UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ÁREA TÉCNICA

MAGÍSTER EN CIENCIAS Y TECNOLOGÍAS DE LA COMPUTACIÓN

TRABAJO DE TITULACIÓN

Redes definidas por software para redes móviles en arquitectura de IoT

Autor: Montaño Blacio, Manuel Asdruval

Director: Torres Tandazo, Rommel Vicente

LOJA - ECUADOR

2021

2021

Aprobación del director del trabajo de titulación

Loja, 02, de Agosto, de 2021

Ph.D.
Rommel Vicente Torres Tandazo
Coordinador de programa de posgrados
Ciudad.-

De mi consideración:

El presente trabajo de titulación denominado: Redes definidas por software para redes móviles en arquitectura de IoT realizado por Manuel Asdruval Montaño Blacio, ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación del mismo. Así mismo, doy fe que dicho trabajo de titulación ha sido revisado por la herramienta antiplagio institucional.

Particular que comunico para los fines pertinentes.

Atentamente,

Rommel Vicente Torres Tandazo.
C.I: 1102885330

Declaración de autoría y cesión de derechos

"Yo, Manuel Asdruval Montaño Blacio, declaro y acepto en forma expresa lo siguiente:

- Ser autor del Trabajo de Titulación denominado: Redes definidas por software para redes móviles en arquitectura de IoT, del Programa de posgrado en ciencias y tecnología de la computación, específicamente de los contenidos comprendidos en: Introducción, Capítulo 1. Conceptualización del problema, Capítulo 2. Marco teórico, Capítulo 3. SDN-MANET y OLSR-MANET, Capítulo 4. Simulación, análisis y discusión de resultados, Conclusiones y Recomendaciones, siendo Rommel Vicente Torres Tandazo, director del presente trabajo; y, en tal virtud, eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones judiciales o administrativas, en relación a la propiedad intelectual. Además, ratifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo son de mi exclusiva responsabilidad.
- Que mi obra, producto de mis actividades académicas y de investigación, forma parte del patrimonio de la Universidad Técnica Particular de Loja, de conformidad con el artículo 20, literal j), de la Ley Orgánica de Educación Superior; y, artículo 91 del Estatuto Orgánico de la UTPL, que establece: "Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado que se realicen a través, o con el apoyo financiero, académico o institucional (operativo) de la Universidad".
- Autorizo a la Universidad Técnica Particular de Loja para que pueda hacer uso de mi obra con fines netamente académicos, ya sea de forma impresa, digital y/o electrónica o por cualquier medio conocido o por conocerse, sirviendo el presente instrumento como la fe de mi completo consentimiento; y, para que sea ingresada al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública, en cumplimiento del artículo 144 de la Ley Orgánica de Educación Superior.

Firma: .....

Autor: Manuel Asdruval Montaño Blacio

C.I.: 0706440674

Dedicatoria

Dedico este trabajo de Fin de Máster a Dios, por haberme permitido llegar a esta instancia, tan importante y anhelada de mi formación y vida profesional. Cómo no dedicar este trabajo a mis padres Manuel y Janeth quienes en todo momento me han dado ese empujón para alcanzar mis objetivos de vida. A mis docentes y amigos por su apoyo y conocimientos impartidos. Sobre todo, con mucho carisma y entusiasmo dedico este trabajo a mi esposa e hija. Jazlin Itzae quien con su llegada a este mundo me ha inspirado a proponerme nuevas metas de vida profesional.

Manuel Montaño

Agradecimiento

Quiero agradecer a todos los docentes que forman parte de la Maestría en Ciencias y Tecnologías de la Computación de la Universidad Técnica Particular de Loja (UTPL) por los conocimientos impartidos, he aprendido mucho de sus experticias en las diferentes ramas que contempla la maestría. Un agradecimiento muy especial y sentido a mi director de titulación Ing. Rommel Torres quien con su conocimiento y experiencia me ha guiado de forma exitosa para culminar mi investigación, así mismo, a los asesores Ing. Patricia Ludeña y Ing. Francisco Sandoval.

Agradezco a mis padres, hermanos sus familias, a mi esposa su familia, a mi hija, y amigos por siempre darme ese apoyo fundamental para alcanzar tan anhelada meta.

