante	HP
Modelo	Notebook envy 14
Sistema operativo	Ubuntu
Versión	18.04
Procesador	Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz 2.40 GHz
Memoria RAM	2 GB
Disco duro	20 GB

Fuente: Autoría

Sobre el sistema Ubuntu se realiza la instalación del ambiente de pruebas Cooja. El proceso de instalación se lo puede observar en el ANEXO B, donde se explica claramente los pasos realizados para obtener el ambiente de pruebas.

Dado que el desarrollo de un ambiente de pruebas es extenso y progresivo es necesario conocer las características, distribución y proceso de despliegue de los elementos. Para ello, se cuenta con una metodología de implementación por etapas en la que se observa el avance paulatino del desarrollo. Con base en lo anterior, se ha divido el proceso en cinco etapas importantes para el desarrollo (Figura 26), que son: definición de plataforma, diseño de topología de red, dimensionamiento de controlador y nodos, implementación y ejecución del sistema.

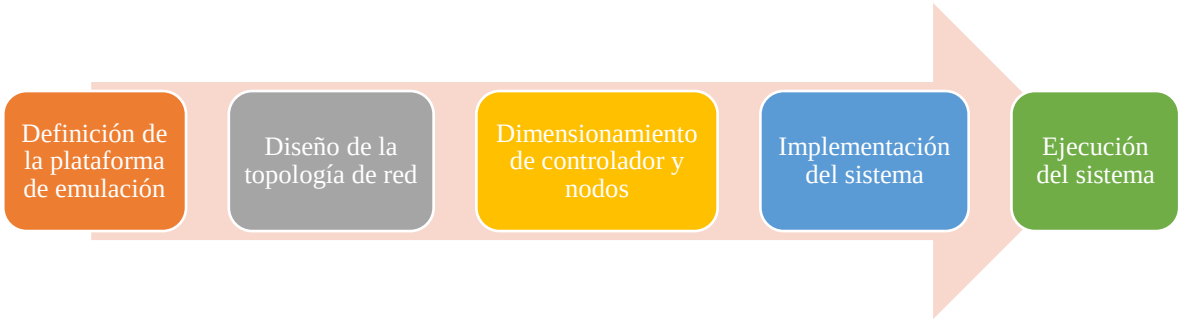


Figura 26. Metodología de implementación del ambiente simulado.

Fuente: Autoría

En función de la arquitectura planteada, se procede a establecer los parámetros de simulación que tendrán cada uno de los elementos que intervienen, como las características de la plataforma sobre la cuales se simulara los nodos, la comunicación inalámbrica (radio), el tiempo de actividad, rangos de transmisión/recepción y velocidad de simulación.

Las configuraciones del ambiente de pruebas vienen establecidas por defecto y se puede realizar ajustes antes de iniciar la simulación. Cooja también permite realizar cambios a los

valores por defecto. En la Tabla 22, se presenta un resumen de las configuraciones que contiene actualmente y con las que se ejecuta el ambiente de simulación para el sistema.

Tabla 22

Configuraciones generales por defecto del ambiente de simulación.

Parámetro de simulación	Valor
Tiempo de ejecución	15 min
Velocidad de ejecución	100%
Rango de transmisión	50m
Rango de interferencia	100m
Tasa de TX/RX	100%
Capa MAC	csma_driver
Capa RDC	nullrdc_driver

Fuente: Autoría

4.2.1 Definición de plataforma

Como se describió anteriormente, los elementos de las capas infraestructura y controlador dependen de las características del hardware sobre el cual se va a emular. El simulador Cooja contempla una gran variedad de plataformas inalámbricas para el despliegue y cada una tiene sus propias características. En la solución SD6WSN propuesta por Miguel (2018), para añadir las funciones SDN estas se adaptaron sobre la plataforma Wismote, por consiguiente, el cambiar de plataforma a otra que no esté adaptada a estas solución causaría errores en la implementación. Por tal razón, se utiliza la plataforma Wismote para la emulación de los nodos.

Wismote TI

El objetivo del proyecto es su funcionamiento sobre SDN y la plataforma de emulacion es necesaria para llevar a cabo el desarrollo, por lo tanto, la encriptación se implementa en la

misma plataforma Wismote de Texas Instruments (Figura 27). Wismote está definida en una placa basada en MSP430 que incluye un módulo inalámbrico IEEE 802.15.4 y con el chip de radio CC2420 para permitir la manipulación del uso de energía, con la función de despertar rápido desde el modo de suspensión. La Tabla 23 presenta un resumen de las características de la plataforma de los nodos.

Tabla 23

Características de la plataforma de simulación Wismote para los nodos

Radio	CC2420
Protocolo de comunicación	IEEE 802.15.4
Plataforma	Wismote
CPU	TI MSP430F5437
Memoria	16-KB SRAM
Almacenamiento	128-KB flash
Frecuencia de reloj	25MHz

Fuente: Autoría

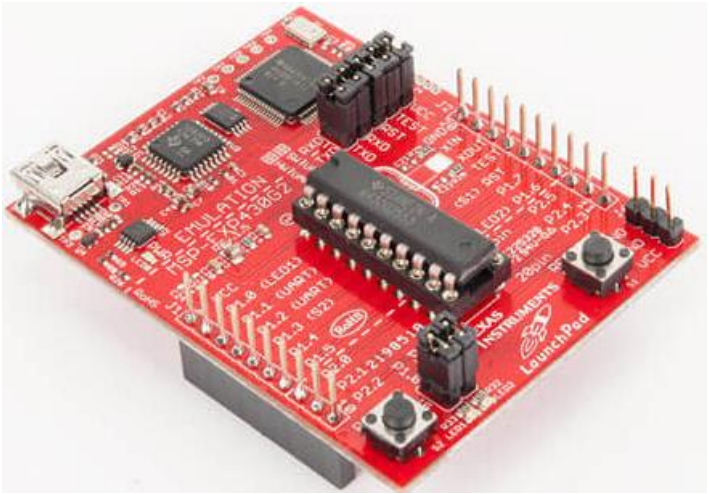


Figura 27. Placa de desarrollo MSP430 de TI

Obtenido de: https://www.ti.com/diagrams/msp-exp430g2_mspexp430g2_lateral.jpg

Como se mencionó anteriormente, esta plataforma debe contemplar varias funciones que los elementos de red necesitan para el correcto funcionamiento del sistema. En la Tabla 24 se muestran una descripción de las características de hardware que necesitan el nodo y controlador de red, así como también las características que la plataforma Wismote cumple.

Tabla 24

Características de la plataforma Wismote

Nodo/Controlador	Plataforma Wismote
Comunicación inalámbrica 802.15.4	✓
Soporte 6LoWPAN	✓
Memoria de 16 KB SRAM	✓
Almacenamiento de 128 KB flash	✓
Funcionamiento de enrutador de borde para conectividad con redes externas	✓

Fuente: Autoría

4.2.2 Dimensionamiento del controlador y nodos

Luego de establecer la configuración que debe tener la red, se procede a su despliegue en el simulador Cooja. Para ello, se procede con la apertura del simulador Cooja desde el terminal de comandos de Linux. Para ejecutar la aplicación (Figura 28), se accede a la carpeta de contiki "tools/cooja", y se inserta el comando "ant run" lo cual abre una ventana de inicio para crear o abrir un proyecto.

```
anthony@ubuntu:~$ cd contiki/tools/cooja/
anthony@ubuntu:~/contiki/tools/cooja$ ant run
Buildfile: /home/anthony/contiki/tools/cooja/build.xml
```

Figura 28. Ejecución de Cooja**.**

Fuente: Autoría

Para iniciar con un nuevo ambiente de simulación, se debe seleccionar en la pestaña "File > new Project". Al seleccionar, se abre una ventana como la Figura 29, donde se permite elegir características como el nombre para el proyecto y es posible determinar el tipo de modelo de transmisiones en RF y el retardo de inicio de los nodos.

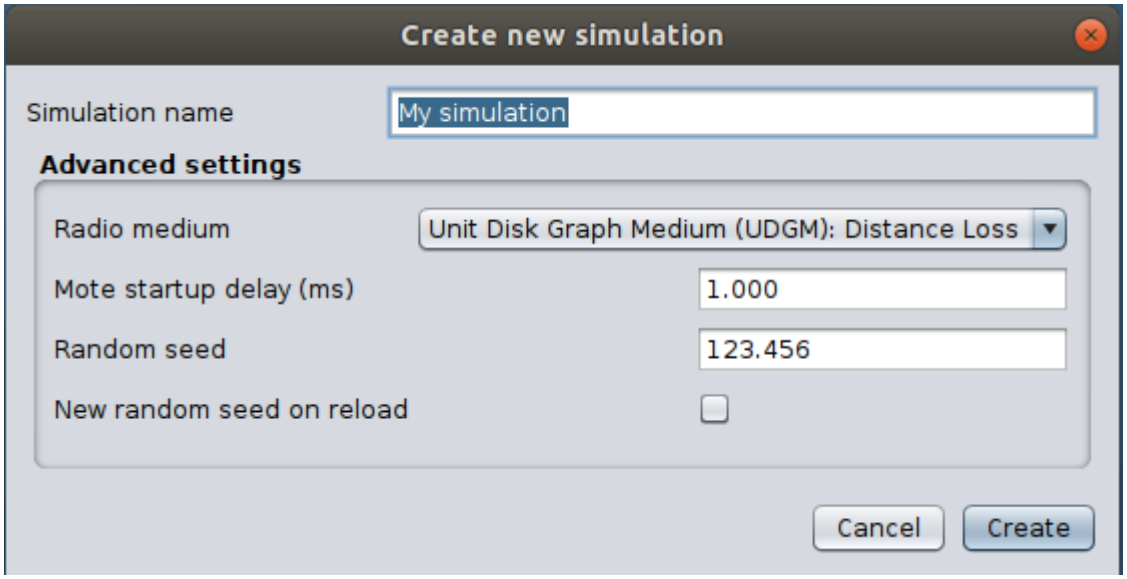


Figura 29. Ventana inicial del simulador Cooja de ajustes iniciales.

Fuente: Autoría

Realizadas las configuraciones iniciales, al pulsar "Create" automáticamente abre un nuevo escenario. Como muestra la Figura 30, se observa que el simulador contiene varios paneles para controlar, administrar y visualizar la simulación y que se los describe a continuación:

a) **Network:** es el panel que permite ubicar los nodos de red en la configuración deseada y que también permite visualizar y ajustar sus rangos de transmisión.

b) **Simulation control:** permite iniciar, pausar, ejecutar por paso o reiniciar todo el entorno. También es posible incrementar o reducir la velocidad de la simulación.

c) **Mote output:** muestra en tiempo real los mensajes que emiten los nodos en las sentencias “Log” de sus programas.

d) **Timeline:** indica los eventos ocurridos cronológicamente sobre cada uno de los nodos.

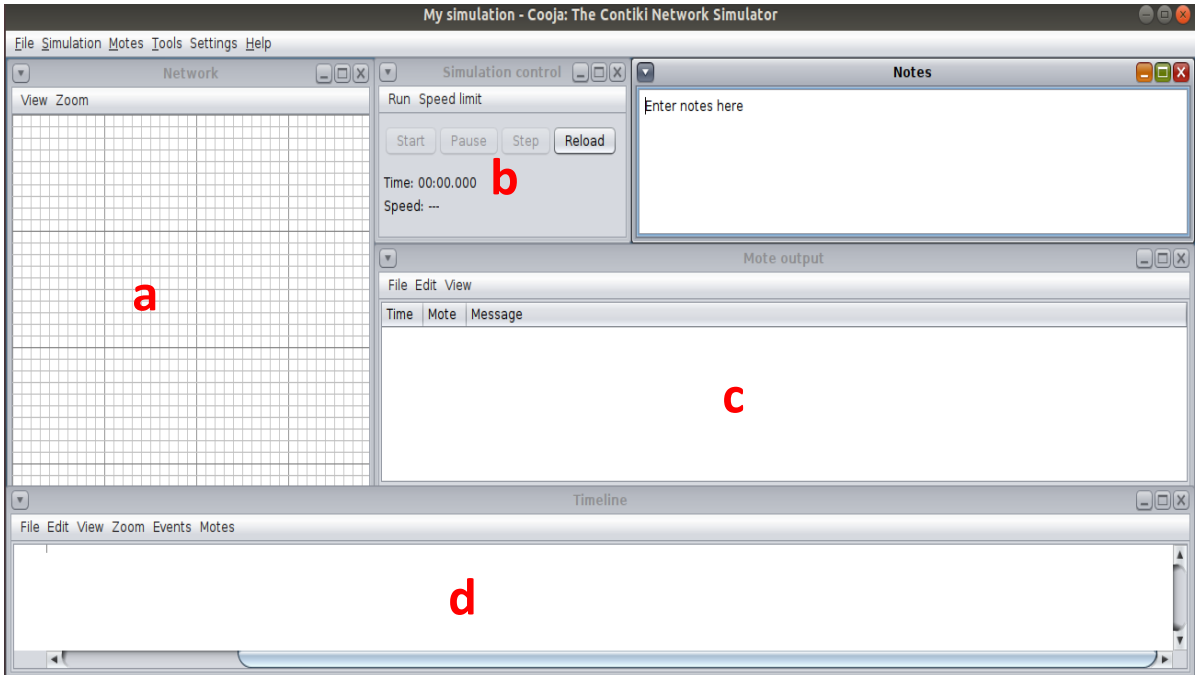


Figura 30. Ventana de paneles de control de la simulación.

Fuente: Autoría

A continuación, se inserta los nodos que van a intervenir en la simulación. Para este proyecto (como fue mencionado anteriormente), los programas de los elementos de red deben ser cargados sobre la plataforma previamente definida Wismote.

Para cargar un nuevo programa a un nodo, primero se debe seleccionar la plataforma sobre la cual va a funcionar. En la pestaña “Motes” se selecciona “Add motes > Create new type mote > Wismote”. Una vez seleccionada la plataforma, brevemente se abre una ventana como muestra la Figura 31, donde se puede nombrar a la nueva mota y elegir el firmware de cada tipo de nodos. En este caso se crea dos tipos de nodos: uno que funciona como controlador

nombrado Controlador-SDN con su respectivo programa, y otro que serán los nodos de datos nombrado Nodo-SDN.

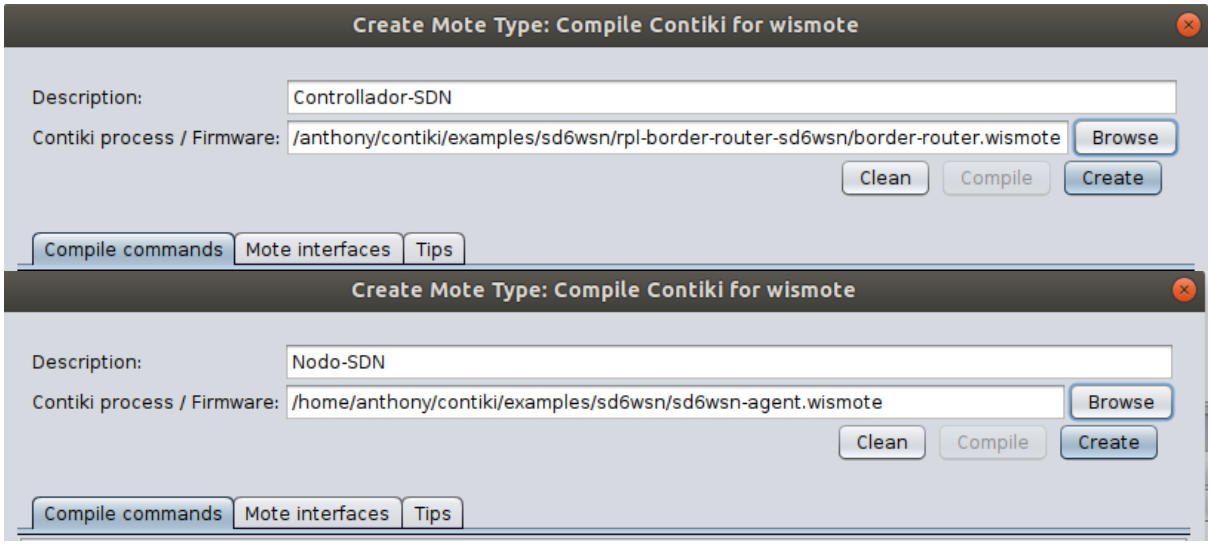


Figura 31. Creación de los nodos y controlador con sus respectivos programas.

Fuente: Autoría

Cargados los programas sobre la plataforma Wismote, finalmente aparece una pequeña ventana como la Figura 32, donde se debe especificar el número de nodos Controlador-SDN y Nodos-SDN se desea añadir a la red. En consecuencia, se crea 1 nodo controlador y 8 nodos emisores.

Una de las ventajas que provee el simulador, es que para agregar más elementos no es necesario volver a cargar el programa a la plataforma, sino que esta permite añadir varias veces el mismo tipo de nodo que haya sido cargado con anterioridad. Se usa esta característica con el fin de añadir los 8 Nodos-SDN.

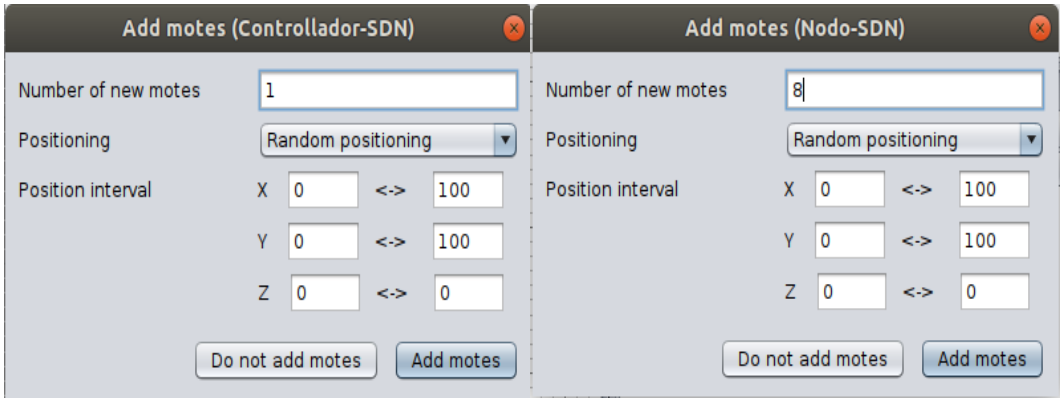


Figura 32. Especificación del número de motas a crear del controlador y de los nodos.

Fuente: Autoría

Finalizada la creación y adición de los nodos, en el panel Network se muestra todas las motas añadidas ubicadas de forma aleatoria las cuales deben ser organizadas para formar una red en malla.

El controlador, como se mencionó con anterioridad es un elemento igual que los nodos, que está basado en la solución SD6WSN y cuenta con las mismas características de hardware mínimas de la solución propuesta por (Miguel et al., 2018), y las cuales en el momento de la ejecución del sistema funcionan en conjunto. Al ser un elemento de red, el controlador y nodos tiene una capacidad máxima para soportar a 25 vecinos y 30 rutas. A continuación, se enlistan el resto de las características del controlador.

- Funcionamiento de enrutador de borde
- Capacidad máxima de 25 nodos de gestión

4.2.3 Diseño de topología

Para el despliegue y topología de la red a utilizar para la implementación de un canal de comunicación, este debe constar (como ya se ha mencionado antes) por un solo controlador central, es decir, un nodo debe ser implementado con la función de controlador de red y constituir de al menos dos nodos que se comuniquen y un máximo permitido de 20 nodos de

acuerdo a lo mencionado por (Baddeley et al., 2018), que afirma que mientras más nodos tiene la red más errores de entrega de mensajes presenta.

En resumen, lo adecuado es contar solo con 8 nodos de red que recolecten/generen los datos y 1 nodo que cumpla la función de controlador. Obteniendo un total de 9 elementos de comunicación. Para llevar a cabo este entorno la topología de red utilizada es una red en malla que va a ayudar a transmitir los datos de nodo a nodo hasta llegar al controlador. Este diseño de la red se puede observar visualmente en la Figura 33, en la cual se puede apreciar la distribución y conexión de cada uno de los elementos.

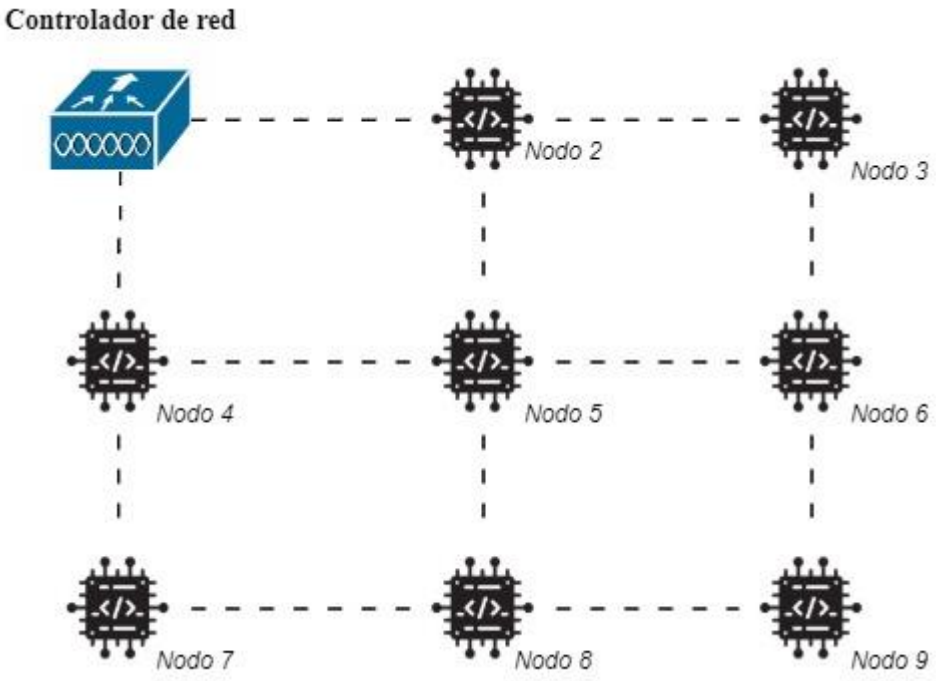


Figura 33. Topología de red en malla de 3x3 para la implementación del cifrado.

Fuente: Autoría

En lo referido al direccionamiento IPv6 dentro de la red, el sistema operativo Contiki incorpora por defecto el direccionamiento en cada uno de los nodos. Para asignar una dirección Contiki usa el stack ligero TCP/IP con el prefijo FE80::1/64 y basado en el número de nodo que representa en la red se asigna el último dígito hexadecimal, logrando que cada nodo

adquiera su propia dirección IPv6 para la comunicación. Esta dirección es la misma para todos los nodos cambiando el ultimo digito el cual es asignado en base al número de nodo que representa en la red como representa la Tabla 25.

Tabla 25

Direccionamiento IPv6 de los nodos en la red

Dispositivo	**IPv6 Local**	**IPv6 Global**
Controlador	fe80::200:0:0:1	fd00::200:0:0:1
Nodo #2	fe80::200:0:0:2	fd00::200:0:0:2
Nodo #3	fe80::200:0:0:3	fd00::200:0:0:3
Nodo #4	fe80::200:0:0:4	fd00::200:0:0:4
Nodo #5	fe80::200:0:0:5	fd00::200:0:0:5
Nodo #6	fe80::200:0:0:6	fd00::200:0:0:6
Nodo #7	fe80::200:0:0:7	fd00::200:0:0:7
Nodo #8	fe80::200:0:0:8	fd00::200:0:0:8
Nodo #9	fe80::200:0:0:9	fd00::200:0:0:9

Fuente: Autoría.

Con base en lo anteriormente mencionado, se procede a reubicar los nodos para formar una red malla dentro del simulador Cooja, como muestra la Figura 34, donde el nodo N°1 es el controlador que se muestra con un color diferente a los demás nodos, con el objeto de indicar que contiene un programa diferente al resto.

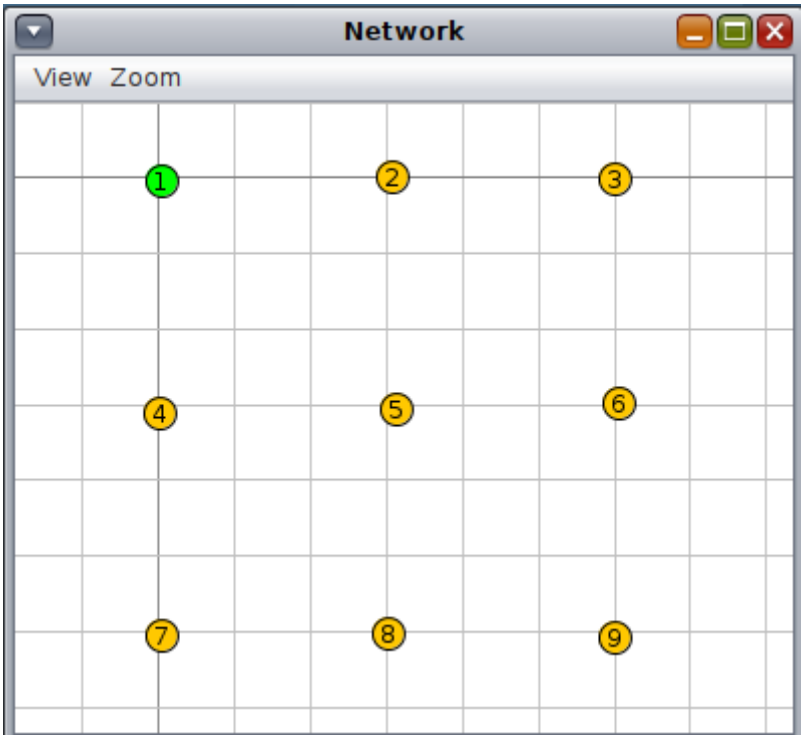


Figura 34. Red malla de nodos desplegada en el simulador Cooja de configuración 3x3.