Manuel Montaño

Índice de Contenido

Carátula I

Aprobación del director del trabajo de titulación II

Declaración de autoría y cesión de derechos III

Dedicatoria V

Agradecimiento VI

Índice de Contenido VII

Resumen 1

Abstract 2

Introducción 3

Capítulo uno 5

Conceptualización del problema 5

1.1 Justificación 6

1.2 Objetivos 7

1.2.1 Objetivo general 7

1.2.2 Objetivos específicos 7

1.3 Metodología 7

1.4 Estructura del documento 7

Capítulo dos 9

Marco Teórico 9

2.1 Redes ad hoc inalámbricas 9

2.2 Redes definidas por software (SDN) 16

2.3 Internet de las Cosas (IoT) 19

2.3.1 Arquitectura IoT 20

2.3.2 Protocolos IoT 21

2.4 Simulador NS-3 23

Capítulo tres 24

Sdn-Manet y Olsr-Manet 24

3.1 Arquitecturas basadas en SDN para MANET 24

3.2 Arquitecturas basadas en SDN para IoT 25

3.3 Arquitectura SDN-MANET e IoT 26

***3.3.1 Estructura interna del nodo MANET* 29**

***3.3.2 Compatibilidad de SDN con el simulador NS-3* 30**

3.4 Protocolo de enrutamiento OLSR – MANET 31

***3.4.1 Mensaje de control OLSR* 33**

Capítulo cuatro 35

Simulación, análisis y discusión de resultados 35

4.1 Proceso de simulación 36

4.2 Definición de escenarios 38

4.3 Análisis de resultados 39

***4.3.1 Rendimiento* 39**

***4.3.2 Tasa de envío de paquetes de aplicación* 51**

***4.3.3 Tasa promedio de envío de paquetes* 53**

***4.3.4 Retardo promedio* 54**

***4.3.5 Tasa de retardo de paquetes* 55**

***4.3.6 Porcentaje de pérdida de paquetes* 56**

Conclusiones 59

Recomendaciones 61

Referencias 62

Apéndice 67

Índice de Tablas

Tabla 1 Controladores SDN 18
Tabla 2 Protocolos IoT.....22
Tabla 3 Archivos y códigos fuente OLSR 35
Tabla 4 Archivos y códigos fuente OpenFlow.....36
Tabla 5 Escenarios de simulación.....38

Índice de Figuras

Figura 1 Redes fijas e inalámbricas 10
Figura 2 Arquitectura MANET 11
Figura 3 Clasificación de protocolos de enrutamiento MANET 16
Figura 4 Arquitectura SDN....18
Figura 5 Arquitectura IoT....21
Figura 6 Arquitectura SDN-MANET 27
Figura 7 Estructura interna de los nodos SDN-MANET 29
Figura 8 Modificaciones a la pila de IP en el simulador ns3....31
Figura 9 Selección del nodo MPR. 32
Figura 10 Mensaje HELLO 33
Figura 11 Mensaje TC 34
Figura 12 Proceso de simulación en NS-3....37
Figura 13 Rendimiento con 20 nodos tráfico normal 41
Figura 14 Rendimiento con 20 nodos con carga de tráfico 42
Figura 15 Rendimiento con 20 nodos con tráfico PPBP....42
Figura 16 Rendimiento con 30 nodos tráfico normal 43
Figura 17 Rendimiento con 30 nodos con carga de tráfico 44
Figura 18 Rendimiento con 30 nodos con tráfico PPBP....44
Figura 19 Rendimiento con 40 nodos tráfico normal 45
Figura 20 Rendimiento con 40 nodos con carga de tráfico 46
Figura 21 Rendimiento con 40 nodos con tráfico PPBP....46
Figura 22 Rendimiento con 50 nodos tráfico normal 48
Figura 23 Rendimiento con 50 nodos con carga de tráfico 48
Figura 24 Rendimiento con 50 nodos con tráfico PPBP....49
Figura 25 Rendimiento con 100 nodos tráfico normal 50
Figura 26 Rendimiento con 100 nodos con carga de tráfico 50
Figura 27 Rendimiento con 100 nodos con tráfico PPBP....51
Figura 28 Tasa de envío de paquetes de aplicación con tráfico, normal, carga y PPBP 52
Figura 29 Tasa de envío de paquetes de datos con tráfico, normal, carga y PPBP 54
Figura 30 Retardo promedio con tráfico, normal, carga y PPBP....55
Figura 31 Tasa de retardo de paquetes con tráfico, normal, carga y PPBP....56
Figura 32 Pérdida de paquetes con tráfico, normal, carga y PPBP 58
Figura 33 Integración OpenFlow a NS-3....67

Resumen

La red MANET se compone de nodos móviles sin infraestructura, se autoorganiza y su comunicación es a través de múltiples saltos hasta llegar al destino, este tipo de red es compatible con múltiples aplicaciones, como el Internet de las Cosas. Los procesos de enrutamiento dentro de MANET se convierten en un desafío, por la movilidad aleatoria de los nodos y el cambio esporádico de la topología de la red, con ello, la no sincronización en las transmisiones ocasiona pérdida de paquetes, enlaces intermitentes y mayor interferencia, dando como resultado que la comunicación no sea confiable. En esta investigación, se presenta una propuesta de arquitectura SDN-MANET para solventar esta problemática. La arquitectura se basa en un enrutamiento proactivo donde los controladores SDN se encargan de la gestión de los nodos dentro de la red, se utiliza el protocolo OpenFlow para la comunicación con los controladores. Se evalúa la propuesta en diversos escenarios considerando densidad de red y carga de tráfico, los resultados indican que la propuesta funciona eficientemente y en la mayoría de los escenarios mejor que el protocolo OLSR.

Palabras claves: SDN, MANET, IoT.

Abstract

A MANET network is made up of mobile nodes without infrastructure, it is self-organized and its communication is multi-hop until it reaches the destination. This network is compatible with multiple applications, such as the Internet of Things. The routing process within MANET becomes a challenging task, due to the random mobility of the nodes and the sporadic change of the network topology, with this the non-synchronization in the transmissions causes loss of packets, intermittent links, and greater interference, resulting in unreliable communication. In this research, an SDN-MANET architecture proposal is presented to solve this problem. The architecture is based on a proactive routing where the SDN controllers are in charge of the management of the nodes within the network, the OpenFlow protocol is used for communication with the controllers. The proposal was evaluated in various scenarios addressing, network density and traffic load, the results indicate that the proposal works efficiently and in most cases better than the OLSR protocol.

*Keywords***:** SDN, MANET, IoT.

Introducción

El tráfico de datos en las redes inalámbricas está aumentado constantemente, pronto superará la cantidad de tráfico de las redes cableadas. Con el escenario 5G futuro el patrón de tráfico inalámbrico alojará una creciente demanda de información, es decir, conexiones de datos donde la fuente y el destino se encuentren relativamente cerca (Montaño-Blacio et al., 2020). Algunos ejemplos de dispositivos que producen este tipo de tráfico son los sensores de Internet de las Cosas (IoT por sus siglas en inglés) conectados en casas o ciudades inteligentes (Aktas et al., 2016), los teléfonos inteligentes que comparten tráfico multimedia en una zona determinada (Pappalardo et al., 2016), o los vehículos autónomos que necesitan compartir información local en tiempo real sobre el tráfico o la presencia de obstáculos en las calles.

Con el uso de los estándares de redes actuales, el problema de gestión y administración de gran cantidad de tráfico y dispositivos se convierte en una tarea compleja de realizar. Los avances en las tecnologías inalámbricas han permitido un rápido desarrollo de redes móviles independientes. Una categoría popular de redes móviles son las redes móviles ad hoc (MANET, por sus siglas en inglés), en las que los nodos se autoorganizan de forma autónoma y establecen una comunicación de varios saltos sin cables entre ellos, no hay dispositivos de reenvío dedicados en dicha red, cada nodo es tanto un dispositivo de reenvío como un dispositivo final.

La red MANET se caracteriza por la ausencia de una entidad central. La entidad central llamada punto de acceso (AP) en una red Wi-Fi o estación base (BS) en una red móvil celular, es el centro de la topología de inicio, que puede organizar la red y enrutar cada paquete de datos al destino previsto. MANET no utiliza el control centralizado, cada nodo tiene que actuar de forma independiente, tomar decisiones de enrutamiento y, en el caso de una red móvil, adaptarse dinámicamente a la topología cambiante (Mauve et al., 2001).

En algunas investigaciones se han propuesto varias soluciones para tomar decisiones de enrutamiento de forma distribuida, y también para hacer frente a una red dinámica. En particular, en los casos en que el tráfico es esporádico, la solución es un

protocolo reactivo que encuentra una trayectoria de ruteo solamente cuando sea necesario, como el protocolo del vector de distancia bajo demanda (AODV por sus siglas en inglés) (Ade & Tijare, 2010; Bhardwaj et al., 2012; Jayakumar & Gopinath, 2007). En el caso de un tráfico más regular entre varios pares de origen y destino, un protocolo proactivo que defina en primera instancia la ruta entre cada par posible de nodos en la red puede funcionar mejor, por ejemplo, el protocolo de enrutamiento de estado de enlace optimizado (OLSR por sus siglas en inglés) (Clausen et al., 2003; De Rango et al., 2008).

El principal problema de segundo enfoque es la sobrecarga a la red, los protocolos proactivos requiere intercambios de mensajes de control frecuentes entre nodos para mantener una topología actualizada de la red en cada nodo. Estos mensajes de control pueden causar una sobrecarga alta y, lo que es más importante, en el caso de una topología que cambia rápidamente, pueden no ser suficiente para proporcionar la información de topología actualizada en cada nodo. Para reducir el número de mensajes de control, se puede utilizar un enfoque híbrido entre los protocolos de ruteo proactivos y reactivos en el enrutamiento no basado en topología. Según este enfoque, la red se divide en grupos o clústeres y, si los nodos de origen y destino están dentro del mismo clúster, se utiliza una estrategia de enrutamiento proactiva como OLSR; de lo contrario, se adopta una estrategia reactiva como AODV.

Un algoritmo de este tipo es el protocolo de enrutamiento de zona (ZRP por sus siglas en inglés) (Haas, 1997). Este enfoque híbrido tiene la ventaja de reducir la cantidad de mensajes de control, ya que la información actualizada de topología debe mantenerse solo entre los nodos de cada clúster. La desventaja principal es que se introduce un retardo significativo cuando un paquete se destina a un nodo que se desea enviar a un clúster diferente, puesto que se debe buscar una nueva trayectoria de manera reactiva. Para abordar esta problemática de enrutamiento y movilidad de los nodos se emplean controladores bajo el concepto de redes definidas por software, con el objetivo de gestionar y mantener la topología actualizada de la red móvil.