Fuente: Autoría

4.2.4 Implementación del sistema

Para que el sistema se encuentre preparado para su ejecución, es necesario iniciar todos los procesos subyacentes necesarios para que el ambiente de pruebas funcione adecuadamente.

Para que el nodo N°1 cumpla su función de controlador, es necesario conectarlo mediante una comunicación serial con la maquina host Linux mediante la función tunslip6 (incorporado en el simulador Cooja). Con esta configuración la maquina host tiene la capacidad de insertar las instrucciones y el enrutador de borde las interpreta y ejecuta dentro de la red. Al crear una nueva conexión y una vez iniciada la simulación, esta función asigna un nueva dirección IPv6 al nodo N°1 que funciona como enrutador de borde y como controlador para realizar una comunicación entre los nodos de la red y una red externa (Miguel et al., 2018).

Por consiguiente, para habilitar esta funcionalidad se procede abrir una nueva ventana del terminal y como muestra la Figura 35, se accede a la carpeta “contiki/examples/sd6wsn” donde

se inserta el comando “sudo ./inicia-router-loop.sh” para ejecutar la funcionalidad de tunslip. Al habilitar la comunicación serial, el host entra en modo “escucha” a la espera de la conexión mediante la dirección de localhost y el puerto 60001 procedente del entorno de simulación.

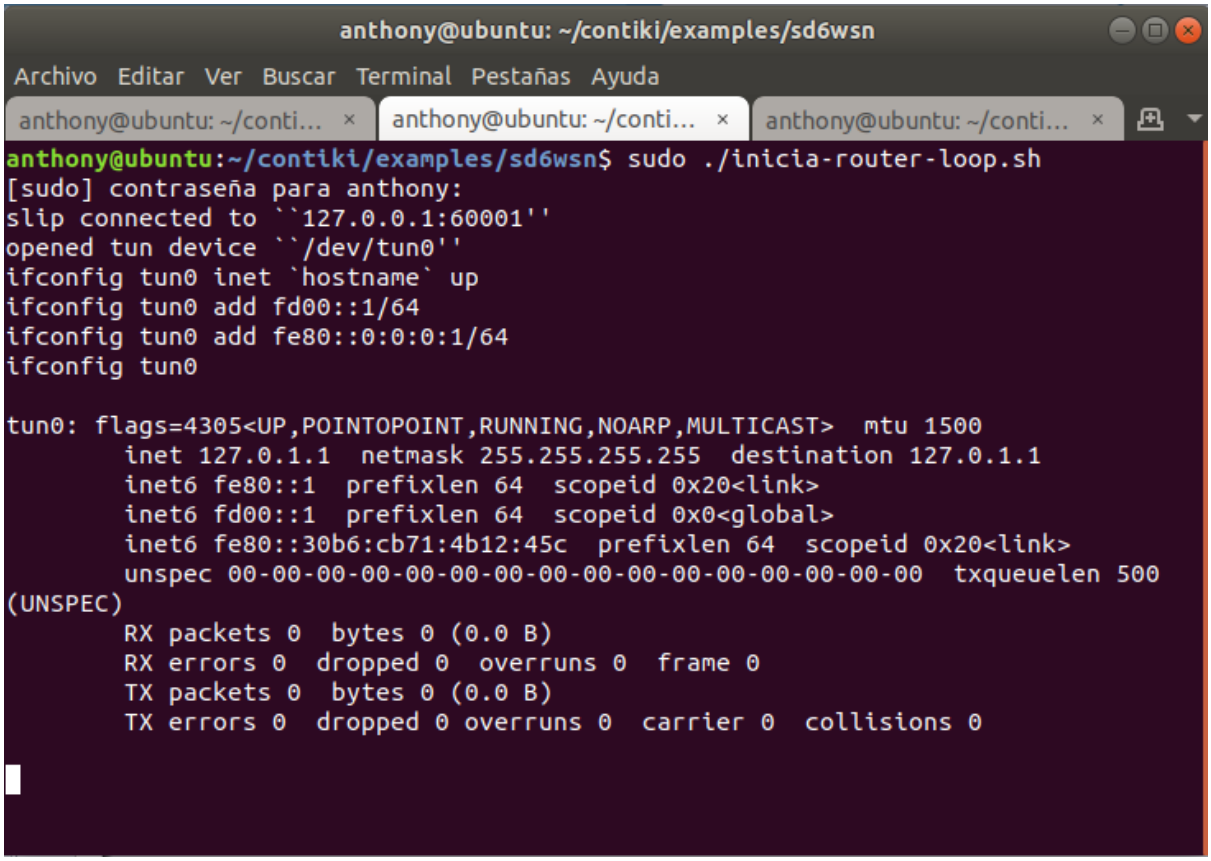


Figura 35. Inicio de la función tunslip6 para la conectividad entre la red simulada y la maquina host.

Fuente: Autoría

Luego de insertar el comando en el terminal e ingresar la contraseña sudo, habrá iniciado el programa tunslip6.

Para realizar la conexión completa del controlador, en el ambiente de simulación se procede a configurar el nodo N°1 para conectar con el host y que este pueda recibir las instrucciones y ejecutarlas. En la ventana Network se selecciona el nodo N°1, se despliega las opciones que contiene (Figura 36) y se selecciona la pestaña “Mote tools for Wismote 1 > Serial Socket (SERVER)”.

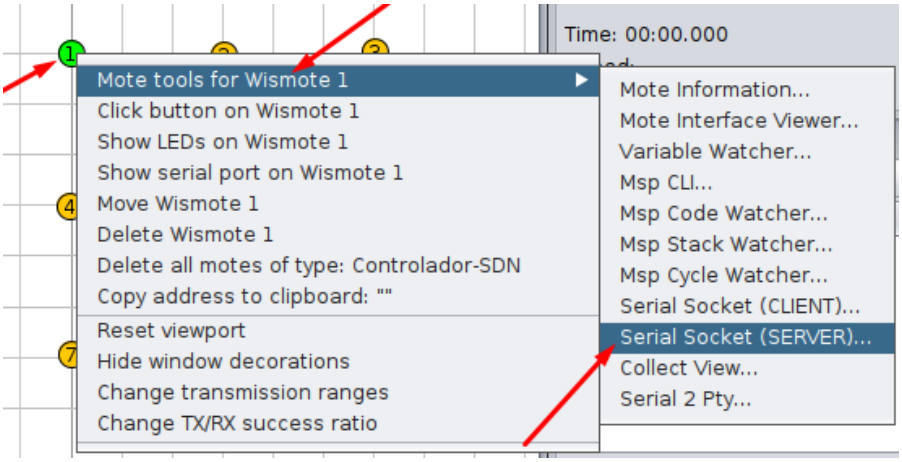


Figura 36. Conexión entre la maquina Linux y el nodo N°1.

Fuente: Autoría

Al seleccionar la opción de conexión serial, aparece una pequeña ventana que permite indicar el puerto por el cual se desea realizar la conexión, por defecto trae insertado el puerto 60001. Con la configuración por defecto, se presiona el botón “Start” y el nodo inicia el proceso de conexión con el host. Como previamente el host permanecía a la escucha por el puerto designado, después de unos segundos, en la ventana de conexión del nodo N°1 muestra un estado de conectado con el host.

Una vez se tiene todas las funciones de la red iniciadas, el sistema se encuentra listo para ejecutar el ambiente de pruebas, registrar la actividad e introducir las instrucciones de conexión en la red desde el controlador.

4.2.5 Ejecución del sistema

Con todos los elementos incorporados en el ambiente de simulación, se procede a iniciar la ejecución del sistema presionando el botón “Start” del panel “Simulation control”.

Como se observa en la Figura 37, al empezar a ejecutarse la simulación, los nodos inician sus procesos de direccionamiento IPv6 y ejecutan sus respectivos programas. Lo primero que realizan es enviar información de su estado a sus vecinos más próximos, tratando de alcanzar

el nodo principal (controlador). Pero hasta que el controlador no instale las instrucciones en las tablas de flujo, los nodos no envían los datos generados.

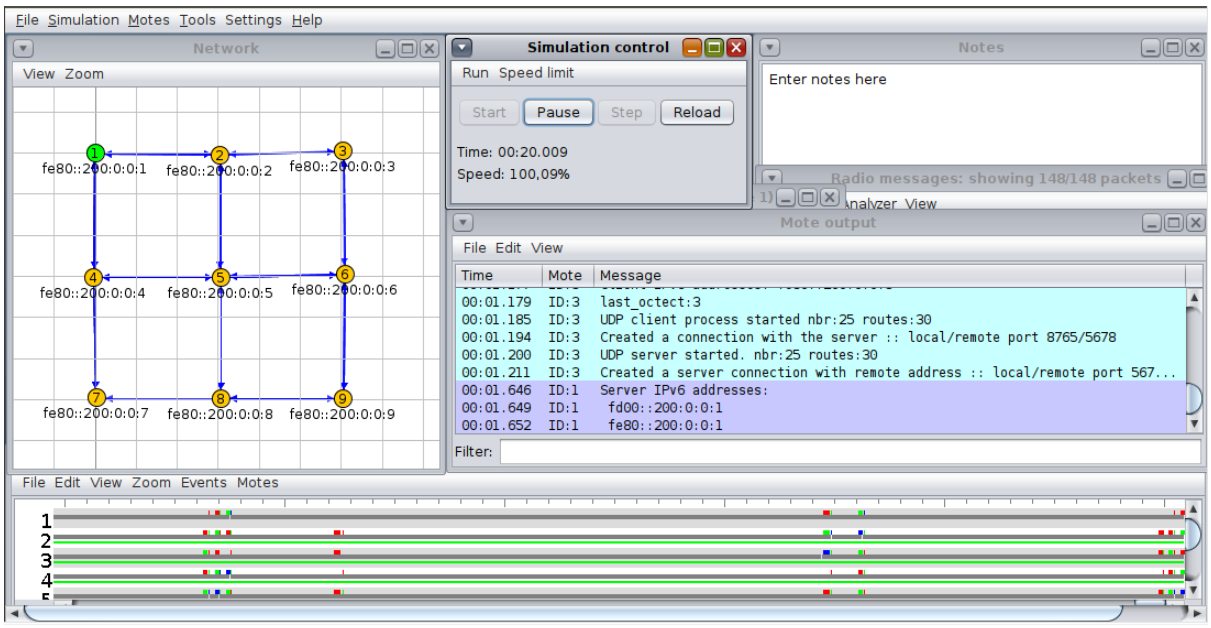


Figura 37. Inicio de simulación de la red SDWSN antes de introducir instrucciones de enrutamiento.

Fuente: Autoría

Al iniciar la simulación de la red, desde el controlador se debe instalar las respectivas reglas de flujo en cada nodo para llegar al nodo raiz (controlador).

4.2.5.1 Instalación de Reglas de Flujo

Para la instalación de reglas, se procede a abrir un nuevo terminal, se ingresa al directorio *~/contiki/examples/sd6wsn/scripts* y se ejecuta las instrucciones del archivo sd6wsn-controller-v52.js mediante el comando *node sd6wsn-controller-v52.js* (Figura 38).

Al ejecutar el archivo, el controlador recopila la información, muestra cuantos nodos hay en la red y cuál es el vecino más próximo de cada nodo. Con esta información el controlador construye una vista general de la topología de red, lo que permite instalar las reglas de flujo en cada nodo dependiendo de su ubicación. Es necesario considerar que las rutas a instalar son para que los nodos alcancen al controlador y viceversa.

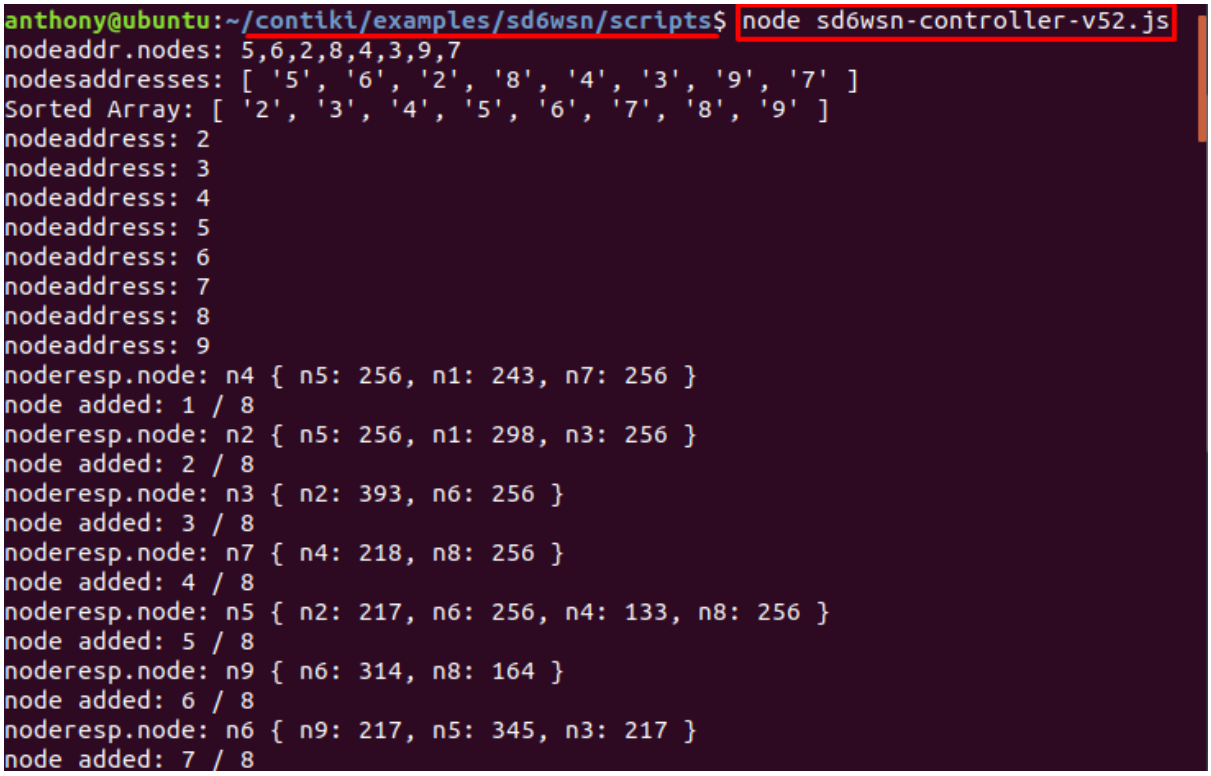


Figura 38. Salida del terminal de la recolección de información del controlador.

Fuente: Autoría.

La ejecución de instrucciones del controlador se las ha dividido en tres etapas: recolección de información, instalación de flujo en la ruta de los nodos para alcanzar al controlador e instalación de reglas de flujo en la ruta del controlador para alcanzar a los nodos.

Recolección de Información

En este primer proceso se realiza la recopilación y presentación de la información de todos los nodos presentes en la red para armar una vista general. A continuación, se enlista la información que presenta este proceso:

- Muestra el número de nodos presentes en la red
- Obtiene el valor del último campo hexadecimal de la dirección IPv6 de cada nodo
- Ordena los nodos en orden ascendente
- Muestra los vecinos más próximos a cada nodo

En la Figura 39 se observa este primero proceso anteriormente descrito que realiza el controlador para presentar la información y que sigue un orden especifico.

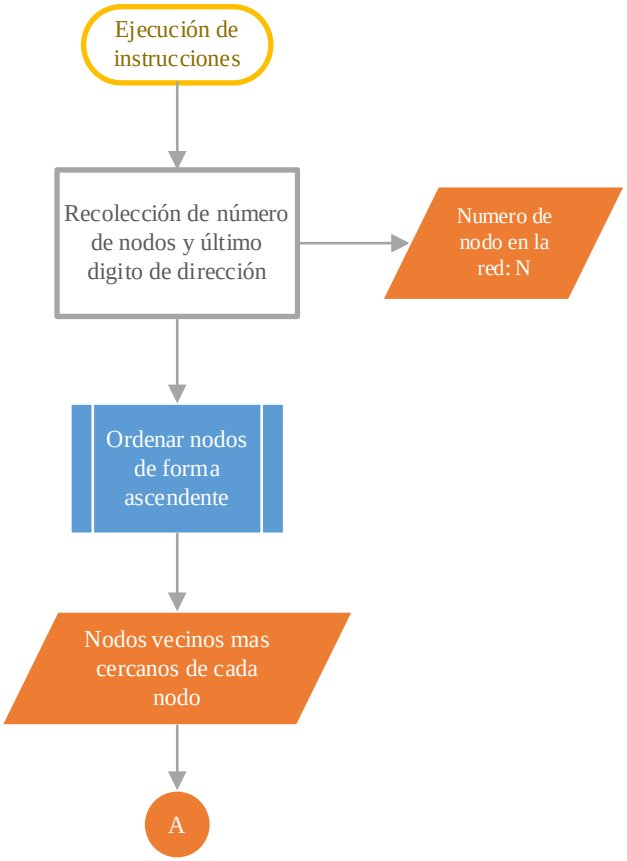


Figura 39. Proceso de recolección de datos del controlador.

Fuente: Autoría.

Dentro de esta función se obtiene las variables con las que se opera para los siguientes procesos. A continuación, se describe las variables que se obtiene:

- N: Numero de nodos presentes en la red
- Next Hop: Los nodos más cercanos a cada uno de los nodos en la red.

Cuando se obtiene estos valores, se procede al siguiente proceso nombrado como proceso "A".

Instalación de flujo en los nodos hacia el controlador

Este proceso realiza la instalación de los respectivos flujos dentro de los nodos para alcanzar al controlador. Cada ruta generada para llegar al destino desde cada nodo se diferencia por la identidad de flujo conocida como “Flow id”. Todos los nodos que se encuentren en la ruta entre un nodo origen y su destino instalan el mismo flujo y parámetros que el nodo origen solo cambiando la IP de su siguiente salto. Este proceso se repite varias veces hasta que todos los nodos en la red tengan la ruta para alcanzar al controlador y se puede observar de forma gráfica en la Figura 40.

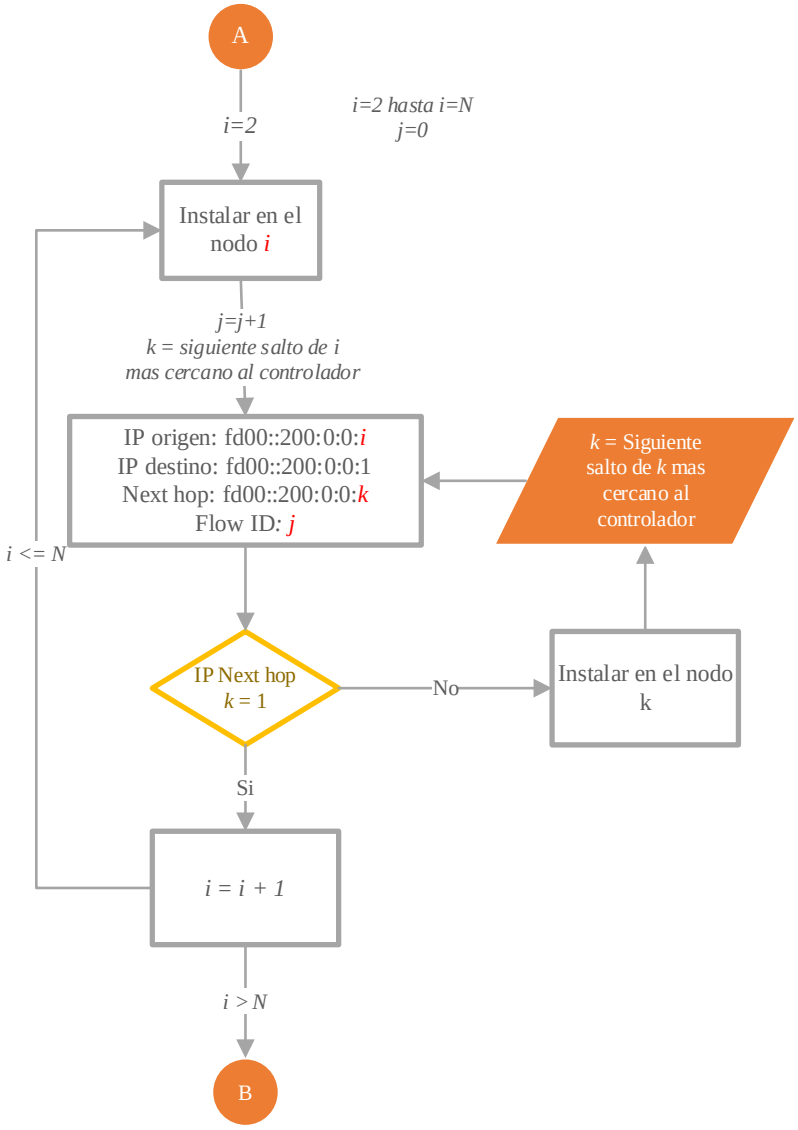


Figura 40. Proceso de instalación de flujos de los nodos hacia el controlador.

Fuente: Autoría.

Como se puede observar en la Figura 40, se ha creado un diagrama genérico del funcionamiento de las reglas de flujo en los nodos y se presentan variables que pueden cambiar dependiendo de la ubicación y número de nodos. Para el caso específico del presente proyecto las variables iniciales son las siguientes:

- $N = 9$
- $i = 2$
- $j=0$
- k= siguiente salto de i más cercano al controlador.

Nota: *i* y *k* representa el último valor del campo hexadecimal de la dirección IPv6, así como también el número de nodo que representa en la red.

Al terminar de instalar los flujos en el último nodo de la red (9) se continua al siguiente proceso llamado “B”.

Instalación de flujo del controlador hacia los nodos

Este proceso es inverso al anterior, es decir que ahora se genera las rutas para que el controlador pueda alcanzar a los nodos. El proceso de instalación es similar cambiando las IP’s de origen y destino, así como los siguientes saltos de cada uno. El contador del Flow ID no se reinicia y continua con la numeración del proceso anterior. En la Figura 41 se puede observar este proceso de forma gráfica para su entendimiento.

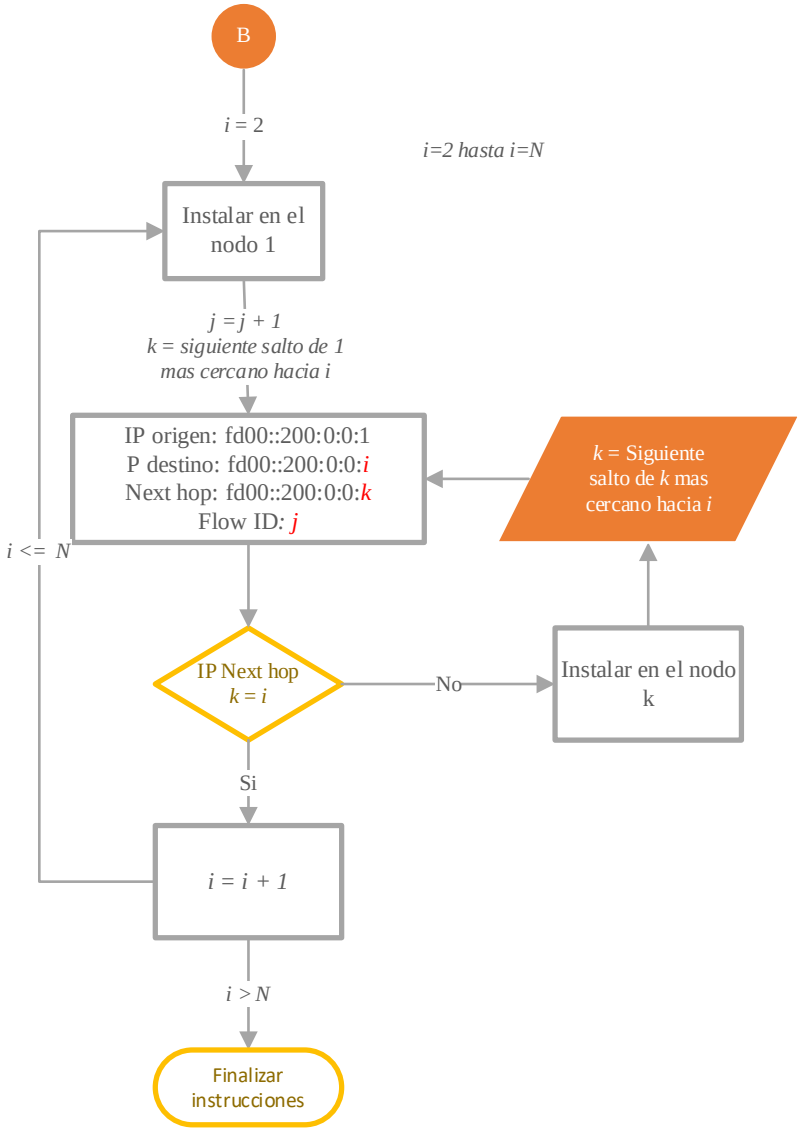


Figura 41. Proceso de instalación de flujos del controlador hacia los nodos.

Fuente: Autoría.