Capítulo uno

Conceptualización del problema

Algunas suposiciones sobre la movilidad de los nodos en la red se basan en los enfoques de enrutamiento reactivo, proactivo e híbrido, y estos supuestos permiten optimizar el equilibrio entre reducir la sobrecarga, en términos de mensajes de control intercambiados en la red, y tener un conocimiento perfecto de la topología de red, lo que permite el diseño de rutas de tráfico óptimas, pero a costa de mensajes de control más frecuentes. Si bien es posible encontrar una compensación óptima dependiendo de la movilidad de la red, es difícil lidiar con una red cuyas características de movilidad son desconocidas, o puede cambiar con el tiempo. Como ejemplo, piense en una multitud de personas que asisten a un evento deportivo, y todas quieren transmitir archivos entre ellas.

Para adaptarse a las características cambiantes de movilidad de red y carga de tráfico, se propone segmentar la inteligencia de la red en base a controladores, dividir el plano de datos de los nodos, incluido todo el tráfico de datos intercambiado localmente entre ellos, y el plano de control comprendiendo todos los paquetes de control y las decisiones de ruteo locales. Esta lógica se toma del paradigma de la red definida por software (SDN por sus siglas en inglés) (Xia et al., 2014), que nació para ser utilizada en las redes cableadas con el objetivo de proporcionar el control centralizado sobre el enrutamiento de paquetes, y con el tiempo han sido adaptadas para redes móviles (Santamaria, 2017).

La aplicación de los principios de SDN a MANET requiere una conexión de protocolo de control de transmisión confiable entre controlador SDN y cada nodo de reenvío para mensajes de control (Dusia, 2019). Pero las características de MANET como la movilidad de los nodos, los enlaces intermitentes y la topología dinámica hacen que las conexiones entre los nodos no sean fiables. La mayoría de las arquitecturas MANET basadas en SDN asumen que el controlador se comunica con el nodo mediante un enlace de un solo salto a través de un canal separado. Además, también suponen que está disponible una estación base para albergar el controlador y un servicio de ubicación para rastrear las posiciones de los nodos móviles.

Tomando en consideración este contexto, la propuesta actual está diseñada para redes sin infraestructura donde el controlador está alojado en cada uno de los nodos móviles, sin la necesidad de un dispositivo central para identificación de la ubicación de los nodos, es por ello que, para solventar esta problemática se hace uso de las redes definidas por software (SDN) y a través del protocolo OpenFlow (Dely et al., 2011) realizar la comunicación del plano de datos con el plano de control, la solución se basa en el uso de en un protocolo de enrutamiento proactivo que se implementa en el simulador NS-3. Las métricas de evaluación y su análisis se obtienen utilizando varios escenarios de red y se comparan con el protocolo OLSR bajo las mismas condiciones.

1.1 Justificación

IoT es un paradigma que permite a los usuarios intercambiar grandes cantidades de información con tiempos de propagación en milésimas de segundo. Esto se logra debido a la constante mejora de software y hardware; así como también a la reducción de distancia entre equipos de comunicación. En este contexto, SDN permite migrar la inteligencia de la red a un nivel de plano de control, al tener centralizada la información de la red, las decisiones de gestión, asignación de recursos y calidad de servicio son más eficientes. Por esta razón, SDN es muy conveniente para ser implementada y solventar las características de arquitectura IoT.

A su vez el paradigma de IoT utiliza las redes MANET para permitir a los usuarios mantener la capacidad de movimiento sin perder la conectividad ni las características asociadas al servicio que estén utilizando. De acuerdo con los requisitos de IoT y las características de MANET, SDN es una alternativa eficiente para la asignación de recursos en la red. Sin embargo, el uso de SDN en redes MANET impacta en el tiempo de respuesta de la red, las aplicaciones y los servicios ofrecidos. Esto presenta algunos retos que son temas de interés actual. Por esta razón, el presente trabajo de investigación se centra en analizar y proponer el uso de SDN en redes móviles tomando en consideración los requisitos de IoT.

1.2 Objetivos

1.2.1 Objetivo general

- Determinar las estrategias para mejorar la capacidad de gestión en forma centralizada en redes móviles tomando en consideración los requisitos de IoT.

1.2.2 Objetivos específicos

- Analizar el estado de la cuestión de las de redes definidas por software para redes móviles.
- Detallar escenarios con diferentes tipos de tráfico y de servicios disponibles en la red.
- Proponer soluciones probables para la gestión centralizada en redes móviles.
- Comparar las soluciones planteadas para determinar las más adecuadas de acuerdo a los requisitos de IoT y las características de las redes MANET.

1.3 Metodología

El presente trabajo de titulación parte de la investigación de las propuestas de redes definidas por software (SDN) para redes móviles (MANET) en los últimos años, y con ello, proponer una solución viable y medible para redes sin infraestructura con movilidad aleatoria. Con la información recolectada y las especificaciones de la red IoT, se procede con la implementación de SDN sobre un protocolo proactivo OLSR en simulador NS-3, así mismo, con la implementación de protocolo OLSR para la comparativa. Para la implementación de los protocolos es necesario un análisis de los requisitos y configuración de la herramienta de simulación, con el sistema funcional se realiza la codificación de los protocolos de enrutamiento en los escenarios propuestos, al final se implementa las respectivas simulaciones para evaluar las métricas de la red y realizar la comparativa entre los procotolos.