En la Figura 41, se observa que se usan las mismas variables, las cuales se modifican o mantienen su valor del proceso anterior. A continuación, las variables iniciales para este proceso son las siguientes:

- $N = 9$
- i = reinicia su valor a 2
- k= siguiente salto de 1 más cercano a i
- j= continua el valor del proceso “A”.

El proceso de instalación de reglas de flujo también es visible desde la ventana del simulador que muestra el establecimiento de las reglas en cada nodo de la red como muestra la Figura 42.

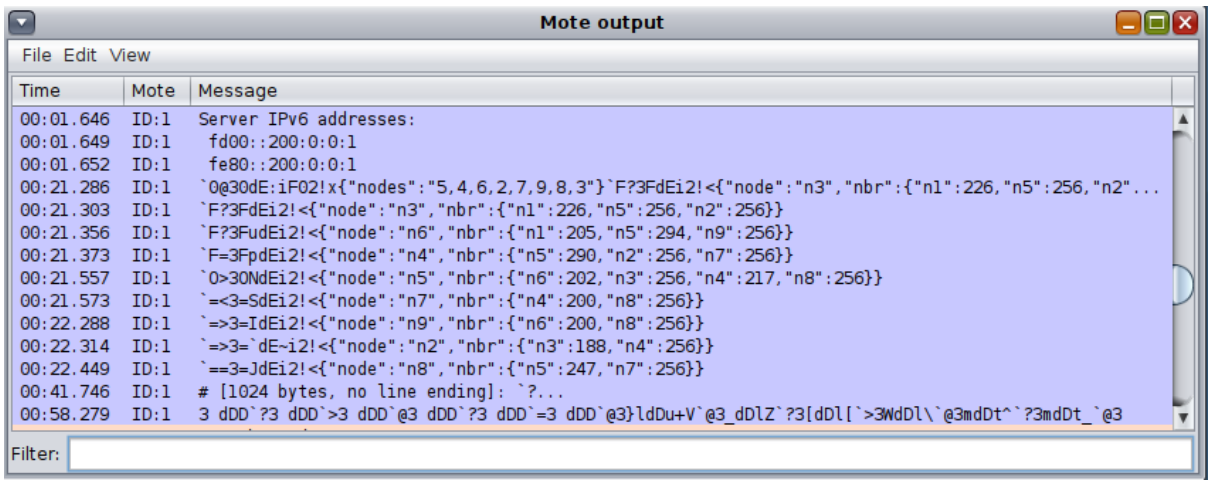


Figura 42. Instalación de reglas de flujo a los nodos observado desde el panel de visualización.

Fuente: Autoría

Cuando los nodos empiecen el envío de “datos” al controlador, se podrá visualizar cada mensaje enviado o recibido en la ventana “Mote output”.

4.2.5.2 Conectividad del controlador

Instaladas las reglas de flujo en cada nodo, inicia los procesos de conectividad. En consecuencia, para garantizar que existe enlace dentro de la red con todos los nodos es necesario realizar ensayos de la red.

Para comprobar la conectividad, se hace uso de la función *ping6* desde el controlador (recordando que la PC tiene una conexión directa con el nodo controlador) hacia el destino que se quiere verificar. Para esta comprobación primero se procede con los elementos que se encuentran más cercanos al controlador, como lo son el nodo 2 y 4, como presenta la Figura 43.

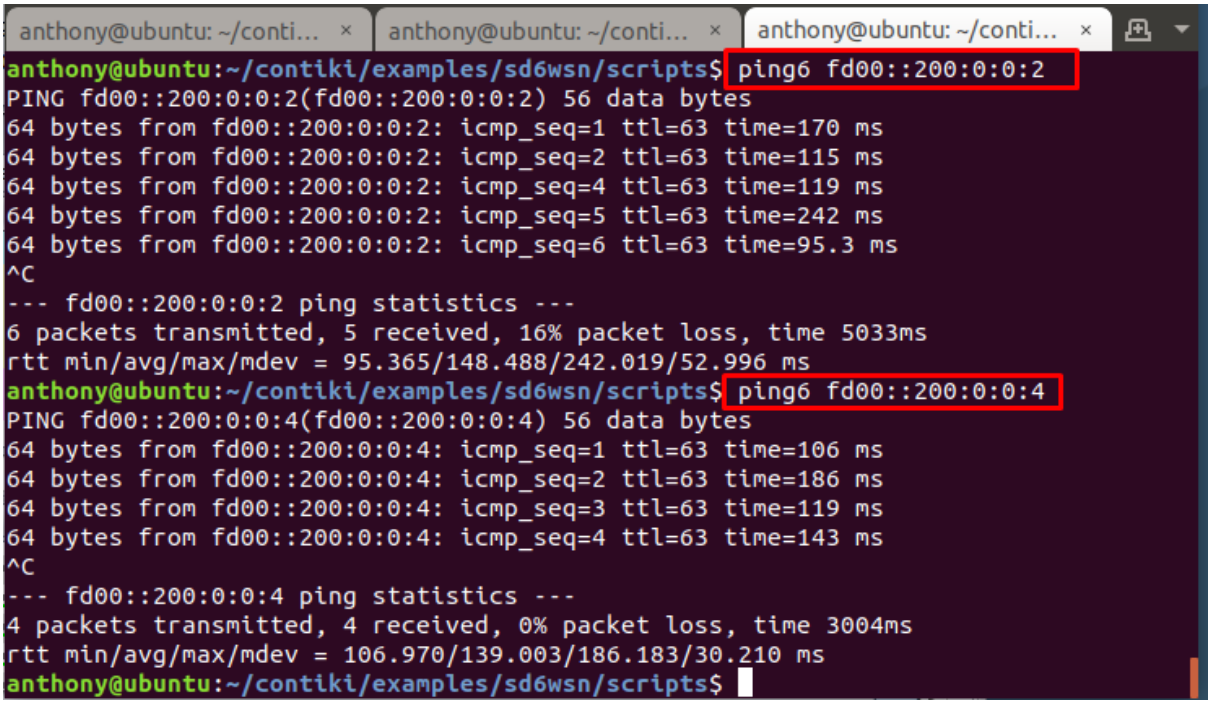


Figura 43. Comprobación mediante el comando ping la conectividad con los nodos 2 y 4.

Fuente: Autoría.

Se puede comprobar efectivamente que los nodos 2 y 4 responden a la prueba de conexión del comando ping. Continuando con la prueba, también se procede a comprobar la conectividad con los nodos más alejados de la red, como los son los nodos 8, 9 y 6. Para ello se sigue el procedimiento anterior cambiando las direcciones IPv6 correspondiente a esos nodos (Figura 44).

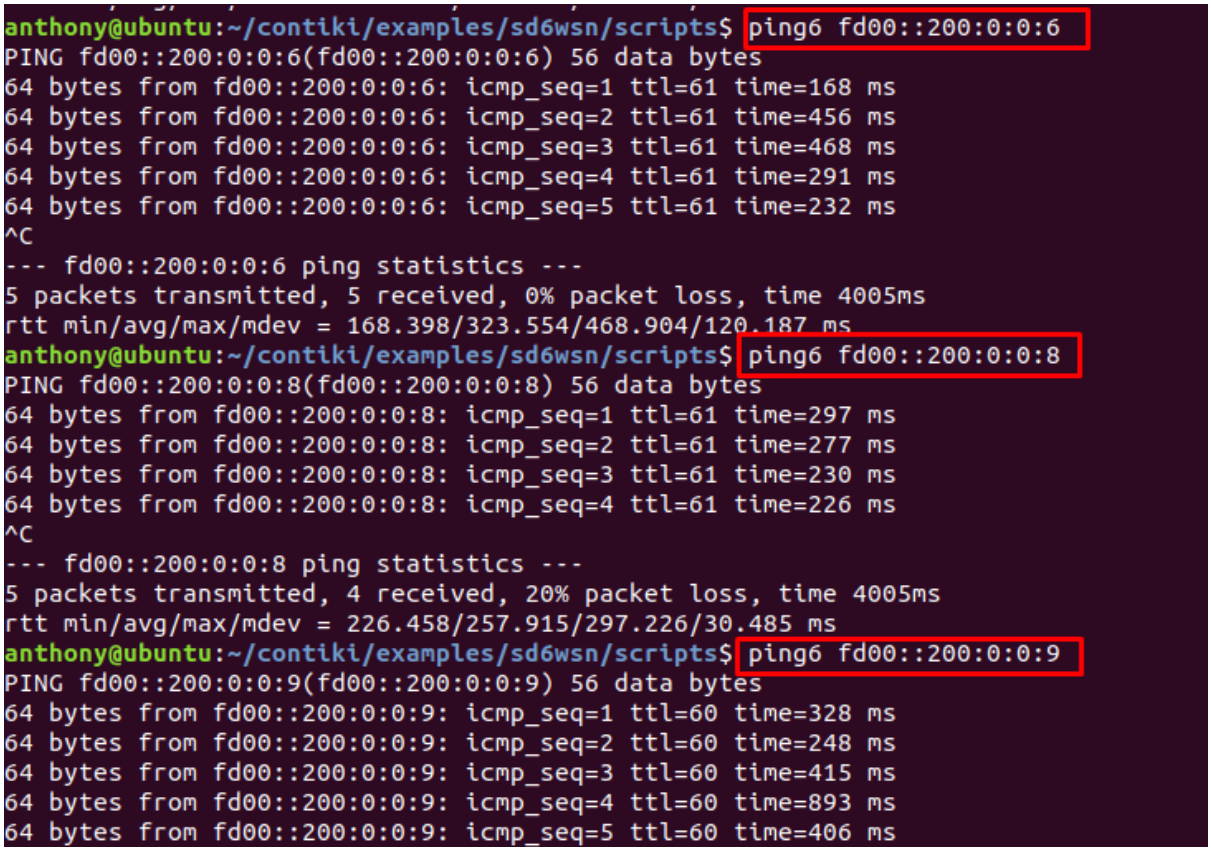


Figura 44. Comprobación la conectividad con los nodos 6, 8 y 9 mediante el comando ping.

Fuente: Autoría.

Como se puede apreciar, los nodos cercanos y lejanos responden correctamente a las peticiones que realiza el controlador, verificando que verdaderamente existe conexión entre los nodos de la red.

4.2.5.3 Conectividad de los nodos

Para verificar la conectividad entre los nodos se utiliza la función que tienen integrado por programación. Cada nodo tiene incorporado la función de ping hacia cada vecino más cercano, es decir, una vez que se instalan las reglas en las tablas de flujo los nodos inician automáticamente la prueba de conectividad.

Los nodos que se toma en cuenta para esta verificación son los nodos 8, 4 y 2 los cuales envían un paquete ping de “echo request” a cada uno de sus vecinos más cercanos y se puede visualizar en la Figura 45.

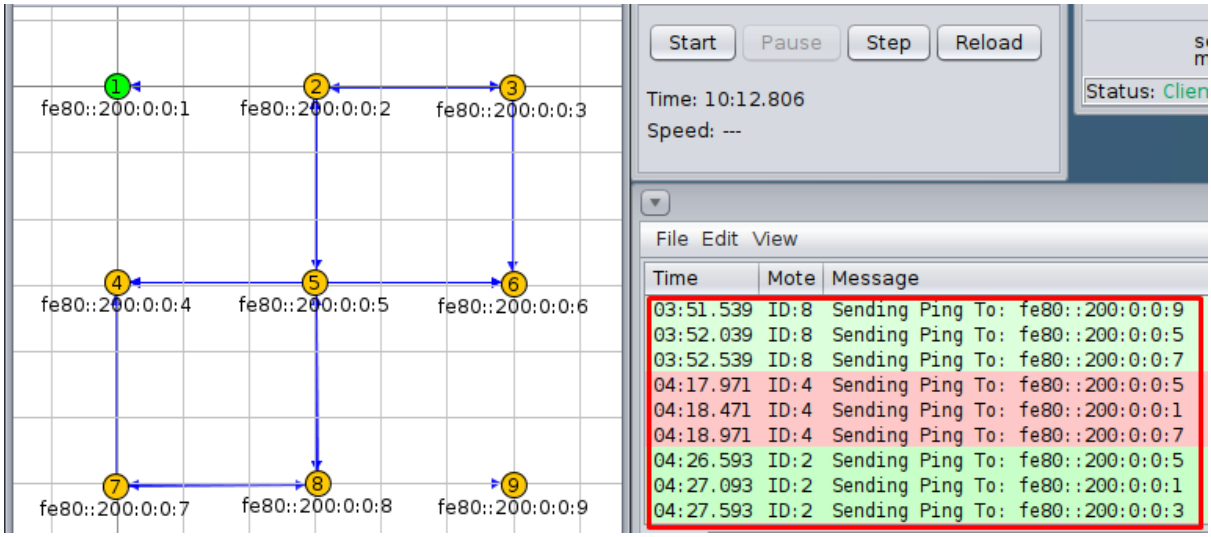


Figura 45. Solicitud de ping enviada a los vecinos más cercanos de cada nodo.

Fuente: Autoría.

En esta ocasión para comprobar que cada nodo está respondiendo a la solicitud ping se hace uso de la herramienta “Radio messages” que incluye el simulador y que sirve para observar todos los mensajes que son transmitidos en la red de la simulación.

Como muestra la Figura 46, se observa el contenido del paquete y se puede notar que es un “Echo Reply” enviando desde el nodo 9 en respuesta a la petición del nodo 8.

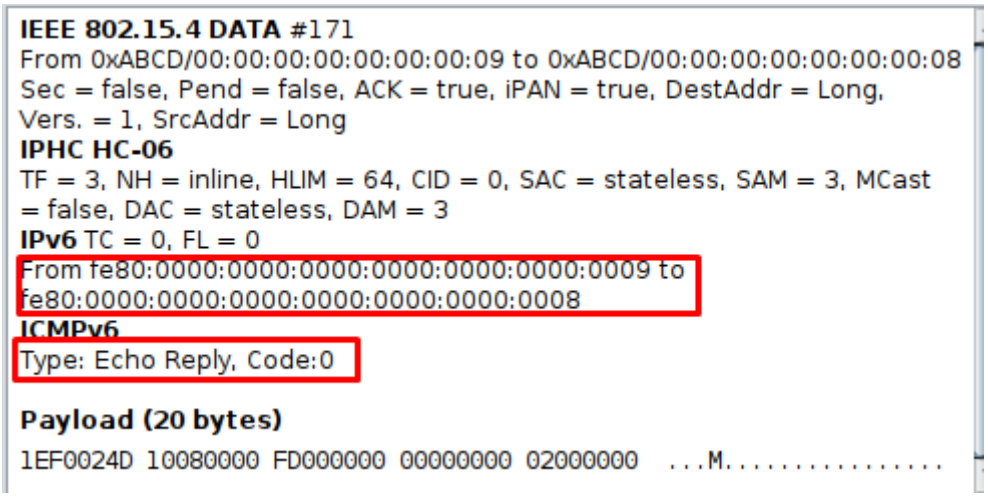


Figura 46. Contenido de la respuesta del nodo 9 al ping realizado por el nodo 8.

Fuente: Autoría

De igual forma en la Figura 47 se observa el contenido del paquete y se puede notar que es un “Echo Reply” enviando desde el nodo 5 en respuesta a la petición del nodo 4.

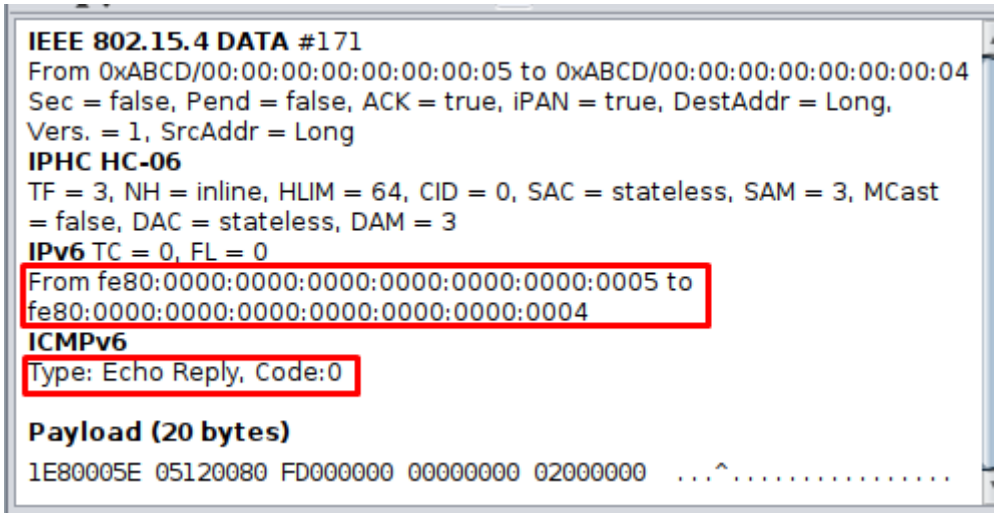


Figura 47. Contenido de la respuesta del nodo 5 al ping realizado por el nodo 4.

Fuente: Autoría.

Por último, en la Figura 48 se observa que el contenido del paquete es un “Echo Reply” enviando desde el nodo 3 en respuesta a la petición del nodo 2.

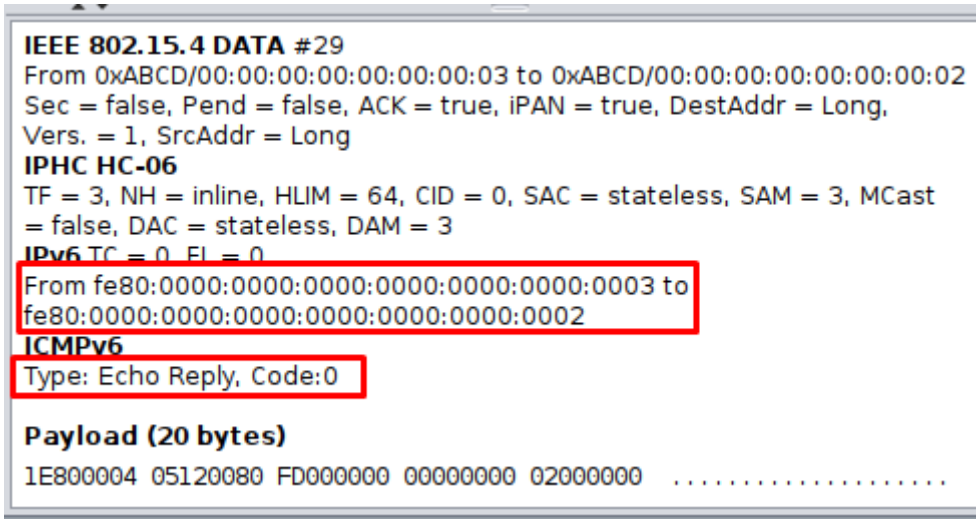


Figura 48. Contenido de la respuesta del nodo 3 al ping realizado por el nodo 2.

Fuente: Autoría

4.2.5.4 Ruta más corta

Cuando un nodo cuenta con varias rutas para llegar hacia el controlador, el protocolo RPL y la aplicación sd6wsn usan la métrica de ETX como el peso de los enlaces correspondientes eligiendo la ruta con el ETX acumulado más bajo. Los nodos para comprobar la calidad de los enlaces tienen un mecanismo llamado “RPL Probing”, el cual genera mensajes de control RPL dentro de ICMPv6 y transmitidos en modo unicast, logrando que se actualice los valores de ETX en cada enlace.

Como se puede observar en la Figura 49, al realizar una traza desde el controlador hacia el nodo más alejado (nodo 9) con dirección fd00::200:0:0:9, muestra que sus siguientes saltos son: primer salto es el nodo 2 (fd00::200:0:0:2), su segundo salto el nodo 5 (fd00::200:0:0:5), el tercer salto es el nodo 8 (fd00::200:0:0:8) y finalmente muestra que ha llega a su destino.

```
anthony@ubuntu:~/contiki/examples/sd6n-encry/scripts$ traceroute fd00::200:0:0:9
traceroute to fd00::200:0:0:9 (fd00::200:0:0:9), 30 hops max, 80 byte packets
 1  * * *
 2  * fd00::200:0:0:2 (fd00::200:0:0:2)  1055.890 ms  1314.898 ms
 3  fd00::200:0:0:5 (fd00::200:0:0:5)  2966.068 ms * *
 4  * * fd00::200:0:0:8 (fd00::200:0:0:8)  3167.768 ms
 5  fd00::200:0:0:9 (fd00::200:0:0:9)  4272.477 ms  2150.654 ms *
```

Figura 49. Traza de saltos para llegar al nodo 9 desde el controlador.

Fuente: Autoría.

Cuando se mueve los nodos que se encuentran de camino intermedio para llegar al nodo 9 y se los aleja, la métrica del etx es cambiada y registrada por la red (Figura 50), por ende, la ruta para el mismo nodo cambia de saltos creando una nueva que esté disponible para alcanzar al nodo raíz o controlador.

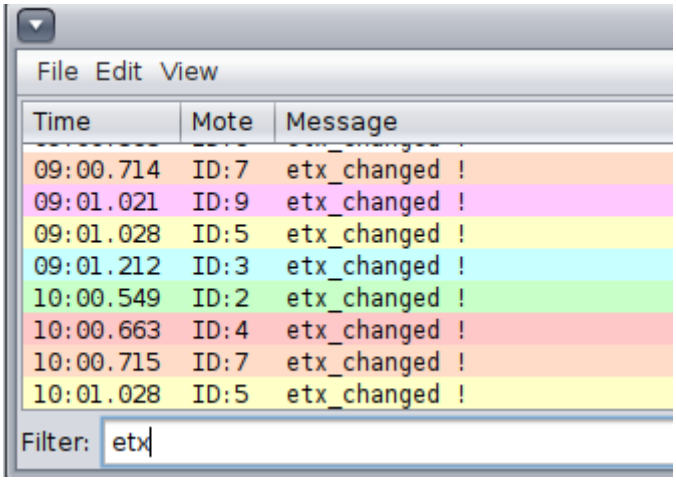


Figura 50. Registro de cambios en las rutas de la red.

Fuente: Autoría.

Al volver a realizar la traza desde el controlador hacia el mismo nodo 9, como se ve en la Figura 51, se observa que ahora sus primeros saltos cambian con respecto al anterior, siendo ahora los siguientes: su primer salto es el nodo 4 (fd00::200:0:0:4), su segundo salto es el nodo 7 (fd00::200:0:0:7), su tercer salto la dirección nodo 8 (fd00::200:0:0:8) y finalmente alcanzando a su destino, el nodo 9.

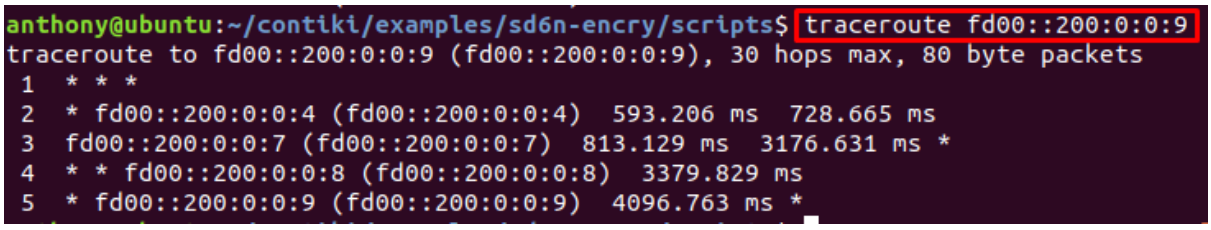


Figura 51. Traza de saltos para llegar al nodo 9 desde el controlador.

Fuente: Autoría.

Al realizar cambios en la red, las métricas de la calidad de los enlaces se van actualizando, permitiendo que en caso de añadir o remover algún dispositivo, todos estos puedan alcanzar su destino.

4.3 Implementación de encriptación Speck

Speck al ser un algoritmo de bloques ligeros, no debe representar una mayor sobrecarga para el dispositivo y también se debe considerar que se va a ejecutar sobre nodos que están adaptados a un sistema SDN. Al incluir nuevas funciones pertenecientes a SDN, ya cuentan con una carga adicional de procesamiento y almacenamiento. Por lo tanto implementar una capa adicional de seguridad ContikiSec como la propuesta por (Casado & Tsigas, 2009) no sería óptimo para el tipo de funcionamiento que se está buscando, debido a que generaría un mayor esfuerzo por parte del dispositivo al añadir una capa de seguridad la cual se debe adaptar para la encriptación y desencriptación.

Speck cuenta con variaciones para su implementación dependiendo del dispositivo en el cual será basado, como menciona (CryptoLUX, 2020) contempla varios dispositivos embebidos tales como: Atmel AVR ATmega128 de 8-bit, Texas Instruments MSP430F1611 de 16-bit y Arduino Due ARM Cortex-M3 de 32-bit. Cada uno de estos dispositivos tienen una adaptación a sus dispositivos en la cual los procesos son distintos. Como se mencionó anteriormente, tanto el controlador como los nodos emisores funcionan sobre la plataforma Wismote la cual está basada en MSP430 de TI. Las pruebas realizadas por CryptoLux (2020) sobre dispositivos MSP430 muestran que el algoritmo de encriptación Speck cuenta con el mejor desempeño en cuanto a tamaño de memoria sobre el dispositivo, siendo el de menor consumo.

Es por tal razón que el proceso de encriptación se va a incluir dentro de los mismos dispositivos, es decir, los mensajes serán encriptados en los nodos emisores empleándose como una función más para luego ser enviado al controlador que recibirá y desencriptará para presentar su contenido y verificar que el proceso funciona. Speck está trabajando sobre la capa de infraestructura de la arquitectura previamente mostrada.

Una vez que se ha decidido la configuración del algoritmo Speck, se procede a obtener las librerías de Speck con su respectivo código fuente para la implementación. Para la implementación del Speck es necesario tener en cuenta que existen diversas librerías y cada una con distintas variantes del algoritmo, por ello se debe seguir un proceso para la correcta ejecución del complemento criptográfico. La Figura 52 presenta un diagrama visual de procesos a seguir para la elección correcta de la librería y su posterior implementación.

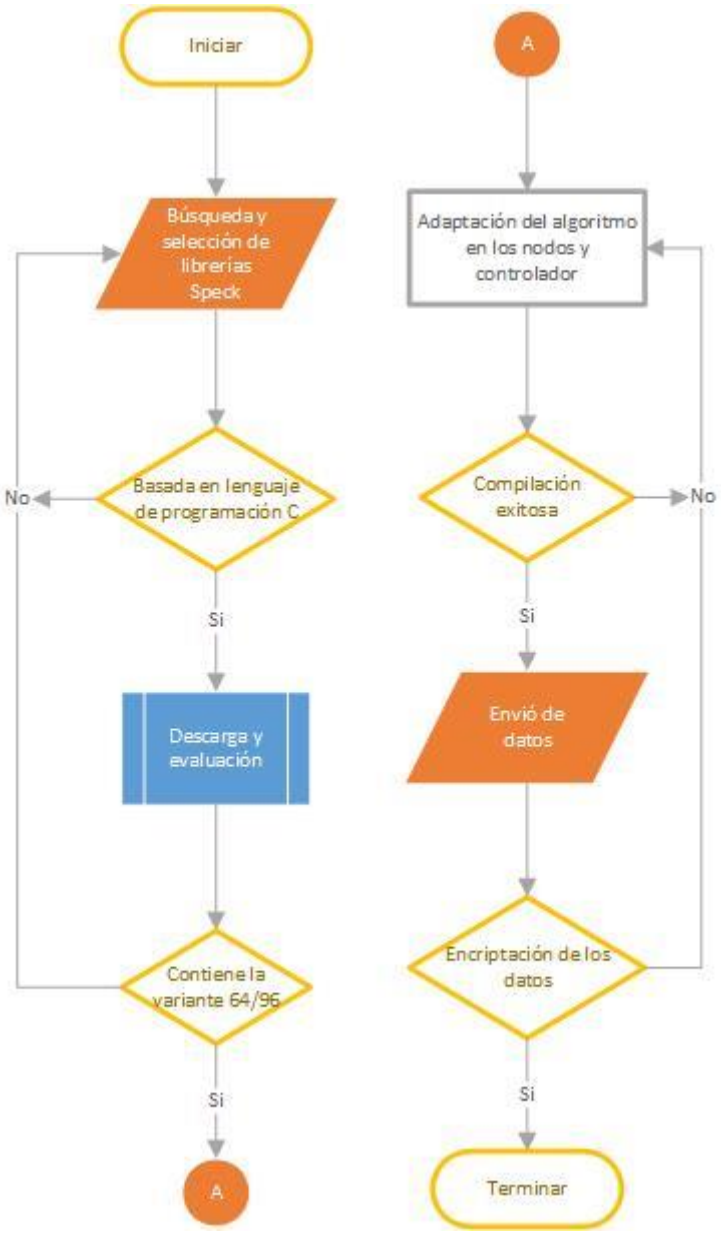


Figura 52. Diagrama de flujo del proceso de selección e implementación del algoritmo Speck.

Fuente: Autoría

4.3.1 Librerías del algoritmo Speck

Con el fin de implementar el algoritmo de encriptación se utiliza una librería que haya sido comprobada que funciona sobre Contiki, con el fin de que, al momento de la implementación no presente errores de compilación en el sistema. Por lo tanto, la librería presentada por (Semushin, 2020), es la opción que cumple con este requerimiento. Esta librería contiene el algoritmo Speck en sus variantes de 32/64, 64/96, 64/128, 128/128, 128/192, 128/256, y además también contiene otros algoritmos por las cuales se podría cambiar en caso de requerirlo.

Se debe notar que los tamaños de clave menores a 80 bytes no proveen un nivel alto de seguridad como lo menciona (Beaulieu et al., 2015). Lo adecuado es implementar una solución intermedia que ofrezca un buen nivel de seguridad pero que no genere una carga. Desde este punto de vista, Speck 64/96 y 48/96 son las opciones más optima a implementar, pero debido a que la configuración de 64/96 cuenta con un número mayor rondas es más adecuada para la encriptación.

Para añadir una nueva librería se debe realizar una copia en el directorio de trabajo y configurar los archivos que contiene. Como primer paso, primero se cambia el archivo llamado *"constants.h"* donde es necesario especificar el algoritmo a utilizar, su tamaño de bloque y tamaño de clave. Como muestra la Figura 53, en la línea 13 se establece el algoritmo Speck, en la línea 14 y 15 se define el tamaño de bloque y de clave respectivamente. La línea 10 define el juego de caracteres sobre los cuales se puede trabajar y la línea 11 el tamaño del mensaje que puede ser configurado en este archivo o en el programa de los elementos directamente.

Abrir constants.h ~/contiki/examples/sd6n-encry Guardar

```
1 #ifndef CONSTANTS_H_
2 #define CONSTANTS_H_
3 #include "algorithms.h"
4
5 #define PORT 1357
6 #define SEND_INTERVAL 10 //seconds
7 #define PACKET_SIZE 48 //bytes
8 //PACKET_SIZE*8 should be divisible by KEY_LENGTH
9
10 #define MESSAGE_CHARSET "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
11 #define MESSAGE_SIZE (10 * 1024)
12
13 #define ALGORITHM SPECK
14 #define BLOCK_LENGTH 64
15 #define KEY_LENGTH 96
16
17 #define KEY "c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0"
18 // it's 256bit, but only first KEY_LENGTH bits will be used
19 // TWOFISH and PRESENT_SPEED algorithms use precalculated keys located in "../
   cryptolibs/twofish" and "../cryptolibs/present_speed"
20
21 #endif /* CONSTANTS_H_ */
```

Figura 53. Archivo “constants.h” que contiene los parámetros para la configuración del algoritmo.

Fuente: Autoría.

Cuando el archivo está configurado correctamente, al momento de ejecutar el programa, “constants.h” llama a la librería Speck respectiva para incorporar la encriptación al proceso de los nodos.

Para que las librerías sean añadidas al momento de la compilación de los nodos, es preciso añadir su ubicación en el archivo Makefile para que se incorporen como parte del código. Dentro del archivo Makefile se especifica también el nombre del archivo que será compilado. Como muestra la Figura 54, en la primera línea se añade que el archivo Emisor será el programa al que se añada las librerías. También se añade la línea 6 “*PROJECT_SOURCEFILES += crypto.c*” para indicar al compilador que las librerías también forman parte del código.

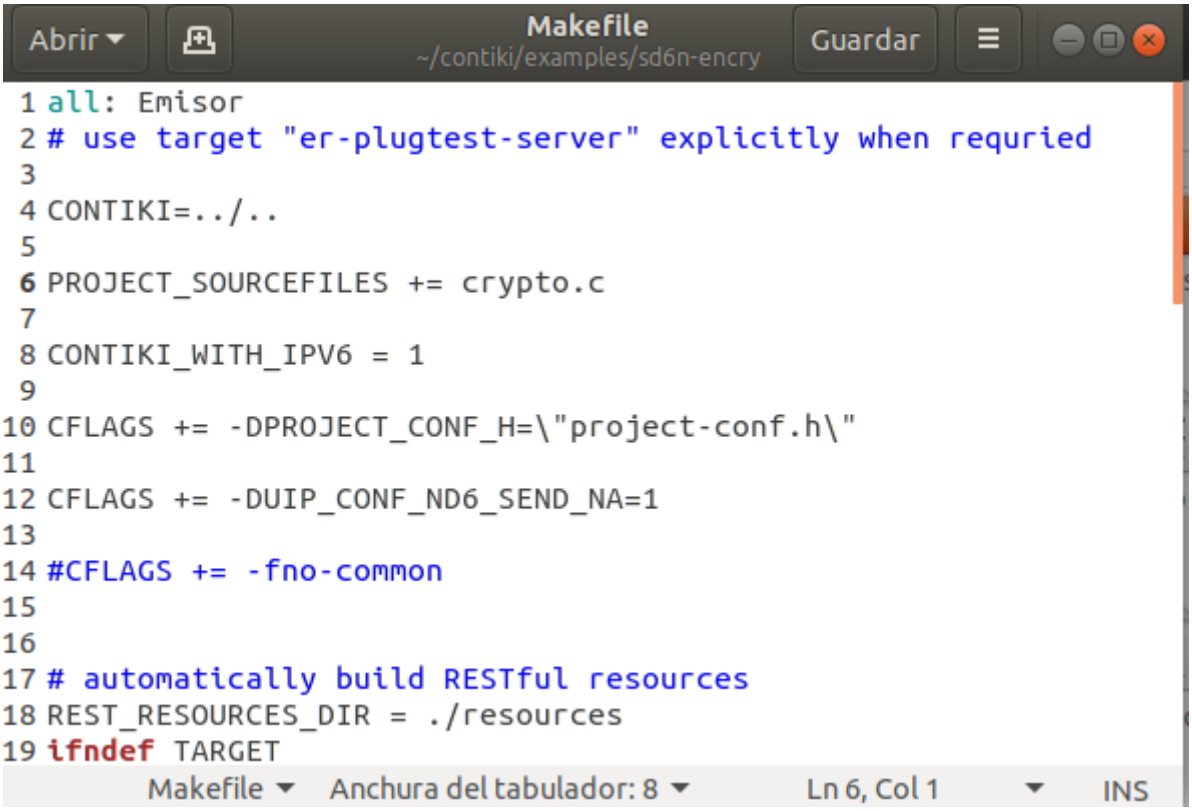


Figura 54. Adición de las librerías criptográficas en el archivo Makefile para la compilación.

Fuente: Autoría.

Con estos dos archivos configurados correctamente, hasta ahora se tiene configurado que se utiliza el algoritmo Speck en su variante 64/96 y que el compilador incorporara las librerías al momento de la compilación de los nodos.

Dado que los archivos Makefile son distintos para el nodo y el controlador, este proceso debe realizarse también en el archivo Makefile del controlador.

4.3.2 Adaptación del algoritmo Speck a los nodos

Después de tener configurados las librerías y el compilador, se procede a añadir la función de cifrado en los nodos que van a enviar los datos (nodos emisores). Para ello, se crea una copia del archivo *sd6wsn-agent.c* a la cual se realiza modificaciones. Esta nueva copia se nombra *Emisor.c* y se ubicara en el mismo directorio.

Dentro del archivo *Emisor.c* al inicio del programa, se debe añadir las librerías para llamar a sus procesos. Como muestra la Figura 55, al final del encabezado de librerías, se añade *#include "../cryptolibs/crypto.h"* para especificar donde se encuentran los algoritmos criptográficos e *#include "constants.h"* para incorporar los parámetros previamente modificados.

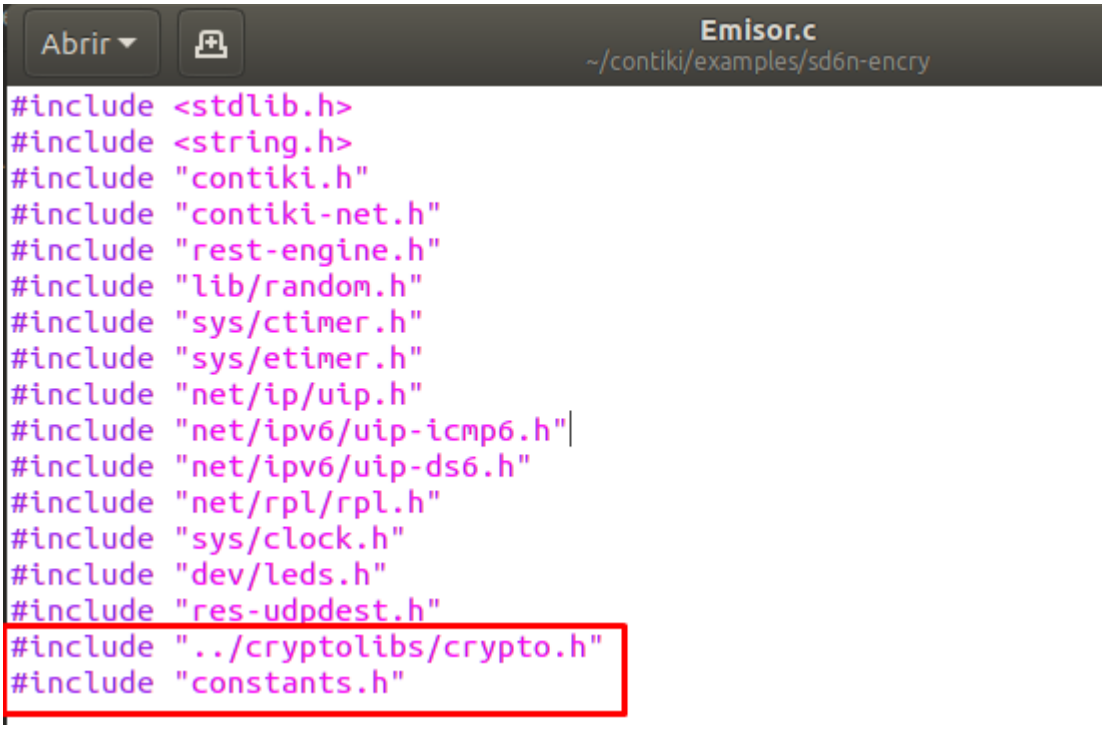


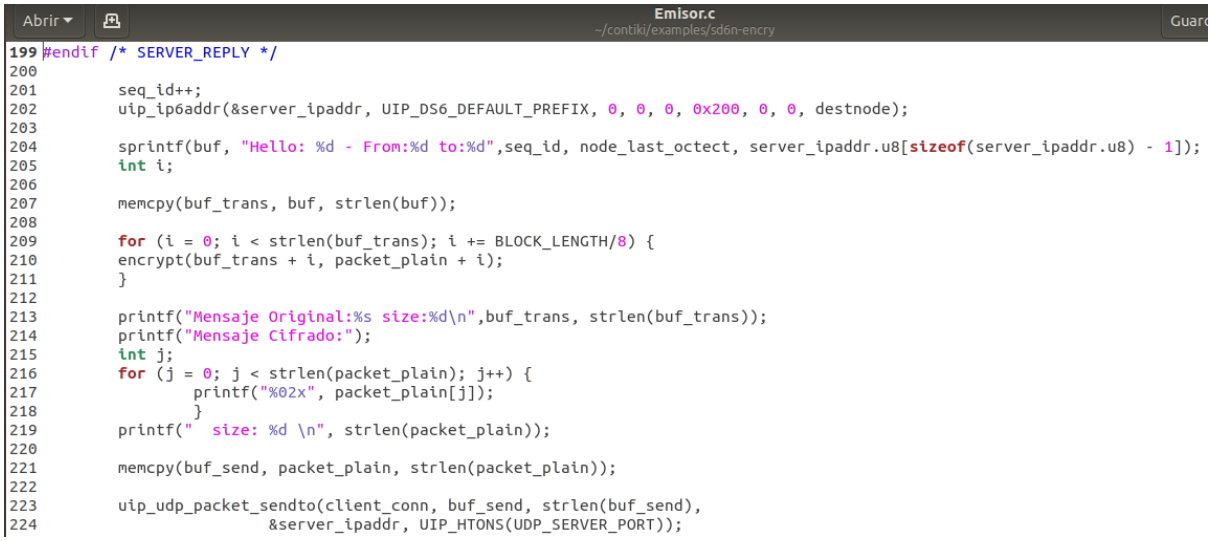
Figura 55. Adición de las librerías al programa de los nodos.

Fuente: Autoría.

Para implementar la encriptación y envió, es necesario realizarlo dentro de la función de envío de paquetes “send paquet” del programa del nodo.

En el archivo *Emisor.c* se define que el contenido a ser encriptado es de la variable “buf” que muestra la Figura 56 que contendrán los datos generados a ser enviados. Mediante la función *sprintf* se almacena los datos dentro de la variable buf, en este caso, se genera mensajes “Hello” de control para enviar al controlador. Cabe mencionar que en el lenguaje de programación C, para utilizar algunas funciones definidas se necesitan diferentes tipos de

variables para cada caso; es decir, para la encriptación los datos de las variables deben transformarse en tipo *unsigned char* (un tipo de variable que no asigna espacio de memoria fijo) y para enviar las variables deben ser tipo *char*.




```
199 #endif /* SERVER_REPLY */
200
201         seq_id++;
202         uip_ip6addr(&server_ipaddr, UIP_DS6_DEFAULT_PREFIX, 0, 0, 0, 0x200, 0, 0, destnode);
203
204         sprintf(buf, "Hello: %d - From:%d to:%d",seq_id, node_last_octect, server_ipaddr.u8[sizeof(server_ipaddr.u8) - 1]);
205         int i;
206
207         memcpy(buf_trans, buf, strlen(buf));
208
209         for (i = 0; i < strlen(buf_trans); i += BLOCK_LENGTH/8) {
210         encrypt(buf_trans + i, packet_plain + i);
211         }
212
213         printf("Mensaje Original:%s size:%d\n",buf_trans, strlen(buf_trans));
214         printf("Mensaje Cifrado:");
215         int j;
216         for (j = 0; j < strlen(packet_plain); j++) {
217                 printf("%02x", packet_plain[j]);
218                 }
219         printf("  size: %d \n", strlen(packet_plain));
220
221         memcpy(buf_send, packet_plain, strlen(packet_plain));
222
223         uip_udp_packet_sendto(client_conn, buf_send, strlen(buf_send),
224                         &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
```

Figura 56. Implementación de las funciones para la encriptación Speck.

Fuente: Autoría.

A continuación, se describe el contenido de la Figura 56:

- En la línea 204, se genera el contenido del mensaje a enviar
- En la línea 207, se copia el contenido de la variable *buf* que es tipo *char* y lo almacena en otra variable *buf_trans* de tipo unsigned char. Por ende, copiado el contenido ya se obtiene el tipo de variable adecuada para le encriptación.
- En la línea 210 se llama a la función *encrypt* de las librerías añadidas para encriptar los datos la cual necesita de dos argumentos; en el primer argumento se inserta la variable que contiene los datos que serán encriptados, en el segundo argumento se inserta la variable que contendrá la salida que devuelve la función cuando ha terminado la encriptación, es decir, los datos encriptados se almacenan en la variable del segundo argumento "*packet_plain*".

- La línea 213 y 214 imprimen el mensaje generado y el mensaje encriptado respectivamente.
- En la línea 221 el contenido de la variable *packet_plain* nuevamente se convierte de *unsigned char* a *char* mediante el proceso de copiado de variables realizado anteriormente dado que para enviar datos estos deben ser de tipo char (dato se llama *buf_send*).
- En la línea 223 se realiza el proceso de envío el cual está regido por las condiciones de conexión con el servidor. Se inserta la variable con su contenido previamente encriptado buf_send y se define su tamaño para indicar al programa que es el paquete para enviar hacia el controlador.

Para el proceso de encriptación se utiliza bucles *for* debido a que la encriptación se realiza dividiendo el contenido en varias partes de acuerdo con el tamaño de bloques. Para encriptar se fracciona el contenido en tamaños de 8 bytes, que es el resultado de la división de tamaño del bloque sobre 8.

Para la visualización de los paquetes salientes hacia el controlador, se cuenta con la función print, el cual mostrará el mensaje original que se envía y será visible en la ventana “Mote output” del simulador cooja.

4.3.3 Adaptación del algoritmo Speck al controlador

Para el desarrollo del controlador, se realiza una copia del programa *border-router.c* para realizar modificaciones. A este nuevo archivo se nombra *Receptor.c* e igualmente se guarda en el mismo directorio.

El controlador contiene un programa diferente al emisor, pero consta de un desarrollo similar para la recepción y desencriptación. Por tanto, en el archivo *Receptor.c* al inicio del programa, se debe añadir las mismas librerías para llamar a las funciones de encriptación.

Como muestra la Figura 57, al final del encabezado, se añade las líneas *#include "../cryptolibs/crypto.h"* e *#include "constants.h"* para especificar donde se encuentran los archivos que contienen los parámetros.

Receptor.c

```
39 #include "contiki.h"
40 #include "contiki-lib.h"
41 #include "contiki-net.h"
42 #include "net/ip/uip.h"
43 #include "net/ipv6/uip-ds6.h"
44 #include "net/rpl/rpl.h"
45 #include "net/rpl/rpl-private.h"
46 #if RPL_WITH_NON_STORING
47 #include "net/rpl/rpl-ns.h"
48 #endif /* RPL_WITH_NON_STORING */
49 #include "net/netstack.h"
50 #include "dev/button-sensor.h"
51 #include "dev/slip.h"
52 #include "examples/cryptolibs/crypto.h"
53 #include "../constants.h"
54
```

Figura 57. Adición de las librerías al programa del controlador.

Fuente: Autoría.

Para añadir la desencriptación, antes se debe recibir los datos para procesarlos. Para ello, el proceso debe ser realizado dentro de la función de recepción de paquetes llamada "tcpip_handler" del programa del nodo. Esta función esta encargada de "manejar" todos los datos entrantes provenientes de los diferentes nodos y los almacena dentro de la variable *appdata* desde donde se los puede manipular. Al igual que en el nodo emisor, se añade una nueva variable de tipo unsigned char (para realizar el proceso de desencriptación) nombrada como *block_decrypted* como muestra la Figura 58.

Abrir — Receptor.c
~/contiki/examples/sd6n-encry/rpl-border-router-sd6wsn

```
107 static void
108 tcpip_handler(void)
109 {
110         char *appdata;
111         char buf_send[MAX_PAYLOAD_LEN];
112         if(uip_newdata()) {
113                 appdata = (char *)uip_appdata;
114                 appdata[uip_datalen()] = 0;
115
116                 PRINTF("DATA recv '%s' from ", appdata);
117
118                 unsigned char block_decrypted[MAX_PAYLOAD_LEN];
119                 int i;
120                 for (i = 0; i < MAX_PAYLOAD_LEN; i += BLOCK_LENGTH/8) {
121                 decrypt(appdata + i, block_decrypted + i);
122                 }
123
124                 PRINTF("%d",
125                 UIP_IP_BUF->srcipaddr.u8[sizeof(UIP_IP_BUF->srcipaddr.u8) - 1]);
126                 PRINTF("\n");
```

Figura 58. Recepción y desencriptación de los datos entrantes al controlador.

Fuente: Autoría.

A continuación, se describe el proceso del descifrado que se muestra en la Figura 58:

- La línea 112 funciona para que cuando llegue algún dato, este sea almacenado en la variable appdata.
- En la línea 122 se llama a la función decrypt de las librerías Speck para la desencriptación, la cual necesita de dos argumentos necesarios. El primer argumento es la variable appdata que contiene los datos encriptados recibidos desde los nodos y el segundo argumento es la variable block_decrypted que contendrá los datos desencriptados.

Cuando ya se cuenta con la información original enviada por los nodos, estos deben imprimir para mostrar su contenido-

En el proceso de envío de respuesta está regido por las condiciones de conexión con los nodos (Figura 59, línea 130), se inserta la variable *block_decrypted* con su contenido previamente desencriptado y se define su tamaño. Para la visualización de los paquetes salientes y entrantes del controlador, se cuenta con la función print en cada uno de los nodos,

los cuales mostrarán el mensaje recibido y que será visible en la ventana “Mote output” del simulador Cooja.

```
127 #if SERVER_REPLY
128                 PRINTF("DATA sending reply\n");
129                 uip_ipaddr_copy(&server_conn->ripaddr, &UIP_IP_BUF->srcipaddr);
130                 uip_udp_packet_send(server_conn, block_decrypted, sizeof(block_decrypted));
131                 uip_create_unspecified(&server_conn->ripaddr);
132 #endif
133         }
134 }
```

Figura 59. Proceso de envió de respuesta del controlador hacia los nodos.

Fuente: Autoría.

Para la comprobación de la encriptación, envío y recepción se realizan las respectivas pruebas y se mostraran los resultados.

4.4 Integración del Sistema

Una vez realizados los respectivos cambios en los programas, se realiza el procedimiento anteriormente descrito de añadir a la plataforma de simulación. Ahora para el programa de los nodos se carga el archivo con las líneas modificadas *Emisor.c* del ANEXO C-1. Para el programa del controlador se carga igualmente el archivo modificado *Receptor.c* que se muestra en el ANEXO C-2.

Con los elementos del sistema configurados con el algoritmo de encriptación Speck, se integra estos nuevos programas al sistema previamente dimensionado para el funcionamiento de la red. Y con esta nueva funcionalidad añadida en los nodos y controlador se observar su comportamiento al generar datos para enviarlos al controlador.

Con el objeto de mejorar el entendimiento del sistema completo, a continuación, se realiza una explicación del funcionamiento al integrar la encriptación al sistema la cual esta visualmente descrita en la Figura 60 que muestra el proceso.