1.4 Estructura del documento

El trabajo de titulación está organizado de acuerdo con la siguiente estructura:

- **Capítulo 1:** En esta sección se presenta la justificación del proyecto y con ello, los objetivos y metodología que se aplicará.
- **Capítulo 2:** Marco teórico, en este capítulo se presenta los conceptos fundamentales para abordar la investigación como: Redes móviles ad hoc (MANET), Protocolos de enrutamiento MANET, Redes definidas por software (SDN), Internet de las cosas (IoT) y la herramienta de simulación NS-3.
- **Capítulo 3:** SDN-MANET y OLSR-MANET, en este capítulo se presenta un estudio detallado del protocolo de enrutamiento proactivo OLSR y SDN sobre redes MANET, donde se detallan propuestas existentes, funcionamiento, características y su compatibilidad con la herramienta de simulación NS-3.
- **Capítulo 4:** Simulación, análisis y discusión de resultados, en este capítulo se detalla los escenarios para la simulación del protocolo de enrutamiento SDN y OLSR, y el proceso de simulación en NS-3. Se presenta los resultados obtenidos con su respectivo análisis, resultados que contemplan, el rendimiento, la tasa de envíos de paquetes de aplicación, la tasa promedio de envío de paquetes, el retardo promedio, la tasa de retardo de paquetes, la pérdida de paquetes. Se realizan comparaciones entre los resultados de las simulaciones del protocolo OLSR-MANET con SDN-MANET.

Como posteriores apartados, se presentan las conclusiones obtenidas de la investigación en base a los objetivos planteados y se proporcionan algunas recomendaciones o posibles trabajos futuros sobre la gestión de redes móviles. Finalmente, se presenta los respectivos anexos que dan a conocer el cumplimiento de la implementación, codificación de los protocolos en la herramienta NS-3.

Capítulo dos

Marco Teórico

En el presente capítulo se muestra un estudio a detalle de los conceptos principales para abordar la problemática planteada, sirve como marco de referencia para la investigación, como eje fundamental se aborda las características de las redes móviles ad hoc (MANET), redes definidas por software (SDN), Internet de las Cosas (IoT) y simulador de red NS-3, con el objetivo de dar a conocer como SDN aporta a la gestión de las redes MANET e IoT.

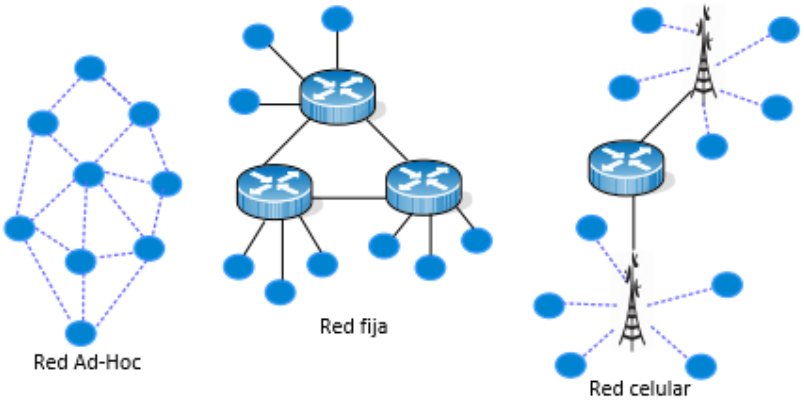
2.1 Redes ad hoc inalámbricas

Una red ad hoc móvil consiste en un grupo de nodos móviles que se comunican directamente entre sí, haciendo uso de un canal inalámbrico común sin requerir una infraestructura fija. Esta característica permite levantar y configurar una red temporal para la comunicación entre dispositivos de igual a igual, sin la necesidad de enrutadores o puntos de acceso.

Una red MANET carece de infraestructura fija, lo que conlleva, que los nodos móviles se encarguen de organizarse dinámicamente, descubrir las rutas para la comunicación entre dispositivos, y así, proporcionar la funcionalidad necesaria a la red. En estos entornos cambiantes, los dispositivos móviles se comportan como nodos de enrutamiento, donde los paquetes viajan desde un origen hacia un destino haciendo uso de uno o múltiples nodos hasta llegar al destinatario. Toda la inteligencia de la red MANET se la aloja en cada uno de los dispositivos de usuarios móviles, esto se puede contrastar con las redes basadas en infraestructura, donde cada dispositivo de la red se comunica con los demás elementos a través de un solo punto de acceso enrutador inalámbrico, donde se centra la inteligencia de la red para la gestión y administración de los dispositivos. Las redes celulares son un ejemplo de comunicación a través de un punto de acceso denominado estación base (BS), en la Figura 1, se muestran dispositivos móviles que se distribuyen como red ad-hoc y red con infraestructura.

Figura 1

Redes fijas e inalámbricas



Nota. Adaptado de (Santamaria, 2017)

Las redes ad hoc se emplean en una amplia gama de aplicaciones, donde no es posible levantar una red con infraestructura que aporte a la movilidad y sobre todo a la autoorganización de los nodos. Algunas aplicaciones referentes a estas redes ad-hoc son, entornos militares, redes de área personal, industria, operaciones de rescate, emergencia, trabajos colaborativos, redes vehiculares (VANET por sus siglas en inglés) y redes de sensores (WSN por sus siglas en inglés) (Hoebeke et al., 2004). Muchas investigaciones en los últimos años se han centrado en el estudio de este tipo de redes móviles ad hoc para solventar problemáticas sobre los protocolos de enrutamiento, tomando en consideración que este tipo de redes cuentan con restricciones en los campos: energía limitada, conexión inalámbrica, ancho de banda, control de potencia, espacio de almacenamiento y seguridad, y con ello, por la naturaleza de red sin infraestructura fija, complican más el estudio de este tipo de redes generando nuevos problemas tales como, mantenimiento de la topología, enrutamiento, descubrimiento de nodos y configuración de red (Basagni et al., 2004).

Entre las características más comunes de MANET se puede mencionar; enlaces intermitentes debido a la movilidad de los nodos y cambios de topología, rango limitado de transmisión inalámbrica provocada por la necesidad de comunicación de varios saltos, colisiones e interferencias debido a las transmisiones no síncronas y enlaces inestables a causa de pérdidas en la propagación.

Las limitaciones y características específicas de las redes MANET imponen desafíos en el diseño de protocolos de red en las capas de la pila de protocolos como se muestra en la Figura 2. La capa física se encarga de solventar y hacer frente a los cambios inesperados y rápidos de los enlaces. La capa de control de acceso a medios (MAC por sus siglas en inglés), cumple con la función de permitir el acceso al canal, minimizar la colisión de tramas, gestionar nodos ocultos y expuestos. La capa de red se encarga del enrutamiento, para ello se requiere de una estrategia inteligente donde los nodos deben ayudar al cálculo de las rutas o caminos de envío y aprovechar correctamente los recursos limitados de los nodos móviles. La capa de transporte se encarga de manejar la pérdida de paquetes y sobre todo la latencia o retardo de red. La capa de aplicación debe manejar eventos esporádicos como posibles desconexiones y reconexiones sin pérdida de servicio (Jayakumar & Gopinath, 2007).

Figura 2

Arquitectura MANET

<table>
<tr><td rowspan="2">Aplicaciones
&
Middleware</td><td>Aplicación 1</td><td>Aplicación 2</td><td>Aplicación k</td><td rowspan="4">Problemas en las capas
• Seguridad y Cooperación
• Conservación de energía
• Simulación
• Calidad de servicio</td></tr>
<tr><td colspan="3">Middleware
Ubicación del servicio, memoria compartida de comunicación grupal</td></tr>
<tr><td>Red</td><td colspan="3">Protocolos de capa de red y transporte
TCP, enrutamiento IP, direccionamiento, ubicación, multidifusión, Interconexión</td></tr>
<tr><td rowspan="2">Tecnologías
Habilitantes</td><td>802.11</td><td>Bluetooth</td><td>HiperLAN</td></tr>
<tr><td colspan="3">Control de acceso al medio, antenas, control de potencia</td></tr>
</table>

Nota. Adaptado de (Jayakumar & Gopinath, 2007)

Enrutamiento en red ad hoc

El enrutamiento entre nodos en la red ad hoc depende de múltiples factores, incluyendo inicio de solitudes, cambios en la topología y cálculos de las mejores rutas, el objetivo es determinar la ruta más rápida y eficiente para enviar los paquetes entre nodo inicio y nodo destino. El tráfico que se genera entre los nodos para llevar los paquetes hacia los nodos destino, la movilidad aleatoria de los nodos y las interferencias del canal inalámbrico complican aún más la labor de los protocolos de enrutamiento sobre estas redes

para seleccionar el camino o ruta idónea, considerando que el enrutamiento es la tarea más relevante que cumplen los nodos en la red móvil para garantizar la eficiencia y correcto funcionamiento de la red ad hoc.

Los protocolos de enrutamiento en una MANET son diferentes del enrutamiento tradicional, los protocolos tradicionales no se pueden adaptar ni usar en este tipo de redes debido principalmente a los cambios aleatorios de los nodos móviles. Se puede mencionar, los enrutamientos de redes tradicionales como los que se encuentra en las redes de malla, denominado redes de sensores inalámbricos (WSN). Por consiguiente, estas redes de sensores se componen de una estación coordinadora que se encarga de administrar y optimizar la comunicación de los nodos dentro de la red (Yick et al., 2008). Sin embargo, en una MANET todo el grupo de nodos de red se mueven con dirección aleatoria de forma continua sin la intervención de un controlador o coordinador de red central, con esta topología cambiante configuran rápidamente sus tablas de enrutamiento para establecer la comunicación. Los protocolos de enrutamiento de red ad hoc se clasifican en tres categorías: proactivo, reactivo e híbrido.