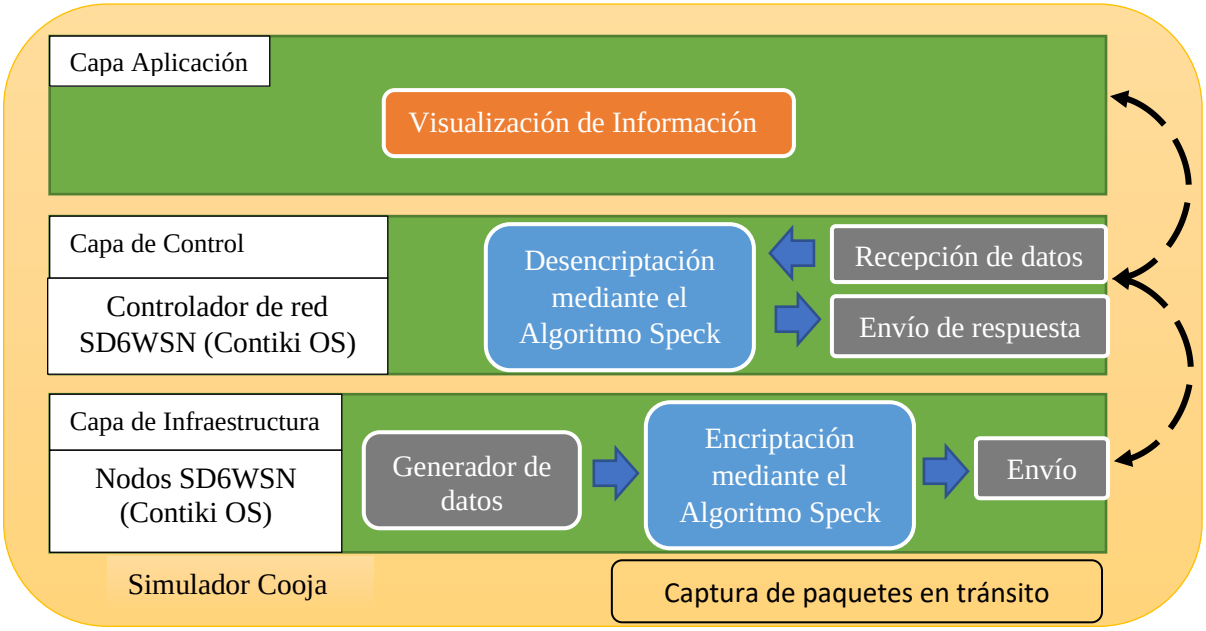


Figura 60. Descripción del proceso de intercambio de paquetes encriptados entre el controlador y los nodos.

Fuente: Autoría.

Capa Infraestructura: dentro de esta capa se encuentran añadidos los nodos sd6wsn basado en el sistema operativo ContikiOs, el cual realiza la función del recolector o generador de datos. El funcionamiento de la red parte desde los nodos con la generación de datos; dentro de la capa infraestructura se implementó la función de encriptación Speck para que los datos pasen un proceso de encriptación previo su envío; cuando se tiene los datos encriptados, estos se envían hacia el controlador y espera su respuesta.

Capa de Control: en esta capa el controlador al igual que los nodos cumple la función de recibir los datos provenientes de la capa inferior. El controlador al iniciar su ejecución primero cumple la función de generar la conectividad en la red instalando reglas en las tablas de flujo de cada nodo. Cuando ya cuenta con conectividad entre todos los nodos comienza su funcionamiento de recibir todos los mensajes entrantes.

Al igual que en la capa infraestructura, en esta capa se encuentra una parte del algoritmo Speck, que es la función desencriptación que se implementa dentro del controlador. Una vez

que el controlador recibe los datos, estos pasan por el proceso de desencriptación y al recibir los datos, se envía un mensaje de recepción del paquete en respuesta, hacia el origen.

Capa Aplicación: en cada proceso de envío y recepción, se cuenta con el visualizador de la red que permite observar todo intercambio entre la capa aplicación e infraestructura.

Siendo así que, se puede corroborar que los paquetes que se reciben en el controlador son los mismo que son enviados desde los nodos emisores. De igual forma dentro de esta capa se puede realizar capturas de los paquetes transmitidos para un análisis posterior mediante la herramienta Wireshark.

En la Figura 61, se puede observar los mensajes compartidos entre los 8 nodos y el controlador. Los mensajes se dividen en tres secciones:

- La primera sección es el contenido (datos) generado por los nodos.
- La segunda sección son los datos encriptados los cuales son enviados al controlador.
- La tercera sección es la respuesta del controlador.

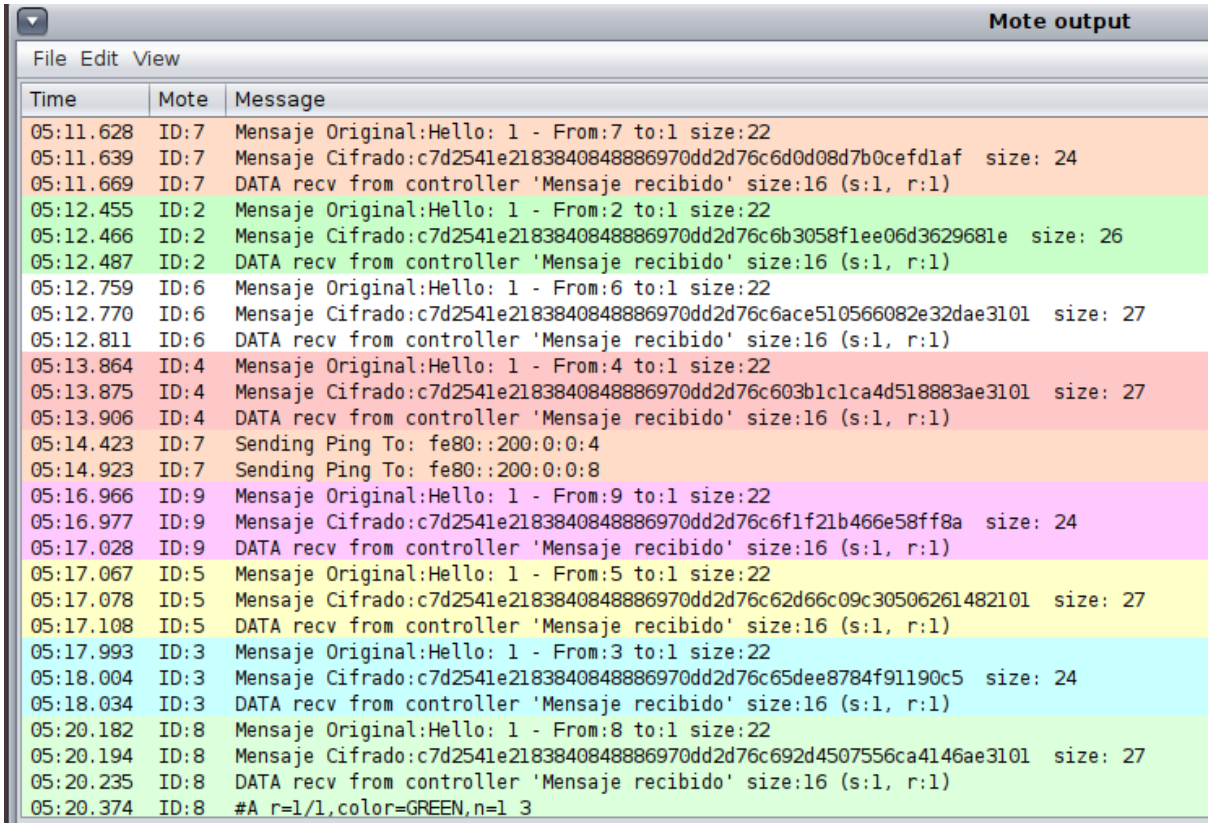


Figura 61. Vista de panel "Mote output" con los mensajes generados, encriptados, enviados y recibidos por los 8 nodos emisores en la red.

Fuente: Autoría.

CAPITULO V: PRUEBAS Y RESULTADOS

Este capítulo comprende las pruebas realizadas al ambiente de simulación una vez que el sistema se encuentra completamente integrado con todos los elementos de red. Las pruebas realizadas son en base al correcto funcionamiento del ambiente SD6WSN su conectividad y la correcta encriptación de los datos.

Las pruebas realizadas están divididas por funcionamiento, rendimiento, consumo de recursos tales como energía y memoria. En el funcionamiento se realiza una comparación del escenario de pruebas antes y después de la implementación del mecanismo de seguridad; en el rendimiento se contrasta, se mide los tiempos de respuesta y tasa de recepción de paquetes; en el consumo de energía se muestra los cambios que genera el aumentar una nueva funcionalidad a los dispositivos; y finalmente se mide el consumo de memoria que conlleva implementar el mecanismo de seguridad.

5.1 Prueba de encriptación del algoritmo Speck dentro de la red SD6WSN

Para verificar que la encriptación realmente tiene el efecto deseado en el intercambio de mensajes entre los nodos y el controlador, se genera un registro de las comunicaciones y se los almacena en un archivo PCAP. Cooja al ser un ambiente de simulación completo, cuenta con un capturador de mensajes de radio incorporado, el cual a su vez permite almacenar los datos capturados en archivos en formato compatibles con el software Wireshark para su análisis.

Para verificar que la encriptación es funcional, se realiza una comparación de las comunicaciones antes y después de implementar la encriptación. Los archivos generados al capturar los paquetes en tránsito se almacenan en la misma carpeta donde se guarda la simulación.

5.1.1 Captura de tráfico previa a la integración de la encriptación

Al ejecutar la red sd6wsn, se tiene un ambiente totalmente libre de modificaciones en su programación y funcionalidades, por tal motivo se puede evaluar el desempeño de esta red al momento de enviar y recibir paquetes de mensajes para observar su comportamiento. En la Figura 62, se aprecia una captura de un paquete enviado desde la el nodo 9 (::200:0:0:9) hacia el nodo 1 con la dirección (::200:0:0:1) que corresponde al controlador.

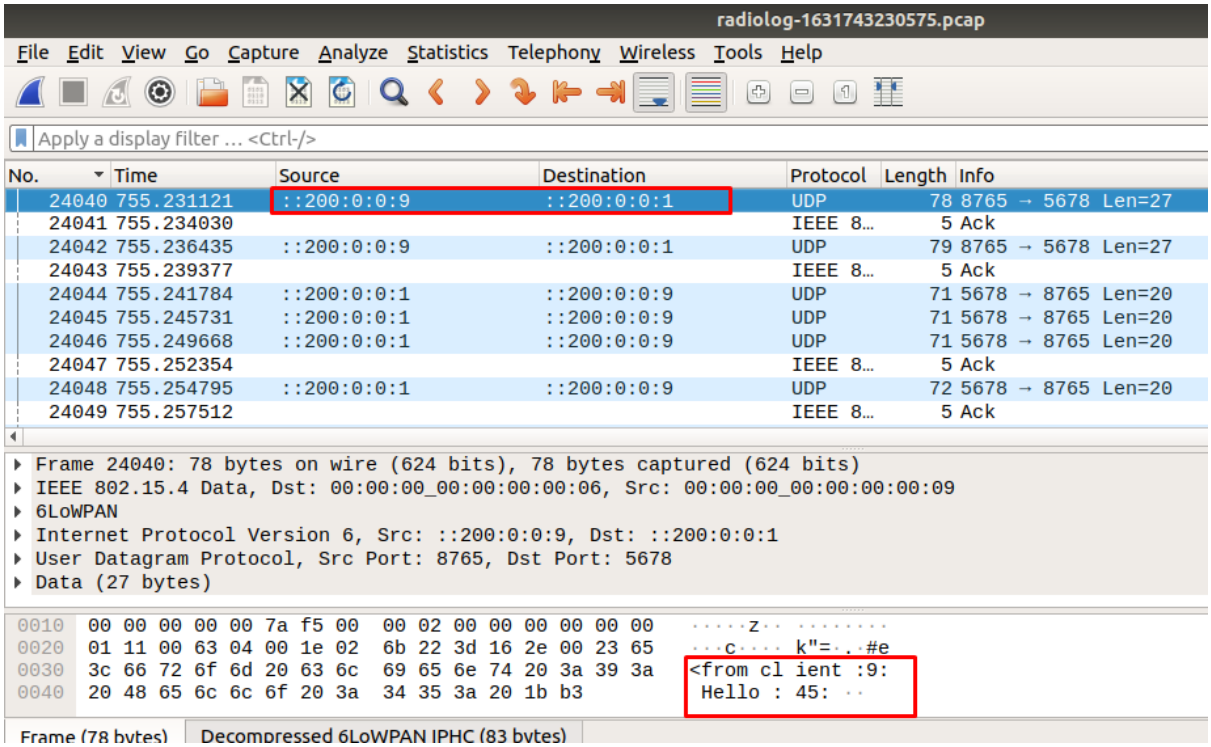


Figura 62. Captura de paquete enviado desde el nodo 9 al controlador visto desde Wireshark

Fuente: Autoría

En la Figura 62, se observa que al seleccionar el paquete 24040, en la parte inferior del programa se pueden notar los valores que comprende el paquete y se nota claramente que la carga útil que el paquete lleva se distingue el mensaje “from client :9 Hello: 45” dando a entender que cualquier mensaje que sea enviado será fácilmente vulnerado. El nodo 9 es uno de los más alejados del controlador, por tanto, es una referencia del funcionamiento de la red al enviar un paquete hacia el nodo raíz.

De igual forma se selecciona otro paquete para corroborar que el comportamiento anterior sucede en todos los dispositivos. Para ello se analiza el paquete con dirección origen proveniente del nodo 5 (::200:0:0:5) y dirección destino el nodo 1 (::200:0:0:1). En la Figura 63 se puede observar que al igual que en el caso anterior, el contenido del paquete es notablemente visible y se puede entender el mensaje que contiene.

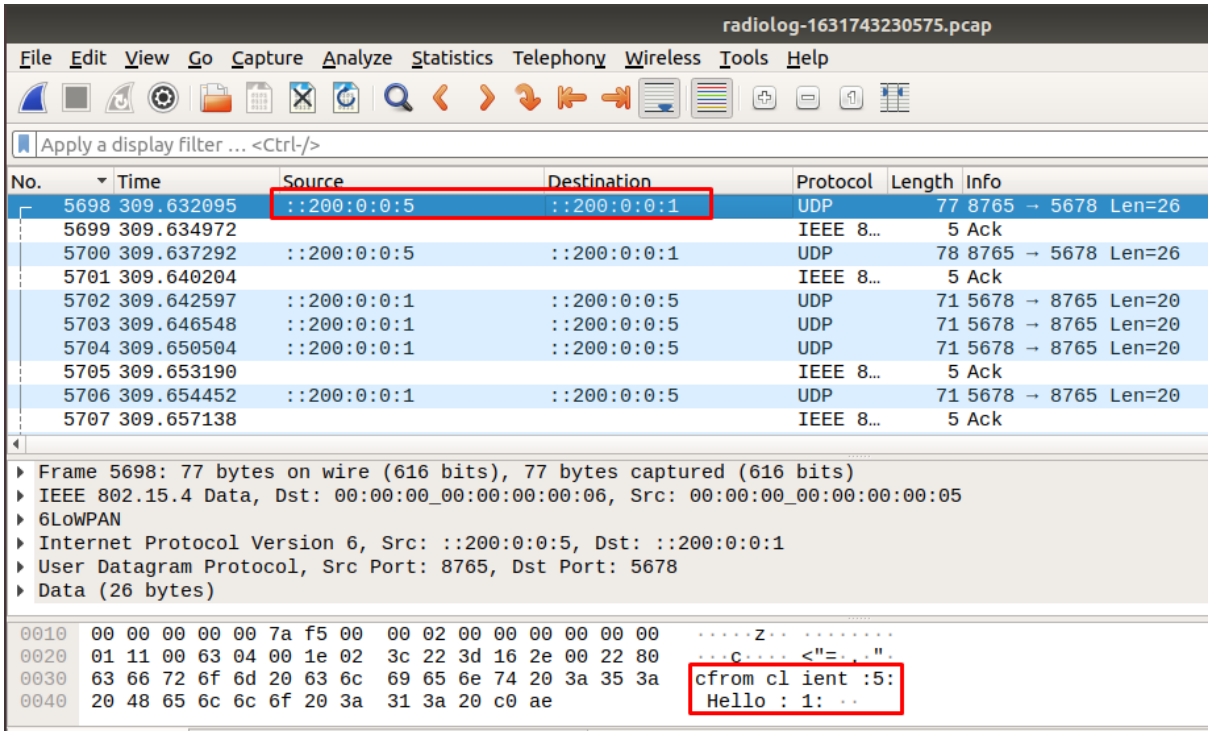


Figura 63. Captura de paquete enviado desde el nodo 5 al controlador visto desde Wireshark.

Fuente: Autoría

En la Figura 63, se puede notar que al igual que cuando el nodo 9 envió un mensaje, su contenido también es visible desde el programa Wireshark, por lo que se puede afirmar que este entorno no es nada seguro para las comunicaciones inalámbricas y que toda comunicación es vulnerable a terceros en el caso de captura de tráfico.

5.1.2 Captura de tráfico posterior a la integración de la encriptación

Al realizar la implementación de la encriptación y ejecutar las pruebas de simulación se evalúa nuevamente el desempeño de las comunicaciones inalámbricas para observar su comportamiento y se obtuvo nuevos resultados al momento de realizar las capturas de tráfico

pertinentes en la red. En las capturas de los paquetes, nuevamente se tomó como ejemplo los paquetes enviados desde el nodo 9 hacia el controlador y se observó un cambio importante en los mensajes capturados, como la Figura 64 muestra.

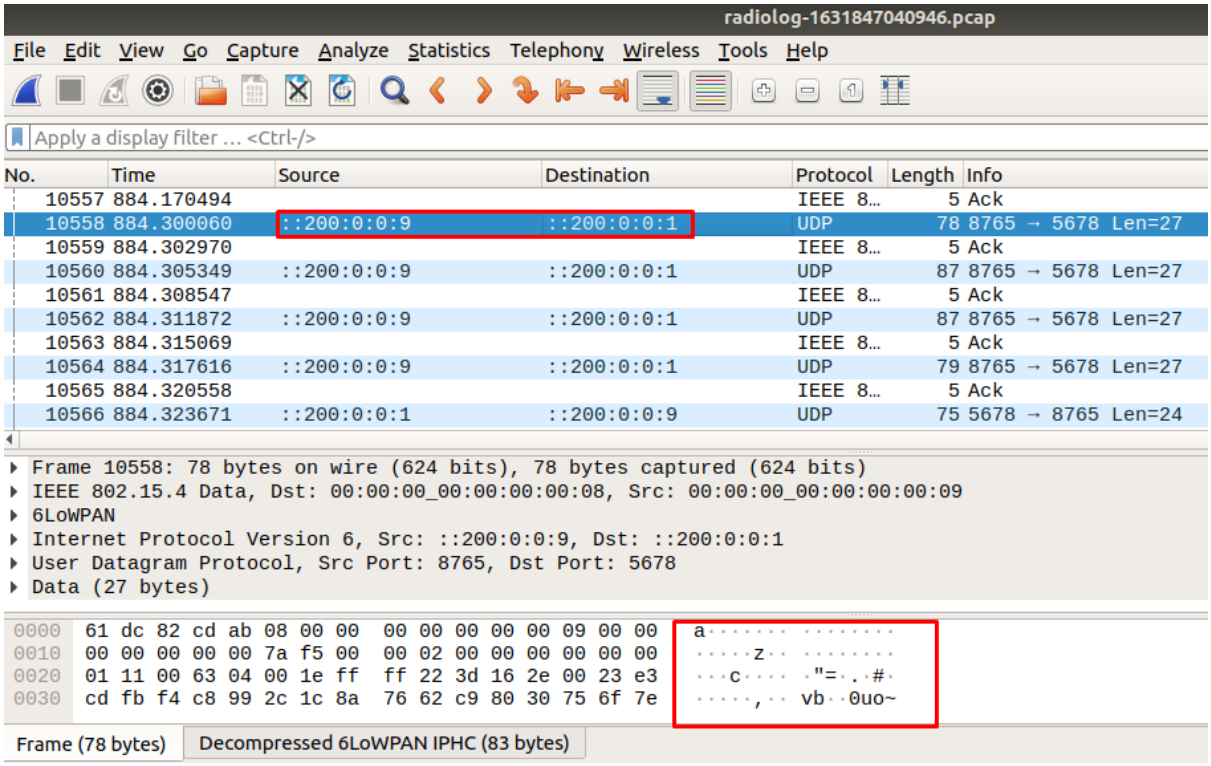


Figura 64. Captura de paquete encriptado enviado desde el nodo 9 al controlador visto desde Wireshark.

Fuente: Autoría.

En la Figura 64, se puede notar que las direcciones de origen y destino son las mismas que las pruebas anteriores, pero en este caso, al ir directamente a la carga útil que el paquete contiene, no se puede distinguir alguna palabra o algún sentido al mensaje, permitiendo señalar que, con respecto al ambiente anterior, este representa una mejora considerable en cuanto a la seguridad de los mensajes enviados hacia el controlador.

Al igual que en la prueba del escenario anterior, se selecciona el paquete del nodo 5 hacia el controlador como muestra la Figura 65 y se puede apreciar que el mensaje se encuentra encriptado y no se puede distinguir ningún dato de su contenido.

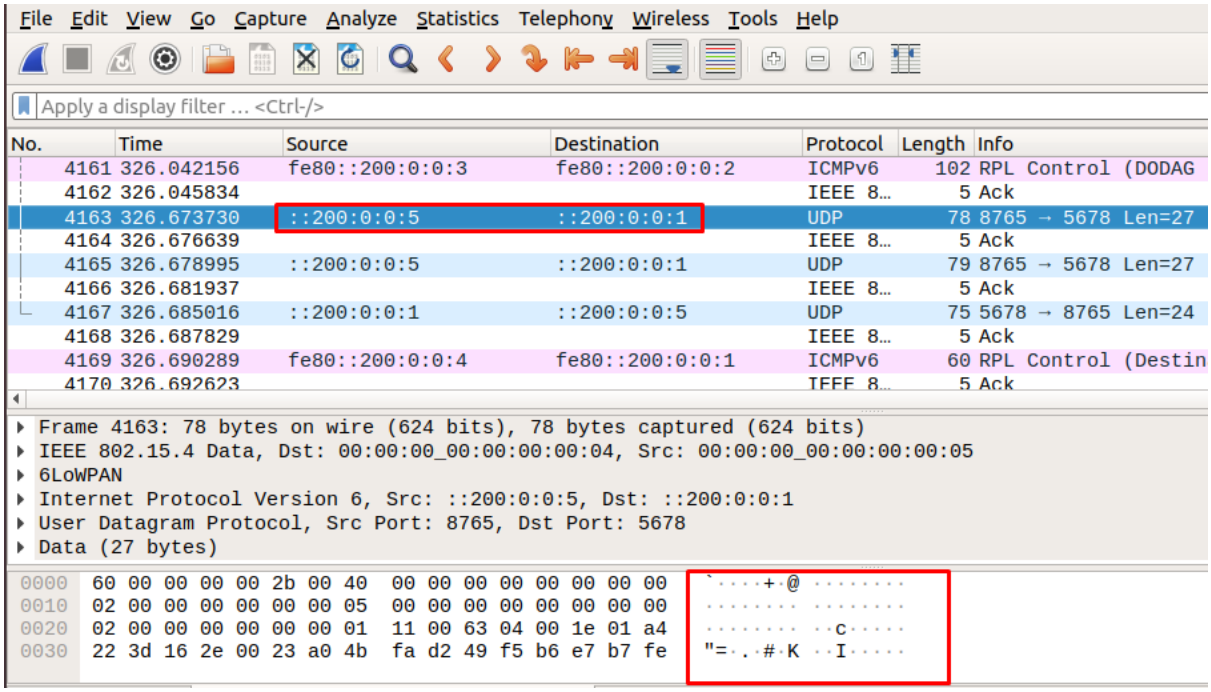


Figura 65. Captura de paquete encriptado enviado desde el nodo 5 al controlador visto desde Wireshark.

Fuente: Autoría.

Con esta prueba se pudo contrastar el estado de la red mediante las capturas del tráfico antes y después de implementar la encriptación y se notó la diferencia entre los dos tipos de ambiente. Con ello, se puede concluir que los paquetes del ambiente con encriptación Speck no muestran su contenido al momento del análisis mediante un sniffer, contrario a lo que se evidencio en el ambiente de pruebas inicial; ambos ambientes fueron probados en condiciones similares con un contenido de mensaje igual, pero se puede notar que, aunque el tamaño de la carga útil aumenta en el segundo escenario con respecto al primer, este ofrece una mejora en la seguridad evitando que los mensajes o los datos que contienen cada paquete puedan ser observados por terceros.

5.2 Rendimiento

Para realizar una medición del rendimiento general de la red, las pruebas se apegaron a la capacidad del simulador para generar archivos que registren la actividad de los nodos en los parámetros de latencia y entrega de paquetes. Como estas dos mediciones están relacionadas

entre sí ambas hacen uso de la función que se observa en la Figura 66; cada nodo dentro de su programación, tienen incorporado la función “printf” que imprime el número de mensajes enviados hacia el controlador y el número de estos que reciben una respuesta.

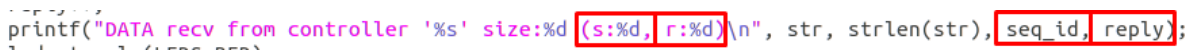


Figura 66. Función de imprimir el número de mensajes enviados y respuestas recibidas.

Fuente: Autoría.