Proactivo: Los protocolos proactivos también conocidos como protocolos de enrutamiento controlado por tablas o basado en tablas de ruteo, hacen uso de un algoritmo de enrutamiento de estado enlace para el descubrimiento de rutas. El proceso que siguen los nodos es realizar un seguimiento de las rutas a todos los destinos posibles, incluyendo a los nodos en los que no es necesario reenviar paquetes, con esta perspectiva se mantiene una tabla de enrutamiento temporal antes de que inicie la comunicación entre nodos. Sin embargo, en caso de cambios de la topología de la red, todos los nodos que conforman la red reaccionan rápidamente actualizando sus tablas de enrutamiento, con el objetivo de mantener una comunicación fiable. El intercambio periódico de mensajes de control sirve para establecer las nuevas rutas por los cambios drásticos de topología, causa ciertas problemáticas, entre ellas, reducción de la potencia del nodo y el aumento de ancho de banda, por esta razón, los protocolos proactivos no son aptos para redes grandes. No obstante, una ventaja de estos protocolos proactivos es su retardo mínimo en la respuesta

a una solicitud cada vez que se necesita una ruta para el envío de los paquetes (Bhardwaj et al., 2012).

Reactivo: Los protocolos reactivos conocidos como protocolos bajo demanda, presentan un enfoque opuesto a los protocolos proactivos, donde los nodos buscan o descubren las rutas solo cuando exista una solicitud para envío de paquetes. Por lo tanto, no necesitan mantener la información de enrutamiento en los nodos de la red si no existe comunicación. El proceso que siguen los nodos para enviar un paquete se da cuando un nodo inicia un proceso de llamado de descubrimiento de ruta y posterior almacena el registro de esa ruta. Mediante este proceso los nodos no necesitan mantener la ruta establecida hasta el destino, razón por la cual, cada nodo lleva la información de forma independiente para ser entregado al nodo adyacente o a los demás nodos de la red. Los principales beneficios de este tipo de protocolos reactivos son, las pequeñas cantidades de datos que ocupan para transferir la información y la no necesidad de guardar toda la tabla de enrutamiento de la red. Sin embargo, existen algunos problemas que se reflejan directamente en el retraso o demoras en el tiempo de entrega de paquetes, que se produce por la búsqueda de la ruta, causando que el tiempo medio de retraso de la red sea mayor (Santamaria, 2017).

Híbrido: este tipo de protocolos combina las características de enrutamiento proactivo basado en tablas y enrutamiento reactivo bajo demanda, con el objetivo de poder masificar el tamaño de la red y la densidad de nodos. Cada nodo en una zona determinada como vecindario local se comporta como un nodo de red proactivo manteniendo la información de la tabla de enrutamiento de su zona de cobertura, mientras que adquieren rutas de forma reactiva a destinatarios que se encuentre fuera de la cobertura, por lo tanto, dentro de una zona determinada se utiliza el enrutamiento proactivo, fuera de ella los nodos utilizan el enrutamiento reactivo. Al combinar correctamente este enfoque, los protocolos bajo demanda facilitan un mejor rendimiento (Khatkar & Singh, 2012).

Protocolos de enrutamiento MANET

Desde el punto de vista de los protocolos de enrutamiento, las características de MANET son diferentes a otras redes inalámbricas y por ende a las redes cableadas, debido a su complejidad dieron como resultado la generación de una nueva gama de protocolos de enrutamiento, denominados protocolos de enrutamiento MANET. En la literatura se han propuesto varios protocolos de enrutamiento, tomando como relación el enrutamiento vector distancia o el de estado enlace, que la gran mayoría de estos protocolos toma como referencia.

Los nodos que utilizan los protocolos de enrutamiento por vector distancia, se encargan solo de supervisar el coste de sus enlaces salientes, y con ello, anuncian su distancia a sus vecinos, estos reciben los costes de los enlaces, los combinan para seleccionar la mejor ruta de destino a través de algoritmos de selección de la ruta más corta. En un protocolo de enrutamiento de estado enlace, cada nodo se encarga de sondear periódicamente sus enlaces a través de mensajes de saludo, posterior enviar la información a todos sus vecinos, con esto, cada nodo mantiene actualizada la información de la topología, la utiliza para seleccionar las rutas individuales a todos los posibles nodos destino. Los protocolos de enrutamiento vector distancia y estado enlace se pueden diseñar de forma proactiva o reactiva (Liu & Kaiser, 2005).