El contenido de la Figura 66 se compone de la siguiente forma:

- *DATA recv from controller '%s':* Muestra el contenido de la variable *str* que corresponde al mensaje recibido del controlador.
- *size:%d*: Muestra el tamaño del mensaje recibido del controlador obtenido de *strlen(str)*
- *s:%d*: Muestra el valor de la variable *seq_id* que es un contador del número de mensajes enviados al controlador
- *r:%d*: Muestra el valor de la variable *reply* que es un contador del número de mensajes recibidos del controlador

Con la ayuda de esta función se puede obtener el número de paquetes enviados por los nodos y el número de respuestas del controlador para calcular el PRR. Y dado que estos valores se imprimen a lo largo del tiempo, se puede obtener los valores de latencia en la red. Para realizar una evaluación correcta, se debe obtener el registro previo y posterior a la implementación del algoritmo Speck.

5.2.1 Latencia

Para medir el retardo de la red al enviar un mensaje y recibir una respuesta, se toma los valores generados por los mismos nodos los cuales imprimen estas métricas. Cuando los nodos

envían un mensaje, estos reciben un mensaje de confirmación por parte del controlador. Los mensajes de impresión se almacenan en un archivo de registro de la simulación.

En base al momento exacto de envío y de recepción se calcula la latencia para todos los paquetes. La latencia de la red de acuerdo con (Ali, 2012) se puede calcular usando la fórmula de la Ec.4 a continuación:

$$Latencia\ Total = \sum_{k=1}^{n} (Recv\ Time(k) - Send\ Time(k)) \qquad \text{(Ec.4)}$$

Donde n es el número de paquetes recibidos exitosamente y el tiempo es proporcionado por el simulador. Para calcular la latencia promedio, se suma la latencia total de la Ec.4 obtenida de cada nodo y se la divide por el número de nodos en la red.

Al tomar los valores de tiempo del registro de la simulación previo a la implementación de la encriptación se filtra lo mensajes enviados y recibido con su respetivo tiempo y mediante la Ec.4 se obtiene la latencia promedio en cada nodo. Estos valores se ven reflejados en la Tabla 26.

Tabla 26

Valores de latencia promedio previa implementación del mecanismo Speck

Nodo	Latencia Promedio (ms)
2	00:00.017
3	00:00.028
4	00:00.017
5	00:00.033
6	00:00.041
7	00:00.028
8	00:00.042
9	00:00.057

Fuente: Autoría

De la misma forma, se realiza el mismo procedimiento en la red posterior a la implementación de la encriptación para verificar el retardo que tiene la red al enviar y recibir un mensaje. Con ayuda del registro de actividad se obtiene los valores y con la Ec.4 se obtiene los valores promedio de latencia en cada nodo. Estos valores se evidencian en la Tabla 27.

Tabla 27

Valores de latencia promedio posterior implementación del mecanismo Speck

Nodo	Latencia Promedio (ms)
2	00:00.026
3	00:00.040
4	00:00.025
5	00:00.039
6	00:00.051
7	00:00.039
8	00:00.050
9	00:00.065

Fuente: Autoría

Al obtener los valores generales de cada nodo en los dos escenarios, se crea una gráfica comparativa entre antes y después para verificar el impacto que ha tenido el añadir una nueva funcionalidad a la red. A partir de los valores adquiridos previamente se crea la Figura 67, donde se puede observar cómo han sido afectados todos los nodos en cuanto al tiempo de respuesta en la red.

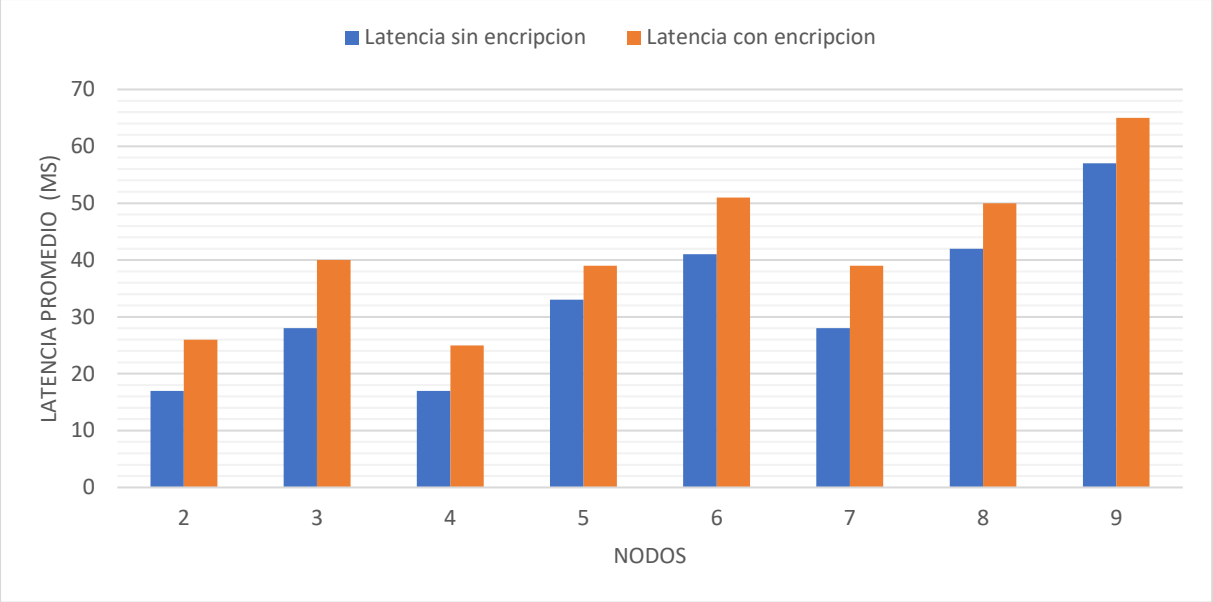


Figura 67. Gráfico comparativo de latencia antes y después.

Fuente: Autoría.

Si bien el aumento de la latencia es un factor importante para una WSN, el no disponer de algún mecanismo de seguridad para la transmisión de datos es una de las mayores vulnerabilidades en una red. La latencia aumentada es debido a que, en el primer escenario los nodos únicamente generan el paquete y envían hacia su destino; el controlador muestra el contenido del paquete y devuelve un mensaje de confirmación. En el segundo escenario, los nodos generan el mensaje y pasan al proceso de cifrado para posteriormente enviar; cuando llega el paquete al controlador, primero descifra el paquete para posteriormente enviar un mensaje de confirmación. Estos factores son los que generan el retardo en la red, pero al realizar un promedio general del retardo entre todos los nodos se obtiene que el incremento no supera los 9ms lo que es un valor aceptable para el tipo de red que se maneja y la seguridad adicional con la que cuenta.

5.2.2 Tasa de recepción de paquete (PRR)

PRR es calculado como la relación porcentual de recepción de paquetes promedio. PRR tiene en cuenta solo los paquetes unicast como los paquetes de datos. Para ello se envía mensajes “Hello” hacia el controlador y este responde con un mensaje de confirmación; para calcular el PRR, es necesario utilizar la Ec.5.

	$PRR\ promedio = \frac{Paquetes\ totales\ Recibidos}{Paquetes\ totales\ enviados} \times 100$	(Ec.5)

Los mensajes de impresión de los mensajes enviados y recibidos se almacenan de igual forma en un archivo de registro de la simulación. A partir de este registro se puede obtener el número de paquetes enviados hacia el controlador y el número de paquetes que se respondieron. Se presenta un resumen de este registro en la Tabla 28.

Tabla 28

PRR de la red previa implementación de la encriptación.

Nodo	Paquetes Enviados	Paquetes de respuesta
2	59	56
3	58	55
4	59	58
5	59	56
6	59	55
7	59	58
8	59	55
9	59	53
Promedio	58.875	55.75

Fuente: Autoría.

Con los valores de la tabla anterior se obtiene que el número de paquete recibidos es de 446 y los paquetes enviados es 471 que, al reemplazar estos valores en la Ec.5 genera un resultado

aproximado de 94,69% de tasa de entrega de los paquetes en la red al implementar la encriptación.

Luego de implementar la encriptación nuevamente se obtiene las métricas con respecto a la tasa de recepción de paquetes en la red. Los nuevos registros muestran los resultados mostrados en la Tabla 29.

Tabla 29

PRR de la red posterior a la implementación de la encriptación.

Nodo	Paquetes Enviados	Paquetes de respuesta
2	59	58
3	59	55
4	59	58
5	59	56
6	59	54
7	59	55
8	59	54
9	59	55
Promedio	59	55,625

Fuente: Autoría.

Con los valores de la tabla anterior se obtiene que el número de paquete recibidos es de 445 y los paquetes enviados es 472 que, al reemplazar estos valores en la Ec.5 genera un resultado aproximado de 94,28% de tasa de entrega de los paquetes en la red al implementar la encriptación.

Los valores de la recepción de paquetes en ambos casos son similares variando con respecto al número de paquetes recibidos. Con esta prueba se puede conocer el estado de la red en cuanto al enlace y latencia, que si en caso de fallar se ve reflejado en el PRR. El error porcentual al contrastar ambos escenarios es de 0,41% lo cual indica que al aumentar el cifrado, no genera perdida de paquetes significativa en los nodos.

5.3 Consumo de energía

Para obtener los valores del consumo de energía de cada nodo se utiliza la herramienta Powertrace que tiene integrado el sistema operativo Contik. Powertrace es un sistema para la generación de perfiles de potencia a nivel de red para redes inalámbricas de baja potencia que estima el consumo de energía para el procesamiento del CPU, la transmisión de paquetes y la escucha. Este mecanismo mantiene una tabla de la duración de un componente como el CPU o el transmisor de radio. Basándose en esta tabla, se obtiene el porcentaje de radio a tiempo. Se calcula el consumo de energía para la transmisión y escucha de radio, dado que estos son los componentes que consumen más energía.

Cada 30 segundos se obtiene una salida generada por powertrace en la ventana Mote output. Estos mensajes por lo general constan de 16 columnas que se encuentran separadas por espacios; de estos valores solo las columnas 5, 6, 7 y 8 son relevantes debido a que representan ALL_CPU, ALL_LPM, ALL_TX y ALL_RX respectivamente y que se encuentran marcadas en rojo en la Figura 68.

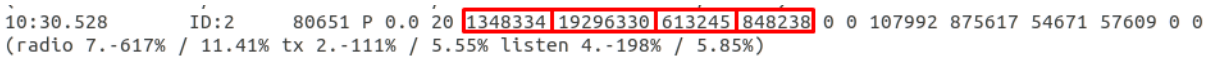


Figura 68. Ejemplo de salida de powertrace para el nodo 2.

Fuente: Autoría

ALL_CPU es el número total de tics de reloj cuando la CPU está en modo activo. ALL_LPM es el total de tics del reloj en el estado del modo de bajo consumo. ALL_TX es el número total de tics de reloj en el estado de transmisión. ALL_RX es el número total de tics de reloj en el estado de recepción. En la Tabla 30 se presenta una muestra de los datos obtenidos del nodo 2 previo a la implementación de la encriptación y que refleja el comportamiento que cada nodo presenta los cuales pueden ser observados en el ANEXO D-1.

Tabla 30

Muestra de datos obtenidos del nodo 2 previo a la implementación de la encriptación.

	ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
0	311928	18366085	50939	18594657
1	326734	19334321	52903	19575729
2	342275	20301818	55300	20556369
3	358465	21268667	57523	21537179
4	378940	22231233	60754	22516976
5	395315	23197900	63180	23497590
6	411746	24164507	65442	24478361
7	433324	25125969	69232	25457598
8	448363	26093970	71237	26438628
9	468178	27057197	74124	27418779
10	487893	28020520	77393	28398543

Fuente: Autoría.

Estos valores generados deben ser procesados para entender la energía consumida por los nodos. Los valores de las columnas descritas anteriormente representan el tiempo en forma de tics, donde un segundo consta de 32768 tics. Por tanto, el tiempo de 15 minutos corresponde a 29491200 tics. De esta forma, puede obtener el tiempo en forma de tics de cada estado.

Para realizar el cálculo de consumo de energía se utiliza la fórmula de la Ec.6 a continuación:

$$Consumo\ de\ energia = \frac{Valor\ total\ de\ energia\ \times corriente \times voltage}{(\text{RTIMER\_SECOND}\ \times \text{Tiempo de ejecucion})} \quad \text{(Ec.6)}$$

El valor total de la energía se obtiene tomando la diferencia entre el número de tics de un intervalo y su intervalo previo.

Donde, de acuerdo con el datasheet de la plataforma Wismote (Mokrenko, 2015), los valores de su funcionamiento son:

- Voltaje: 3V
- Corriente: 2.2mA, 2µA, 25.8mA y 18.5mA para CPU, LPM, Tx y Rx respectivamente.
- RTIMER: 32768
- Tiempo de ejecución: 900s

Reemplazando los valores de voltaje y corriente en la ecuación, el consumo de energía total en mW de los 10 últimos intervalos se muestra en la Tabla 30. En el ANEXO E-1 se encuentra el cálculo del resto de nodos en la red.

Tabla 31

Muestra de cálculo de energía para 10 intervalos del Nodo 2 previo a la implementación.

	CPU (mW)	**LPM (mW)**	**TX (mW)**	**RX (mW)**	**TOTAL (mW)**
1	0.00331352	0.00019699	0.00515454	1.84629639	1.85496143
2	0.00347801	0.00019684	0.00629095	1.8454834	1.8554492
3	0.00362325	0.00019671	0.00583429	1.84580332	1.85545757
4	0.00458221	0.00019583	0.0084798	1.84389694	1.85715479
5	0.00366465	0.00019667	0.00636707	1.84543447	1.85566286
6	0.00367719	0.00019666	0.00593665	1.84572993	1.85554042
7	0.00482906	0.00019561	0.0099469	1.84284307	1.85781464
8	0.00336566	0.00019694	0.00526215	1.84621735	1.85504209
9	0.00443451	0.00019597	0.00757697	1.84456314	1.85677058
10	0.00441213	0.00019599	0.00857953	1.84383484	1.85702249

Fuente: Autoría.

El promedio de consumo de energía total durante los 10 últimos intervalos para el nodo 2 es de 1.85608761 mW.

Luego de implementar la encriptación Speck, nuevamente se ejecuta la simulación y se mide los mismos parámetros, obteniendo nuevos valores de consumo. En la Tabla 32 se

observa los datos obtenidos de nodo 2. En el ANEXO D-2 se observa los valores obtenidos para el resto de los nodos sensores.

Tabla 32

Muestra de datos obtenidos del nodo 2 posterior a la implementación de la encriptación.

	ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
0	328593	18349420	50965	18594609
1	348581	19312474	53738	19574873
2	366127	20277966	55917	20555726
3	384992	21242140	58537	21536142
4	408401	22201771	62353	22515360
5	426153	23167062	64758	23495990
6	442808	24133445	66586	24477197
7	467123	25092169	70799	25456015
8	487233	26055099	73813	26436037
9	506078	27019455	76373	27416674
10	527622	27980791	79474	28396444

Fuente: Autoría.

Al conocer las métricas de funcionamiento con que trabaja la plataforma Wismote, nuevamente se reemplazan todos los valores en la ecuación Ec.6 y se añaden los resultados obtenidos en la anterior tabla para conseguir los rangos de consumo de todos los nodos en la red. El consumo de energía total del nodo 2 en mW de los 10 últimos intervalos se muestra en la Tabla 33. El cálculo de energía del resto de nodos se puede observar en el ANEXO E-2.

Tabla 33

Muestra de cálculo de energía para 10 intervalos del Nodo 2 posterior a la implementación.

	CPU (mW)	LPM (mW)	TX (mW)	RX (mW)	Total (mW)
1	0.00447323	0.00019593	0.00727777	1.8447758	1.85672273
2	0.00392672	0.00019643	0.00571881	1.84588425	1.8557262

3	0.0042219	0.00019616	0.00687622	1.84506185	1.85635613
4	0.00523883	0.00019524	0.01001514	1.84280731	1.85825652
5	0.00397282	0.00019639	0.00631195	1.84546458	1.85594574
6	0.00372732	0.00019661	0.00479761	1.84655045	1.85527198
7	0.00544159	0.00019505	0.01105707	1.84205455	1.85874826
8	0.00450053	0.00019591	0.00791028	1.84432037	1.85692709
9	0.00421743	0.0001962	0.00671875	1.84547775	1.85661013
10	0.00482145	0.00019558	0.00813861	1.84384613	1.85700178

Fuente: Autoría.

Promedio del consumo de energía total durante los 10 últimos intervalos para el nodo 2 es de 1.85675666mW.

Al realizar el proceso del cálculo para todos los nodos en la red, se consigue un consumo general de energía en los dos escenarios de simulación. El comparar las medidas obtenidas previo (SE) y posterior a la implementación (CE), permite construir la Figura 69 donde se aprecia que la energía requerida por los nodos es mayor en la mayoría de los casos y aumenta con respecto al escenario sin encriptación.

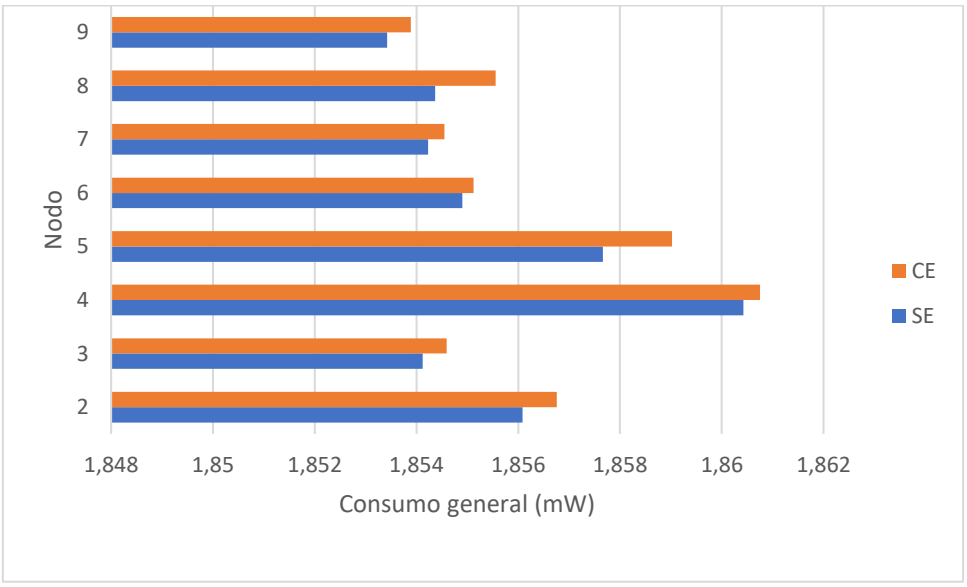


Figura 69. Comparación de consumo de energía en escenarios antes y después de la implementación de la encriptación.

Fuente: Autoría.

Si bien el valor de la energía requerida aumenta por parte de los nodos sensores cuando cuentan con el proceso de encriptación, al realizar un cálculo promedio se obtiene que el aumento general es de 0.00065295mW con respecto al escenario sin la encriptación. Considerando que el aumento en algunos casos es más alto que otros, es un costo de energía del cual se puede disponer, teniendo en cuenta el nuevo mecanismo de seguridad en la red y que su consumo no sube más allá del 0.165% (en el caso de mayor diferencia) siendo un valor máximo aceptable para una red de características SD6WSN.

5.4 Memoria RAM y ROM

Debido a que es un dispositivo en el cual el espacio de memoria es estrechamente limitado, la implementación se divide entre consumo de memoria RAM y ROM. En el caso de la mota Wismote el algoritmo usado es almacenado dentro de la ROM, pero las variables son almacenadas en la RAM. En el presente trabajo, la medición de la ROM utilizada fue mediante el comando *size* proporcionado por el compilador MSP430-gcc utilizado, el cual calcula el valor automáticamente del archivo compilado. Normalmente, el consumo de ROM del programa es la recopilación considerada de bytes de texto y datos, adoptada para determinar el consumo de ROM utilizado únicamente para la parte de cifrado dedicada.

Para verificar el aumento en el consumo de memoria en los nodos se utiliza uno de los comandos que vienen integrados con el sistema operativo Contiki y que se obtuvo de la página oficial (Contiki-NG, n.d.). Al ingresar el comando *size Emisor.wismote* y *size sd6wsn-agent.c* en el terminal de Linux, muestra varios valores como muestra la Figura 70.

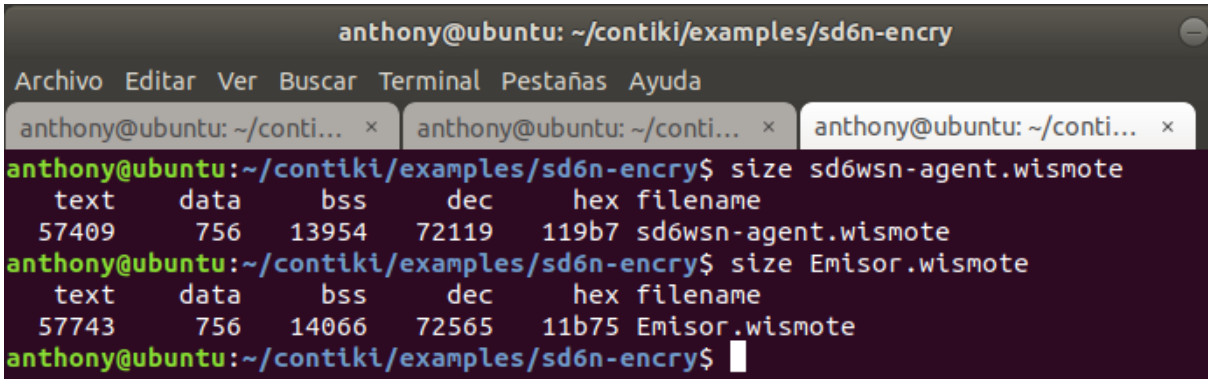


Figura 70. Obtención de valores de la Ram y Rom de los nodos emisores.

Fuente: Autoría.

De igual forma, se realiza la misma operación con el controlador para obtener sus valores en consumo de RAM y ROM como muestra la Figura 71.

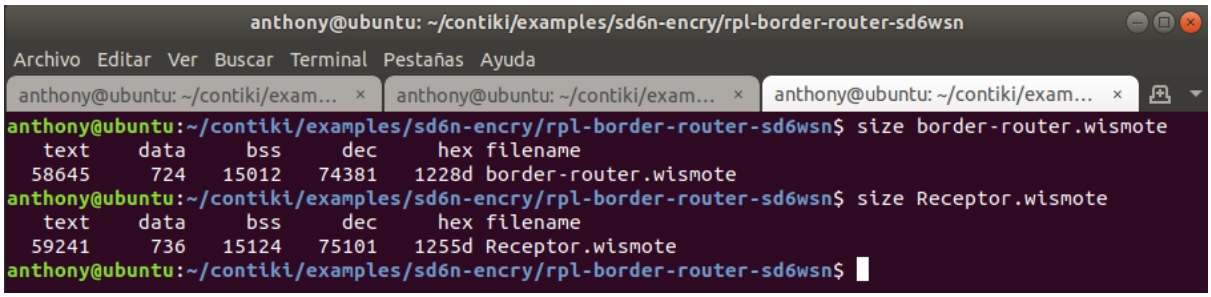


Figura 71. Obtención de valores de la Ram y Rom del controlador.

Fuente: Autoría.

De acuerdo con página oficial, el valor de la columna *text* corresponde al tamaño de memoria ROM en bytes que está siendo utilizada; en cuanto al uso de la memoria RAM, corresponde los valores de las columnas *data* y *bss* que contienen variables almacenadas. Este proceso se realizó con los archivos compilado antes y después de implementar el algoritmo de cifrado. Para obtener el valor de aumento, se contrastan los valores de estos dos archivos a fin de realizar una operación básica de sustracción y obtener un valor general como se presenta en la Tabla 34.

Tabla 34

Comparación de consumo de memoria antes y después de implementar la encriptación

Elemento	**ROM (bytes)**			**RAM (bytes)**		
	Antes	Después	Aumento	Antes	Después	Aumento
Nodo	57409	57743	334	14710	14822	112
Controlador	58645	59241	596	15736	15860	124

Fuente: Autoría.

Se puede observar en la tabla anterior que el aumento de utilización de memoria en ambos casos no es alto.

De forma general el aumento de recursos ocupados se encuentra dentro de los parámetros aceptables, esto debido a que la capacidad del dispositivo es de 128KB de memoria flash y 16KB de memoria SRAM tanto para el nodo como para el controlador. El consumo de ROM en el nodo es de 57.7KB y en el controlador de 59.2KB incrementando un 0.58% y 1.02% con respecto al espacio utilizado. En cuanto a la memoria RAM, en el nodo es de 14.8KB y en el controlador es de 15.8KB aumentando un 0,76% y 0,79% respectivamente acercándose al límite de la capacidad, pero no sobrepasando estos valores. Obteniendo así, que el cifrado es adecuado para ser implementado dentro de los nodos Wismote, teniendo la suficiente capacidad para funcionar sin generar sobrecargas de memoria dentro de sus dispositivos emulados.

Al final se obtiene el canal de comunicación cifrado basado en el algoritmo Speck con el protocolo 6LoWPAN y en la plataforma Wismote logrando el objetivo de implementar un mecanismo de seguridad, pero bajo el análisis estadístico de las pruebas generales realizadas se nota que la encriptación demanda de varios recursos de los dispositivos, sin embargo, en términos generales estos no exceden el 1% en cada prueba, lo que garantiza que se establece un sistema con comunicación optima y seguridad añadida.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Para la implementación de un canal de comunicación cifrado se utilizó el algoritmo Speck y se dividió en dos partes; la encriptación es una función dentro de los nodos y la desencriptación una función dentro del controlador. Mediante este proceso, se puede generar mensajes que solamente el destino puede interpretar sin tener un mayor impacto en cuanto a los pocos recursos que manejan los dispositivos basados en el protocolo 6LoWPAN.

Si bien ya existen una amplia variedad de sistemas SDN para las redes tradicionales, las redes SDWSN se encuentran en desarrollo y las soluciones propuestas siguen siendo relativamente recientes. Aunque ya se cuenta con varias que funcionan con el protocolo 6lowpan, todavía presentan algunas carencias en cuanto a funcionamiento que se deben mejorar, pero con las cuales se puede realizar pruebas experimentales.

Los mecanismos ligeros de encriptación tienen una gran variedad de algoritmos que pueden ajustarse a la necesidad de las circunstancias, como en el presente trabajo el mecanismo Speck esta optimizado para su funcionamiento en software, haciéndolo adecuado para ejecutar en un ambiente de pruebas simulado.

Los nodos/motas con funciones de las redes SDN tienen procesos complejos dentro de sus códigos fuente, por tanto, el añadir la nueva función de encriptación puede resultar en el fallo de alguna de las funciones SDN, debiendo tener cuidado al momento de implementar el proceso de encriptación/desencriptación para no afectar ninguna de procesos incorporados por defecto en el nodo.

La implementación de alguna nueva función y sus respectivas librerías no solo deben ser especificadas en los archivos de las aplicaciones sino también en los archivos Makefile en

los cuales también se debe especificar su ubicación al igual que el archivo *project-conf.h*; estos tres archivos son necesarios para una compilación y funcionamiento exitoso debido a que contienen variables que están relacionadas con el ambiente de simulación.

El añadir una nueva característica a los nodos de la red, causa un impacto en cuanto al rendimiento de los nodos en la red debido a que al procesar los datos generados para encriptarlos aumenta el retardo y por ende los recursos consumidos. Si bien estos valores tuvieron un incremento con respecto al ambiente sin encriptación, los resultados reflejaron que el aumento no es relevante considerando que, al realizar las capturas de los paquetes en la red, no se visualiza el contenido o se distinguen mensajes.

Se observo que mientras más aumentar el número de nodos en la red genera provoca que la simulación en general pierda velocidad de simulación, llegando a tal punto en que se detiene por completo.

Si bien existen varios algoritmos de encriptación que tienen funciones similares, Speck fue escogido debido a su bajo consumo de memoria para su implementación con respecto a otros y principalmente porque está enfocado a la optimización de software, lo cual es una de las características que más se evalúa en los ambientes de simulación.

El ambiente de simulaciones Cooja es un software completo que permite realizar cambios en cuanto a protocolo y funcionamiento de redes WSN que permite evaluar varios parámetros en distintos escenarios. Cuenta con herramientas que permiten obtener valores en tiempo real y generar registros para posteriores análisis. Debido a que está basado en software libre permitió el cambio en la funcionalidad de sus programas y la evaluación de estos.

Recomendaciones

Para mejorar la seguridad de los canales de comunicaciones inalámbricas, se sugiere añadir una función de autenticación para la verificación de todos los dispositivos pertenecientes a la red la cual sea compatible con el ambiente SDN y el protocolo 6LoWPAN.

Incursionar en el despliegue de redes de sensores basados en contiki requiere conocer lenguaje de programación C por lo que es recomendable tener conocimientos básicos de programación, virtualización y software inalámbrico para redes WSN para un manejo óptimo de las funcionalidades de cada dispositivo.

Es recomendable para futuros despliegues añadir un servidor web dedicado con interfaz gráfica para la visualización de los mensajes, las configuraciones y la administración de los nodos de red para generar una gestión de la red mucho más optimizada que permita guardar los registros de los cambios o suceso en la red.

La implementación de una nueva función en los nodos fue en base a prueba y error para llegar a una ejecución satisfactoria, por lo que se recomienda tener copias de respaldo de los programas en caso de afectar algún proceso relacionado con la funcionalidad del dispositivo se puede revertir y evitar la reinstalación los softwares.

Al trabajar con software virtualizado es necesario contar con característica de “snapshot” para tener respaldos de las máquinas virtuales en caso de que estas se vean corrompidas por algún archivo o sistema mal configurado logrando así, volver a una etapa anterior antes de que el sistema tenga un mal funcionamiento.

Los sistemas Contiki son complejos, pero cuentan con documentos actualizados y una comunidad muy activa. En caso de requerir conocer alguna funcionalidad o proceso, se puede buscar en la wiki oficial de Contiki en GitHub donde se encuentra todos los archivos fuentes y las explicaciones de cada caso para la implementación.

Al realizar la conexión del nodo controlador a la maquina Host verificar que sea el nodo 1 dado que las instrucciones están programadas para funcionar desde este nodo; en caso usar otro se debe realizar los respectivos cambios en los archivos de ejecución. Y debido que los archivos están desarrollados en Java, se recomienda conocer los conceptos de este lenguaje de programación en caso de añadir una nueva funcionalidad.

Algunas de sus variables son sensibles en cuanto al tipo de datos (la función de envío solo permite variables tipo char), por lo que al implementar una nueva funcionalidad se recomienda verificar si el tipo de variable afecta o no al desempeño del sistema, que en caso de ser necesario se puede cambiar utilizando funcionalidades especificas del lenguaje de programación.

Contar con un número mayor de nodos en la red puede causar un mal funcionamiento del controlador debido a que su procesamiento aumenta. Dependiendo del tipo de evaluación que se realice y el número de nodos se recomienda discreción al momento de añadir nuevas funcionalidades o elevar el número de dispositivos en la red.

Al realizar pruebas de consumo de los dispositivos, se recomienda tener cuidado al momento de añadir esta herramienta al programa debido a que dependiendo de su ubicación puede causar o no un mal funcionamiento de los dispositivos provocando una recolección de datos errónea y por ende un resultado invalido.

REFERENCIA BIBLIOGRÁFICA

Alassaf, N., & Gutub, A. (2019). Simulating light-weight-cryptography implementation for IoT healthcare data security applications. *International Journal of E-Health and Medical Communications*, *10*(4), 1–15. https://doi.org/10.4018/IJEHMC.2019100101

Ali, H. (2012). *A Performance Evaluation of RPL in Contiki: A Cooja Simulation based study*. Retrieved from http://www.bth.se/tek/aps/mbo.nsf

Alves, R. C. A., Oliveira, D. A. G., Gustavo, N. S., & Margi, C. B. (2017). IT-SDN: Improved architecture for SDWSN. *XXXV Brazilian Symposium on Computer Networks and Distributed Systems.*

Amulothu, V. S. N., Huralikupp, N. J., Kapur, A., & Shukla, V. (2012). *END - TO - END ENCRYPTION IN A SOFTWARE DEFINED NETWORK*. 24–29.

Baddeley, M., Nejabati, R., Oikonomou, G., Sooriyabandara, M., & Simeonidou, D. (2018). Evolving SDN for Low-Power IoT Networks. *2018 4th IEEE Conference on Network Softwarization and Workshops, NetSoft 2018*, (NetSoft), 212–216. https://doi.org/10.1109/NETSOFT.2018.8460125

Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., & Wingers, L. (2015). Simon and Speck: Block Ciphers for the Internet of Things. *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*, (July), 1–6. Retrieved from http://dl.acm.org/citation.cfm?doid=2744769.2747946

Blandón Gómez, D. (2013). Openflow: El protocolo del futuro. *Páginas: Revista Académica e Institucional de La UCPR*, (93), 6.

Cama, A., & Cama, D. (2012). Las redes de sensores inalámbricos y el internet de las cosas. *Inge-Cuc*, *8*(1), 162–172.

Cañete, J. (2015). Análisis de seguridad del protocolo 6LoWPAN. *Angel Jesus Cañete Valverde*, 43, □□□□ □□□□□.

Carrasco, L. (2019). *Estudio Sobre La Aplicación De Las Redes Definidas Por Software a La Internet De Las Cosas. Diseño De Un Escenario De Pruebas*. 53. Retrieved from https://bit.ly/2PF1TBv

Casado, L., & Tsigas, P. (2009). ContikiSec: A secure network layer for wireless sensor networks under the Contiki Operating System. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5838 LNCS*, 133–147. https://doi.org/10.1007/978-3-642-04766-4_10

Castillo Merchán, H. A. (2016). *Análisis de la gestión de seguridad y fallos en internet de las cosas, usando el estándar 6lowpan*. Retrieved from http://dspace.udla.edu.ec/handle/33000/6952

Centeno, A. G., Manuel, C., Vergel, R., & Calderón, C. A. (2014). Controladores SDN, elementos para su selección y evaluación. *Revista Telem@tica*, *13*(3), 10–20.

Chen, Y., Jia, H., Huang, K., Lan, J., & Yan, X. (2016). A Secure Network Coding Based on Broadcast Encryption in SDN. *Mathematical Problems in Engineering*, *2016*. https://doi.org/10.1155/2016/7145138

Cheng, L., & Ye, Q. (2007). Wireless Sensor Networks (WSNs). *Encyclopedia of Wireless and Mobile Communications*, 1–20. https://doi.org/10.1201/noe1420043266.ch172

Contiki-NG. (n.d.). Home · contiki-ng/contiki-ng Wiki · GitHub. Retrieved October 13, 2021, from https://github.com/contiki-ng/contiki-ng/wiki

Contiki. (2014). *Contiki: The Open Source Operating System for the Internet of Things*. Retrieved from http://www.contiki-os.org/

CryptoLUX. (2020). FELICS. Retrieved August 11, 2021, from https://www.cryptolux.org/index.php/FELICS

Dhanda, S. S., Singh, B., & Jindal, P. (2020). Lightweight Cryptography: A Solution to Secure IoT. *Wireless Personal Communications*, *112*(3), 1947–1980. https://doi.org/10.1007/s11277-020-07134-3

Eissa, H. A., Bozed, K. A., & Younis, H. (2019). Software Defined Networking. *19th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering, STA 2019*, 620–625. https://doi.org/10.1109/STA.2019.8717234

Farzaneh, A., List, E., Lucks, S., & Wenzel, J. (2013). Cryptanalysis of the SPECK Family of Block Ciphers. *IACR Cryptology EPrint Archive*. Retrieved from https://eprint.iacr.org/2013/568.pdf

Fernández, J. A. P., Villalba, L. J. G., & Kim, T. H. (2018). Software defined networks in wireless sensor architectures. *Entropy*, *20*(4), 1–23. https://doi.org/10.3390/e20040225

Galluccio, L., Milardo, S., Morabito, G., & Palazzo, S. (2015). SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks. *Proceedings - IEEE INFOCOM*, *26*, 513–521. https://doi.org/10.1109/INFOCOM.2015.7218418

García García, D. (2015). *Estudio de 6loWPAN para su aplicación a Internet de las Cosas 6LoWPAN study for application to the Internet of Things*. 211–218. Retrieved from https://riull.ull.es/xmlui/bitstream/handle/915/945/Estudio de 6loWPAN para su aplicacion a Internet de las Cosas.pdf?sequence=1

Gonzalez, C., Flauzac, O., & Nolot, F. (2018). Evolución y Contribución para el Internet de las Cosas por las emergentes Redes Definidas por Software. *Memorias de Congresos UTP*, *1*(1), 28–33. Retrieved from http://revistas.utp.ac.pa/index.php/memoutp/article/view/1842

Hanes, D., Salgueiro, G., Grossetete, P., Barton, R., & Henry, J. (2017). *IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things*. Indianapolis, IN 46240 USA All: Cisco Press.

Jiménes, L., & Berrueco, J. (2016). *Diseño e implementación de etapa de comunicación 6LoWPAN para redes de sensores inalámbricas*.

Khan, S., Ali, M., Sher, N., Asim, Y., Naeem, W., & Kamran, M. (2016). Software-Defined Networks (SDNs) and Internet of Things (IoTs): A Qualitative Prediction for 2020. *International Journal of Advanced Computer Science and Applications*, *7*(11), 385–404. https://doi.org/10.14569/ijacsa.2016.071151

Lama Sleem, & Raphaël Couturier. (n.d.). The original Speck64/96 cipher | Download Scientific Diagram. Retrieved January 11, 2022, from https://www.researchgate.net/figure/The-original-Speck64-96-cipher_fig12_344539102

Lee, J. S., Su, Y. W., & Shen, C. C. (2007). A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. *IECON Proceedings (Industrial Electronics Conference)*, (January 2014), 46–51. https://doi.org/10.1109/IECON.2007.4460126

LF Projects. (2019). Home - OpenDaylight. Retrieved November 17, 2020, from https://www.opendaylight.org/

Lobato, A. G. P. (2013). *Redes Definidas por Software Introdução*.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., … Turner, J. (2008). OpenFlow. *ACM SIGCOMM Computer Communication Review*, *38*(2), 69–74. https://doi.org/10.1145/1355734.1355746

Miguel, M. L. F., Jamhour, E., Pellenz, M. E., & Penna, M. C. (2018). SDN architecture for 6LoWPAN wireless sensor networks. *Sensors (Switzerland)*, *18*(11), 1–23. https://doi.org/10.3390/s18113738

Millán Tejedor, R. J. (2014). *SDN: el futuro de las redes inteligentes*. Retrieved from https://www.ramonmillan.com/tutoriales/sdnredesinteligentes.php

Mokrenko, O. (2015). *Energy management of a Wireless Sensor Network at application level*. 140. Retrieved from https://tel.archives-ouvertes.fr/tel-01285378

NetSim™. (n.d.). NetSim-Network Simulator & Emulator | NetSim Academic. Retrieved July 22, 2021, from https://www.tetcos.com/netsim-acad.html

Ojo, M., Adami, D., & Giordano, S. (2016). A SDN-IoT architecture with NFV implementation. *2016 IEEE Globecom Workshops, GC Wkshps 2016 - Proceedings*. https://doi.org/10.1109/GLOCOMW.2016.7848825

Open Network Foundation. (2020). Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions. Retrieved November 17, 2020, from https://opennetworking.org/onos/

Otto, C. (2017). Diez años usando smartphones: así nos ha cambiado la vida en esta década. *La Vanguardia*. Retrieved from https://www.lavanguardia.com/tecnologia/20170226/42274940927/diez-anos-smartphones-cambiado-vida.html

Pritchard, S. W., Hancke, G. P., & Abu-Mahfouz, A. M. (2018). Cryptography Methods for Software-Defined Wireless Sensor Networks. *IEEE International Symposium on Industrial Electronics*, *2018-June*, 1257–1262. https://doi.org/10.1109/ISIE.2018.8433630

Project Floodlight. (n.d.). Architecture - Floodlight Controller - Project Floodlight. Retrieved November 17, 2020, from https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview

Ranjbar, A., Komu, M., Salmela, P., & Aura, T. (2016). An SDN-based approach to enhance the end-to-end security: SSL/TLS case study. *Proceedings of the NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 281–288. https://doi.org/10.1109/NOMS.2016.7502823

Riot. (2016). RIOT - The friendly Operating System for the Internet of Things. Retrieved January 9, 2020, from http://riot-os.org/

Rose, K., Eldridge, S., & Chapin, L. (2015). *Para entender mejor los problemas y desafíos de un mundo más conectado*. Retrieved from https://www.internetsociety.org/iot/%0Ahttps://www.internetsociety.org/wp-content/uploads/2017/09/report-InternetOfThings-20160817-es-1.pdf

RYU SDN Project Team. (2016). *RYU SDN Framework*. 455. Retrieved from https://osrg.github.io/ryu/

Semushin, S. (2020). Contiki_EA_comparison: Comparison of different encryption algorithms integrated in Contiki OS. Retrieved July 27, 2021, from https://github.com/kezzyhko/Contiki_EA_comparison

Serrano, A. (2015). *TRABAJO FIN DE GRADO Redes Definidas por Software (SDN): OpenFlow*. Retrieved from https://riunet.upv.es/bitstream/handle/10251/62801/SERRANO - Redes Definidas por Software (SDN): OpenFlow.pdf?sequence=3

Shingledecker, R. (n.d.). Welcome to Tiny Core Linux. Retrieved January 9, 2020, from http://tinycorelinux.net/intro.html

Singh, S., Sharma, P. K., Moon, S. Y., & Park, J. H. (2017). Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. *Journal of Ambient Intelligence and Humanized Computing*, *0*(0), 1–18. https://doi.org/10.1007/s12652-017-0494-4

Smetana, G. M. (2012). *IPv4 e IPv6*. Retrieved from https://trustnet.com.mx/ipv4-e-ipv6/

Stephen Evanczuk. (2020, June 3). Fundamentos de seguridad de IoT, Parte 1: Uso de la criptografía. Retrieved January 5, 2021, from https://www.digikey.com/es/articles/iot-security-fundamentals-part-1-using-cryptography

Tanenbaum, A., & Wetherall, D. (2012). Comunicaciones y redes de computadoras. *Prentice Hall*, 816.

TinyOS Alliance. (2012). TinyOS Home Page. Retrieved January 9, 2020, from http://www.tinyos.net/

Triana, J. (2017). *Estado del Arte sobre SDN y NFV*.

Wu, J., Huang, Y., Kong, J., Tang, Q., & Huang, X. (2015). *A Study on the Dependability of Software Defined Networks*. https://doi.org/10.2991/meita-15.2015.58

ANEXOS

ANEXO A: Norma ISO/IEC/IEEE 29148

INTERNATIONAL STANDARD

ISO/IEC/ IEEE 29148

Second edition
2018-11

Systems and software engineering — Life cycle processes — Requirements engineering

Ingénierie des systèmes et du logiciel — Processus du cycle de vie — Ingénierie des exigences

Reference number
ISO/IEC/IEEE 29148:2018(E)

ANEXO B: INSTALACIÓN DEL SISTEMA OPERATIVO CONTIKI Y EL SIMULADOR COOJA

Instalación de Contiki

La forma más simple de instalar Contiki es en base a los repositorios de su página oficial en GitHub.

Para ello se necesita una máquina virtual que cuente con un sistema operativo basado en Linux como es Ubuntu 18.04 que funcione sobre el sistema de virtualización VMWARE.

Una vez se cuenta con Ubuntu ya instalado, se abre el terminal y se actualiza el sistema con los comandos: `sudo apt-get update && sudo apt-get upgrade.`

```
anthony@ubuntu:~$ sudo apt-get update && sudo apt-get upgrade
Des:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88,7 kB]
Obj:2 http://ppa.launchpad.net/openjdk-r/ppa/ubuntu bionic InRelease
Obj:3 http://us.archive.ubuntu.com/ubuntu bionic InRelease
Des:4 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease [88,7 kB]
Des:5 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease [74,6 kB]
Descargados 252 kB en 1s (180 kB/s)
Leyendo lista de paquetes... Hecho
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Calculando la actualización... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no
```

Luego, desde el mismo terminal se procede a la descarga de Contiki con su ambiente de pruebas. Para ello se inserta el siguiente comando:

```
git clone https://github.com/contiki-os/contiki.git
```

```
anthony@ubuntu:~$ git clone https://github.com/contiki-os/contiki.git
Clonando en 'contiki'...
remote: Enumerating objects: 100481, done.
remote: Total 100481 (delta 0), reused 0 (delta 0), pack-reused 100481
Recibiendo objetos: 100% (100481/100481), 71.63 MiB | 3.67 MiB/s, listo.
Resolviendo deltas: 100% (72483/72483), listo.
```

Al ejecutar el comando, empieza a clonar el repositorio de Contiki OS hasta que todas las carpetas y módulos se clonen en la carpeta raíz. Una vez que ha terminado de descargar si bien ya cuenta con el sistema operativo contiki, aun no se puede ejecutar el simulador de red dado que aún faltan algunos archivos. Para ello, se ingresa dentro de la carpeta descargada con el comando `cd Contiki` y se ejecuta el comando:

```
git submodule update --init --recursive
```

El comando sirve para descargar todas las subcarpetas vinculadas a otros repositorios pero que son parte del sistema.

```
anthony@ubuntu:~/contiki$ git submodule update --init --recursive
Submódulo 'cpu/cc26xx-cc13xx/lib/cc13xxware' (https://github.com/contiki-os/cc13
xxware.git) registrado para ruta 'cpu/cc26xx-cc13xx/lib/cc13xxware'
Submódulo 'cpu/cc26xx-cc13xx/lib/cc26xxware' (https://github.com/contiki-os/cc26
xxware.git) registrado para ruta 'cpu/cc26xx-cc13xx/lib/cc26xxware'
Submódulo 'platform/stm32nucleo-spirit1/stm32cube-lib' (https://github.com/STcla
b/stm32nucleo-spirit1-lib) registrado para ruta 'platform/stm32nucleo-spirit1/st
m32cube-lib'
Submódulo 'tools/cc2538-bsl' (https://github.com/JelmerT/cc2538-bsl.git) registr
ado para ruta 'tools/cc2538-bsl'
Submódulo 'tools/mspsim' (https://github.com/contiki-os/mspsim.git) registrado p
ara ruta 'tools/mspsim'
Submódulo 'tools/sensniff' (https://github.com/g-oikonomou/sensniff.git) registr
ado para ruta 'tools/sensniff'
Clonando en '/home/anthony/contiki/cpu/cc26xx-cc13xx/lib/cc13xxware'...
Clonando en '/home/anthony/contiki/cpu/cc26xx-cc13xx/lib/cc26xxware'...
Clonando en '/home/anthony/contiki/platform/stm32nucleo-spirit1/stm32cube-lib'..
.
Clonando en '/home/anthony/contiki/tools/cc2538-bsl'...
Clonando en '/home/anthony/contiki/tools/mspsim'...
Clonando en '/home/anthony/contiki/tools/sensniff'...
Ruta de submódulo 'cpu/cc26xx-cc13xx/lib/cc13xxware': check out realizado a '0f4
4f949b4a7862ae3273739279df6297a4a4e45'
Ruta de submódulo 'cpu/cc26xx-cc13xx/lib/cc26xxware': check out realizado a 'e81
```

Una vez que el proceso termine, se puede ejecutar el ambiente de pruebas Cooja para la simulación de motas de red.

ANEXO C-1: PROGRAMA FUENTE DEL NODO EMISOR

```
 57 #include "../cryptolibs/crypto.h"
 58 #include "constants.h"
 .
 .
 .
146 // Generating random data
147 void generate_random_data(unsigned char* result, int size) {
148         static const char charset[] = MESSAGE_CHARSET;
149     int i;
150     for (i = 0; i < size; i++) {
151         int key = rand() % (sizeof charset - 1);
152         result[i] = charset[key];
153     }
154 }
155
156
157 static void
158 tcpip_handler(void)
159 {
160         char *str;
161         if(uip_newdata()) {
162                 str = uip_appdata;
163                 str[uip_datalen()]= '\0';
164                 reply++;
165                 printf("DATA recv from controller '%s' size:%d (s:%d,
    r:%d)\n", str, strlen(str), seq_id, reply);
166                 leds_toggle(LEDS_RED);
167
.
.
.   static void
179 send_packet(void *ptr)
180 {
181         char buf[MAX_PAYLOAD_LEN], buf_send[MAX_PAYLOAD_LEN];
182         //char *puntero_buf, *puntero_ecrypt, *puntero_desencryp;
183         unsigned char buf_trans[MAX_PAYLOAD_LEN],
184 packet_plain[MAX_PAYLOAD_LEN], desencryp[MAX_PAYLOAD_LEN];
185
186 #ifdef SERVER_REPLY
187         uint8_t num_used = 0;
188         uip_ds6_nbr_t *nbr;
189
190         nbr = nbr_table_head(ds6_neighbors);
191         while(nbr != NULL) {
192                 nbr = nbr_table_next(ds6_neighbors, nbr);
193                 num_used++;
194         }
195
196         if(seq_id > 0) {
197                 ANNOTATE("#A r=%d/%d,color=%s,n=%d %d\n", reply,
    seq_id, reply == seq_id ? "GREEN" : "RED", uip_ds6_route_num_routes(),
198 num_used);
199         }
```

```
200 #endif /* SERVER_REPLY */
201
202         seq_id++;
203         uip_ip6addr(&server_ipaddr, UIP_DS6_DEFAULT_PREFIX, 0, 0, 0,
204 0x200, 0, 0, destnode);
205
206         sprintf(buf, "         From:%d to:%d - Hello: %02d:",
    node_last_octect, server_ipaddr.u8[sizeof(server_ipaddr.u8) - 1],
207 seq_id);
208         int i;

          memcpy(buf_trans, buf, strlen(buf));
209
210         for (i = 0; i < strlen(buf); i += BLOCK_LENGTH/8) {
211         encrypt(buf_trans + i, packet_plain + i);
212         }
213
214         printf("Mensaje Original:%s size:%d\n",buf_trans,
215 strlen(buf_trans));
216         printf("Mensaje Cifrado:");
217         int j;
          for (j = 0; j < strlen(packet_plain); j++) {
218                 printf("%02x", packet_plain[j]);
219                 }
220         printf("  size: %d \n", strlen(packet_plain));
221
222         memcpy(buf_send, packet_plain, strlen(packet_plain));
223
224         uip_udp_packet_sendto(client_conn, buf_send, strlen(buf_send),
225                         &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
226
227 }
228
229 /*---------------------------------------------------------------------
230 -------*/
231 PROCESS_THREAD(sender_process, ev, data) {
232         PROCESS_BEGIN();
233
234         // prepare the key
235         PRINTF("Key generation started");
236         setKey((unsigned char *)(KEY), KEY_LENGTH);
237         PRINTF("Key generated");
238         PROCESS_END();
239 }
.
.
.
```

ANEXO C-2: PROGRAMA FUENTE DEL RECEPTOR (CONTROLADOR)

```
52 #include "examples/cryptolibs/crypto.h"
53 #include "../constants.h"
54 //#include "powertrace.h"
55
56 #include <stdio.h>
57 #include <stdlib.h>
58 #include <string.h>
59 #include <ctype.h>
60
.
.
108 static void
109 tcpip_handler(void)
110 {
111         char *appdata;
112         char f='F';
113         char *result;
114         if(uip_newdata()) {
115                 appdata = (char *)uip_appdata;
116                 appdata[uip_datalen()] = 0;
117
118                 PRINTF("DATA recv '%s' from ", appdata);
119
120                 unsigned char block_decrypted[MAX_PAYLOAD_LEN + 1];
121                 int i;
122                 for (i = 0; i < MAX_PAYLOAD_LEN; i += BLOCK_LENGTH/8) {
123                 //for (i = 0; i < 24; i += BLOCK_LENGTH/8) {
124                 decrypt(appdata + i, block_decrypted + i);
125                 }
126                 block_decrypted[MAX_PAYLOAD_LEN]=0;
127                 result=strchr(block_decrypted, f);
128
129                 printf("\nData Desencryp:%s", result); //quitar esta
    linea en caso de no ser necesaria
130                 PRINTF("%d",
131                 UIP_IP_BUF->srcipaddr.u8[sizeof(UIP_IP_BUF-
    >srcipaddr.u8) - 1]);
132                 PRINTF("\n");
133 #if SERVER_REPLY
134                 PRINTF("DATA sending reply\n");
135                 uip_ipaddr_copy(&server_conn->ripaddr, &UIP_IP_BUF-
    >srcipaddr);
136                 uip_udp_packet_send(server_conn, "Mensaje recibido",
    sizeof("Mensaje recibido"));
137                 //uip_udp_packet_send(server_conn, block_decrypted,
138 sizeof(block_decrypted));
                 uip_create_unspecified(&server_conn->ripaddr);
139 #endif
140         }
141 }
142
143 /*-----------------------------------------------------------------------
.   ------*/
```

```
.
.
232 PROCESS_THREAD(reciever_process, ev, data) {
233         PROCESS_BEGIN();
234
235         // prepare the key
236         PRINTF("Key generation started");
237         setKey((unsigned char *)(KEY), KEY_LENGTH);
238         PRINTF("Key generated");
239
240
241 PROCESS_END();
242 }
```

ANEXO D-1: DATOS OBTENIDOS DE LOS NODOS PREVIA IMPLEMETNACION DE LA ENCRIPTACION

Nodo 2

	ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
0	336577	18341436	58585	18586995
1	353822	19307233	60943	19567673
2	380727	20263366	66462	20545182
3	397602	21229531	69193	21525488
4	411497	22198676	70885	22506826
5	429198	23164017	73434	23487314
6	443233	24133033	75136	24468659
7	463273	25096019	78402	25448410
8	487215	26055117	83052	26426793
9	502758	27022617	85465	27407417
10	522856	27985557	89198	28386718

Nodo 3

	ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
0	189106	18488907	24270	18399440
1	199984	19461071	25678	19381072
2	214071	20430022	28004	20361782
3	224501	21402632	29223	21343601
4	236113	22374059	31040	22324822
5	246771	23346444	32369	23306533
6	259104	24317149	34419	24287517
7	273048	25286244	36621	25268349
8	284641	26257691	38200	26249808
9	296034	27229340	39864	27231181
10	307175	28201238	41577	28212500

Nodo 4

	ALL_CPU	ALL_LPM	ALL_TX	ALL_RX

0	332285	18345728	57226	18421155
1	361924	19299131	63427	19397977
2	377446	20266647	65511	20378926
3	399986	21227147	70009	21357456
4	429240	22180932	76253	22334232
5	458491	23134724	82423	23311087
6	489058	24087195	88870	24287668
7	511170	25048122	93112	25266460
8	532580	26009752	96883	26245722
9	561599	26963776	103100	27222534
10	587222	27921191	108608	28200057

Nodo 5

	ALL_CPU	**ALL_LPM**	**ALL_TX**	**ALL_RX**
0	368103	18309910	54860	18590703
1	394306	19266749	59274	19569318
2	415541	20228552	62301	20549322
3	436876	21190257	65556	21529099
4	460502	22149670	69473	22508213
5	486805	23106410	74007	23486715
6	510344	24065909	77864	24465883
7	528051	25031241	79824	25446957
8	546916	25995416	82686	26427130
9	564494	26960881	84851	27407999
10	589808	27918605	88809	28387068

Nodo 6

	ALL_CPU	**ALL_LPM**	**ALL_TX**	**ALL_RX**
0	222014	18455999	25869	18619777
1	237827	19423228	28057	19600624
2	250169	20393924	29408	20582304
3	264102	21363030	31117	21563632
4	279135	22331037	33183	22544603

5	293902	23299313	35095	23525728
6	312214	24264039	38313	24505543
7	327371	25231922	40361	25486529
8	343048	26199284	42505	26467422
9	356128	27169247	44267	27448699
10	367712	28140701	45401	28430599

Nodo 7

	ALL_CPU	**ALL_LPM**	**ALL_TX**	**ALL_RX**
0	178175	18499838	20627	18386191
1	188472	19472583	21757	19368100
2	199163	20444930	22979	20349913
3	210366	21416766	24257	21331672
4	220290	22389882	25415	22313553
5	230590	23362625	26461	23295547
6	242661	24333592	28049	24276994
7	256337	25302956	30069	25258010
8	271510	26270822	32807	26238307
9	289421	27235953	36797	27217355
10	300779	28207634	38076	28199111

Nodo 8

	ALL_CPU	**ALL_LPM**	**ALL_TX**	**ALL_RX**
0	232495	18445518	29842	18236394
1	244327	19416728	30874	19218402
2	260070	20384023	32912	20199393
3	273186	21353946	34659	21180682
4	283761	22326411	35453	22162925
5	297208	23296007	36946	23144470
6	309397	24266856	38522	24125928
7	322018	25237274	39890	25107596
8	334429	26207903	41164	26089358
9	350366	27175009	43699	27069860

10	365723	28142690	45953	28050639

Nodo 9

	ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
0	173529	18504484	20024	18625641
1	184362	19476693	21158	19607547
2	195346	20448747	22292	20589449
3	205197	21421935	23600	21571177
4	215723	22394449	25126	22552686
5	226456	23366759	26385	23534463
6	235998	24340255	27284	24516599
7	245678	25313615	28275	25498647
8	256330	26286003	29346	26480613
9	266587	27258788	30412	27462585
10	278209	28230204	31974	28444061

ANEXO D-2: DATOS OBTENIDOS DE LOS NODOS POSTERIOR IMPLEMETNACION DE LA ENCRIPTACION

Nodo 2

	ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
0	361788	18316225	62227	18583324
1	394326	19266729	68903	19559676
2	418082	20226011	73072	20538534
3	439010	21188123	76346	21518290
4	469967	22140375	82651	22495179
5	501359	23091856	88908	23471777
6	533995	24042258	95453	24448260
7	555028	25004264	98522	25428228
8	573054	25969279	100778	26409000
9	599067	26926308	105581	27387231
10	625462	27882951	110573	28365265

Nodo 3

	ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
0	212519	18465494	26525	18396910
1	229827	19431228	29364	19377111
2	246872	20397221	32066	20357444
3	259513	21367619	33428	21339120
4	274158	22336014	34877	22320708
5	286294	23306921	35934	23302692
6	301336	24274917	37968	24283692
7	315056	25244236	39482	25265213
8	326800	26215532	40508	26247225
9	341252	27184123	42061	27228709
10	353991	28154422	43154	28210652

Nodo 4

	ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
0	361777	18316236	61427	18416715
1	377407	19283648	63099	19398079
2	399296	20244797	66913	20377295
3	424491	21202641	71463	21355768
4	443282	22166890	73717	22336549
5	461777	23131438	75979	23317324
6	480788	24095465	78480	24297858
7	508516	25050776	83639	25275727
8	538962	26003370	89905	26252484
9	562167	26963208	93770	27231653
10	584718	27923695	97509	28210945

Nodo 5

	ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
0	383289	18294724	55926	18589659
1	410977	19250078	60558	19568059

2	435141	20208953	63860	20547791
3	463278	21163854	68229	21526454
4	489178	22120994	72605	22505108
5	516565	23076650	77185	23483556
6	544101	24032152	81515	24462257
7	564309	24994983	83623	25443182
8	590732	25951600	87859	26421974
9	617901	26907474	92041	27400824
10	644990	27863423	96124	28379772

Nodo 6

	ALL_CPU	**ALL_LPM**	**ALL_TX**	**ALL_RX**
0	244543	18433470	26443	18619222
1	264506	19396550	28955	19599748
2	280832	20363260	31337	20580397
3	296792	21330340	33292	21561479
4	316044	22294128	36525	22541272
5	329117	23264095	37522	23523312
6	344496	24231756	39108	24504765
7	358097	25201195	40895	25486013
8	371376	26170956	42007	26467939
9	385754	27139618	43504	27449477
10	402102	28106310	45399	28430619

Nodo 7

	ALL_CPU	**ALL_LPM**	**ALL_TX**	**ALL_RX**
0	202805	18475208	24255	18382301
1	215827	19445228	25549	19364045
2	228647	20415446	26633	20345996
3	241788	21385345	27848	21327819
4	254461	22355711	29057	22309650
5	266647	23326568	30286	23291461
6	283421	24292832	32608	24272172

7	302016	25257276	36114	25251699
8	315747	26226586	37577	26233273
9	328043	27197332	38703	27215186
10	343680	28164733	40521	28196398

Nodo 8

	ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
0	251554	18426459	29403	18236577
1	269337	19391719	31805	19217212
2	282065	20362028	32970	20199080
3	296073	21331059	34290	21180798
4	310497	22299676	35599	22162529
5	328695	23264520	37687	23143477
6	350379	24225874	41355	24122840
7	369570	25189722	44027	25103205
8	388436	26153897	46765	26083502
9	404584	27120791	48589	27064717
10	421095	28087318	50586	28045754

Nodo 9

	ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
0	188799	18489214	19977	18625691
1	202019	19459036	21169	19607539
2	214339	20429754	22252	20589491
3	225864	21401268	23310	21571473
4	238710	22371463	24549	22553271
5	251195	23342020	26046	23534814
6	264583	24311670	27492	24516404
7	276310	25282983	28578	25498358
8	288728	26253605	29494	26480480
9	300647	27224727	30559	27462454
10	314466	28193947	31901	28444147

ANEXO E-1: CALCULO DE ENERGIA EN LOS NODOS PREVIA IMPLEMENTACION DE LA ENCRIPTACION

Nodo 2

	CPU (mW)	LPM (mW)	TX (mW)	RX (mW)	TOTAL (mW)
1	0.003859355	0.000196492	0.006188599	1.845554911	1.855799356
2	0.00602122	0.000194526	0.01448468	1.839591115	1.860291541
3	0.00377655	0.000196567	0.007167542	1.844854838	1.855995497
4	0.003109639	0.000197173	0.004440674	1.846796977	1.854544463
5	0.003961405	0.000196399	0.00668988	1.845197347	1.856045032
6	0.003140971	0.000197147	0.004466919	1.84681015	1.854615187
7	0.004484863	0.00019592	0.008571655	1.843810374	1.857062813
8	0.005358114	0.000195129	0.012203979	1.841235911	1.858993133
9	0.003478455	0.000196838	0.006332947	1.845453288	1.855461528
10	0.004497843	0.000195911	0.009797302	1.842963511	1.857454567

Nodo 3

	CPU (mW)	LPM (mW)	TX (mW)	RX (mW)	TOTAL (mW)
1	0.00243445	0.00019779	0.00369531	1.84735026	1.85367781
2	0.00315261	0.00019713	0.00610461	1.84561513	1.85506949
3	0.00233419	0.00019788	0.00319928	1.84770218	1.85343352
4	0.00259871	0.00019764	0.00476874	1.84657679	1.85414188
5	0.00238521	0.00019783	0.00348798	1.84749893	1.85356995
6	0.00276007	0.00019749	0.00538025	1.84613078	1.85446859
7	0.00312061	0.00019716	0.00577917	1.84584473	1.85494167
8	0.00259446	0.00019764	0.0041441	1.84702469	1.8539609
9	0.0025497	0.00019768	0.00436719	1.84686284	1.85397742
10	0.00249331	0.00019773	0.00449579	1.84676122	1.85394805

Nodo 4

	CPU (mW)	LPM (mW)	TX (mW)	RX (mW)	TOTAL (mW)

1	0.00663308	0.00019397	0.0162746	1.83829824	1.86139988
2	0.00347375	0.00019684	0.00546948	1.84606491	1.85520499
3	0.00504435	0.00019541	0.01180505	1.84151255	1.85855737
4	0.00654692	0.00019405	0.01638745	1.83821167	1.86134008
5	0.00654624	0.00019405	0.01619324	1.83836034	1.86129387
6	0.00684076	0.00019378	0.01692023	1.8378447	1.86179946
7	0.00494857	0.0001955	0.01113318	1.84200562	1.85828286
8	0.00479146	0.00019564	0.00989703	1.84289012	1.85777426
9	0.00649432	0.0001941	0.01631659	1.83827942	1.86128443
10	0.00573431	0.00019479	0.01445581	1.83961746	1.86000237

Nodo 5

	CPU (mW)	**LPM (mW)**	**TX (mW)**	**RX (mW)**	**TOTAL (mW)**
1	0.00586412	0.00019467	0.01158459	1.84167252	1.8593159
2	0.0047523	0.00019568	0.0079444	1.8442865	1.85717887
3	0.00477468	0.00019566	0.00854279	1.8438593	1.85737243
4	0.00528739	0.00019519	0.01028021	1.84261159	1.85837439
5	0.00588649	0.00019465	0.01189954	1.84145986	1.85944054
6	0.00526792	0.00019521	0.01012274	1.84271322	1.85829909
7	0.00396275	0.0001964	0.00514404	1.84630015	1.85560334
8	0.0042219	0.00019616	0.00751135	1.84460454	1.85653396
9	0.00393388	0.00019642	0.00568207	1.84591436	1.85572673
10	0.00566516	0.00019485	0.01038782	1.84252691	1.85877473

Nodo 6

	CPU (mW)	**LPM (mW)**	**TX (mW)**	**RX (mW)**	**TOTAL (mW)**
1	0.00353888	0.00019678	0.00574243	1.84587296	1.85535105
2	0.00276208	0.00019749	0.00354572	1.84744059	1.85394588
3	0.00311814	0.00019717	0.00448529	1.84677816	1.85457876
4	0.00336432	0.00019694	0.00542224	1.84610631	1.85508981
5	0.00330479	0.000197	0.00501807	1.84639613	1.85491598

6	0.00409814	0.00019627	0.00844568	1.84393082	1.85667091
7	0.00339207	0.00019692	0.005375	1.84613454	1.85509853
8	0.00350844	0.00019681	0.00562695	1.84595952	1.85529173
9	0.00292725	0.00019734	0.00462439	1.84668218	1.85443115
10	0.00259245	0.00019764	0.0029762	1.84785461	1.8536209

Nodo 7

	CPU (mW)	**LPM (mW)**	**TX (mW)**	**RX (mW)**	**TOTAL (mW)**
1	0.00230442	0.00019791	0.0029657	1.84787155	1.85333958
2	0.0023926	0.00019782	0.00320715	1.84769089	1.85348846
3	0.00250718	0.00019772	0.00335413	1.84758926	1.85364829
4	0.00222095	0.00019798	0.00303918	1.84781886	1.85327697
5	0.00230509	0.00019791	0.00274524	1.84803151	1.85327975
6	0.00270144	0.00019754	0.00416772	1.84700211	1.85406881
7	0.00306063	0.00019722	0.00530151	1.846191	1.85475036
8	0.00339565	0.00019691	0.00718591	1.8448379	1.85561638
9	0.0040084	0.00019636	0.0104718	1.84248739	1.85716395
10	0.00254187	0.00019769	0.00335675	1.84758362	1.85367993

Nodo 8

	CPU (mW)	**LPM (mW)**	**TX (mW)**	**RX (mW)**	**TOTAL (mW)**
1	0.00264795	0.00019759	0.0027085	1.84805786	1.8536119
2	0.00352321	0.0001968	0.00534875	1.84614395	1.85521272
3	0.0029353	0.00019733	0.00458502	1.84670476	1.85442242
4	0.00236664	0.00019785	0.00208386	1.84850011	1.85314846
5	0.00300938	0.00019726	0.0039184	1.84718653	1.85431157
6	0.00272784	0.00019752	0.00413623	1.84702281	1.8540844
7	0.00282452	0.00019743	0.00359033	1.84741801	1.8540303
8	0.00277753	0.00019747	0.00334363	1.84759491	1.85391354
9	0.00356663	0.00019676	0.00665314	1.84522369	1.85564022
10	0.00343683	0.00019688	0.00591565	1.84574498	1.85529434

Nodo 9

	CPU (mW)	LPM (mW)	TX (mW)	RX (mW)	TOTAL (mW)
1	0.00242438	0.0001978	0.0029762	1.84786591	1.85346428
2	0.00245817	0.00019776	0.0029762	1.84785838	1.85349051
3	0.00220461	0.000198	0.00343286	1.84753092	1.85336639
4	0.00235567	0.00019786	0.004005	1.84711878	1.85367732
5	0.002402	0.00019782	0.00330426	1.84762314	1.85352721
6	0.00213546	0.00019806	0.00235944	1.84829875	1.8529917
7	0.00216634	0.00019803	0.00260089	1.84813314	1.8530984
8	0.00238387	0.00019783	0.00281085	1.84797882	1.85337138
9	0.00229547	0.00019791	0.00279773	1.84799011	1.85328123
10	0.00260095	0.00019764	0.00409949	1.84705668	1.85395476

ANEXO E-2: CALCULO DE ENERGIA EN LOS NODOS POSTERIOR IMPLEMENTACION DE LA ENCRIPTACION

Nodo 2

	CPU (mW)	LPM (mW)	TX (mW)	RX (mW)	Total (mW)
1	0.00728186	0.000193381	0.01752124	1.837413737	1.862410218
2	0.005316488	0.000195166	0.010941589	1.842129822	1.858583065
3	0.004683594	0.000195742	0.008592651	1.843819784	1.857291771
4	0.00692804	0.000193736	0.016547546	1.838424327	1.862093649
5	0.007025391	0.000193579	0.01642157	1.837876689	1.861517228
6	0.007303792	0.00019336	0.017177429	1.837660268	1.862334849
7	0.004707092	0.000195721	0.008054626	1.84421875	1.857176189
8	0.004034139	0.000196333	0.005920898	1.845731812	1.855883182
9	0.005821594	0.000194708	0.01260553	1.84094986	1.859571692
10	0.005907084	0.00019463	0.013101563	1.840579122	1.859782398

Nodo 3

	CPU (mW)	LPM (mW)	TX (mW)	RX (mW)	TOTAL (mW)
1	0.00387345	0.00019648	0.00745099	1.84465724	1.85617816
2	0.0038146	0.00019653	0.00709143	1.84490565	1.85600821
3	0.002829	0.00019743	0.00357458	1.84743306	1.85403408
4	0.00327749	0.00019702	0.00380292	1.84726746	1.85454488
5	0.00271598	0.00019753	0.00277411	1.8480127	1.85370032
6	0.00336633	0.00019694	0.00533826	1.84616089	1.85506242
7	0.00307048	0.00019721	0.00397351	1.84714137	1.85438256
8	0.00262826	0.00019761	0.00269275	1.84806539	1.853584
9	0.00323429	0.00019706	0.00407587	1.84707174	1.85457896
10	0.00285093	0.00019741	0.00286859	1.84793554	1.85385247

Nodo 4

	CPU (mW)	LPM (mW)	TX (mW)	RX (mW)	TOTAL (mW)
1	0.00349792	0.00019682	0.00438818	1.84684591	1.85492884
2	0.00489866	0.00019555	0.01000989	1.84280355	1.85790764
3	0.00563853	0.00019487	0.01194153	1.84140528	1.85918022
4	0.00420534	0.00019618	0.00591565	1.84574875	1.85606592
5	0.0041391	0.00019624	0.00593665	1.84573746	1.85600944
6	0.00425458	0.00019613	0.0065639	1.84528392	1.85629853
7	0.0062054	0.00019436	0.01353986	1.84026861	1.86020822
8	0.00681368	0.00019381	0.01644519	1.83817591	1.86162859
9	0.00519318	0.00019528	0.01014374	1.8427151	1.85824729
10	0.00504681	0.00019541	0.00981305	1.84294657	1.85800185

Nodo 5

	CPU (mW)	LPM (mW)	TX (mW)	RX (mW)	TOTAL (mW)
1	0.00619645	0.00019437	0.01215674	1.8412679	1.85981546
2	0.0054078	0.00019508	0.00866614	1.84377462	1.85804364
3	0.00629694	0.00019428	0.01146649	1.84176285	1.85972055
4	0.00579631	0.00019473	0.01148486	1.84174591	1.85922181

5	0.00612909	0.00019443	0.01202026	1.84135824	1.85970202
6	0.00616243	0.0001944	0.01136414	1.84183436	1.85955533
7	0.00452246	0.00019589	0.00553247	1.84601974	1.85627056
8	0.00591335	0.00019462	0.01111743	1.84200562	1.85923102
9	0.0060803	0.00019447	0.01097571	1.84211477	1.85936525
10	0.0060624	0.00019449	0.01071588	1.84229919	1.85927196

Nodo 6

	CPU (mW)	**LPM (mW)**	**TX (mW)**	**RX (mW)**	**TOTAL (mW)**
1	0.00446763	0.00019594	0.00659277	1.84526886	1.8565252
2	0.00365369	0.00019668	0.00625159	1.84550034	1.85560229
3	0.00357178	0.00019675	0.00513092	1.84631521	1.85521466
4	0.00430851	0.00019608	0.00848505	1.84388941	1.85687906
5	0.00292568	0.00019734	0.00261664	1.84811808	1.85385774
6	0.00344175	0.00019687	0.00416248	1.8470134	1.8548145
7	0.00304384	0.00019723	0.00469	1.8466276	1.85455868
8	0.00297178	0.0001973	0.00291846	1.84790354	1.85399108
9	0.00321773	0.00019707	0.00392889	1.84717336	1.85451706
10	0.00365861	0.00019667	0.00497345	1.84642812	1.85525685

Nodo 7

	CPU (mW)	**LPM (mW)**	**TX (mW)**	**RX (mW)**	**TOTAL (mW)**
1	0.00291427	0.00019735	0.00339612	1.84756104	1.85406877
2	0.00286906	0.00019739	0.00284497	1.84795059	1.85386201
3	0.0029409	0.00019733	0.00318878	1.84770971	1.85403671
4	0.00283616	0.00019742	0.00317303	1.84772476	1.85393138
5	0.00272717	0.00019752	0.00322552	1.84768712	1.85383734
6	0.00375395	0.00019659	0.00609412	1.84561701	1.85566166
7	0.00416148	0.00019622	0.00920154	1.84338882	1.85694806
8	0.00307294	0.00019721	0.00383966	1.84724111	1.85435091
9	0.00275179	0.0001975	0.0029552	1.84787908	1.85378357

10	0.00349949	0.00019682	0.00477136	1.84655986	1.85502753

Nodo 8

	CPU (mW)	**LPM (mW)**	**TX (mW)**	**RX (mW)**	**TOTAL (mW)**
1	0.00397976	0.00019638	0.00630408	1.84547399	1.85595421
2	0.00284847	0.00019741	0.00305756	1.84779439	1.85389783
3	0.00313493	0.00019715	0.00346436	1.84751211	1.85430854
4	0.00322803	0.00019707	0.00343549	1.84753657	1.85439715
5	0.00407263	0.0001963	0.00547998	1.84606303	1.85581194
6	0.00485278	0.00019559	0.00962671	1.84308019	1.85775527
7	0.00429486	0.0001961	0.0070127	1.84496587	1.85646952
8	0.00422213	0.00019616	0.00718591	1.8448379	1.8564421
9	0.00361385	0.00019672	0.00478711	1.8465655	1.85516318
10	0.00369509	0.00019664	0.00524115	1.84623052	1.8553634

Nodo 9

	CPU (mW)	**LPM (mW)**	**TX (mW)**	**RX (mW)**	**TOTAL (mW)**
1	0.00295858	0.00019731	0.00312842	1.84775675	1.85404106
2	0.00275716	0.00019749	0.00284235	1.84795247	1.85374947
3	0.00257924	0.00019766	0.00277673	1.84800893	1.85356256
4	0.00287488	0.00019739	0.00325177	1.84766266	1.85398669
5	0.00279409	0.00019746	0.00392889	1.84718277	1.85410321
6	0.00299618	0.00019728	0.00379504	1.84727122	1.85425971
7	0.00262445	0.00019761	0.00285022	1.84795624	1.85362852
8	0.00277909	0.00019747	0.00240405	1.8482724	1.85365302
9	0.00266742	0.00019758	0.0027951	1.84799388	1.85365398
10	0.00309263	0.00019719	0.00352209	1.84746506	1.85427697