El enrutamiento reactivo es usual utilizarlo en las MANET con tráfico de red poco frecuente. Los protocolos que se destacan en este enrutamiento son, el protocolo de enrutamiento dinámico de origen (DSR por sus siglas en inglés) (Johnson et al., 2007) y el protocolo vector distancia ad hoc bajo demanda (AODV por sus siglas en inglés) (Ade & Tijare, 2010). Estos protocolos operan a través de un procedimiento de solicitud de ruta, que contempla, mensajes de solicitud de inundación, con ello, hace que el remitente aprende la ruta hacia el destino. También, pueden utilizar el procedimiento de mantenimiento de ruta para lograr mantener las rutas utilizadas previamente. El uso de estos protocolos ocasiona un alto retraso en la red, a causa del almacenamiento de los paquetes en búfer, mientras los nodos ejecutan el procedimiento de descubrimiento de ruta.

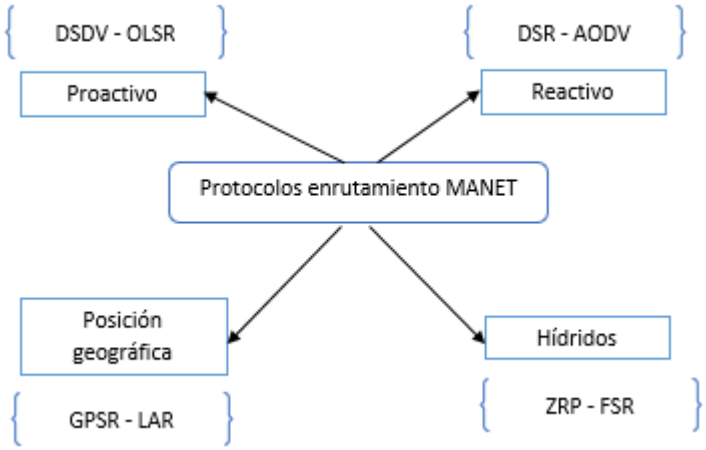
Por otro lado, el enrutamiento proactivo en las MANET se utiliza con frecuencia en las redes con tráfico continuo y regular, en donde, las posibles rutas de destino se almacenan de antemano entre los nodos de la red. Entre los protocolos se puede mencionar, el protocolo de vector distancias secuenciado por destino (DSDV por sus siglas en inglés) (Ade & Tijare, 2010) y el protocolo de enrutamiento de estado enlace optimizado (OLSR) (De Rango et al., 2008). La actualización proactiva de las tablas de enrutamiento y la selección de rutas con antelación de todos los nodos de la red ocasiona una sobrecarga elevada para lograr un rendimiento eficiente de la red, estos protocolos utilizan diferentes tipos de mensajes para el descubrimiento de la red, según sus características de enrutamiento ya sea vector distancia o estados enlace.

Los protocolos de enrutamiento híbridos adoptan las características de los protocolos proactivos y reactivos. Algunos ejemplos son el protocolo de enrutamiento de zona (ZRP) (Khatkar & Singh, 2012) y el protocolo de enrutamiento de estado de ojo de pez (FSR por sus siglas en inglés) (Khatkar & Singh, 2012). La idea general de estos protocolos hídricos es identificar o establecer zonas entre nodos de la red, luego de segmentar las zonas, permiten el enrutamiento proactivo dentro de cada zona y el enrutamiento reactivo entre las zonas de la red. Estos protocolos apuntan a un compromiso entre la sobrecarga y el retraso en función de las características de la red, como el tamaño, la densidad y la movilidad. Sin embargo, lograr una compensación óptima es difícil cuando las características de la red no están disponibles o varían con el tiempo.

Algunos protocolos de enrutamiento (Mauve et al., 2001), hacen uso de los servicios de ubicación como GPS, para rastrear las posiciones de los nodos y seleccionar rutas entre ellos. Sin embargo, estos protocolos enfrentan dificultades cuando las topologías reales basadas en la pérdida de ruta y la conectividad son diferentes de las estimadas. En la Figura 3, se presenta un resumen de los protocolos abordados.

Figura 3

Clasificación de protocolos de enrutamiento MANET



Nota. Adaptado de (Bhardwaj et al., 2012)

Uno de los desafíos de enrutamiento más comunes en las redes MANET es la escalabilidad, donde el protocolo deber ser capaz de trabajar eficientemente con el crecimiento exponencial de la red o más parámetros de red como, tamaño de red, densidad de la red, la velocidad del nodo y carga de tráfico (Hong et al., 2002). Los protocolos en este contexto experimentan variaciones en su rendimiento cuando se presentan variaciones dentro de estos parámetros críticos. Tomando en consideración que varias clases de protocolos operan en diferentes escenarios, generalmente comparten objetivos comunes, como reducir la sobrecarga, maximizar el rendimiento y minimizar la demora. El factor diferenciador entre los protocolos es la forma en que seleccionan y mantienen las rutas. Casi todos los protocolos tienen sus escenarios de implementación y combinaciones de parámetros en los que superan a otros protocolos. La selección óptima a menudo requiere un análisis cuidadoso del escenario, los requisitos y las características de la red.

2.2 Redes definidas por software (SDN)

La Open Networking Foundation (ONF) es la organización dedicada al desarrollo, estandarización y comercialización de SDN. ONF define de forma explícita a SDN como: una arquitectura de red emergente donde el control de red está desacoplado del reenvío y es directamente programable (ONF, n.d.). Estas redes SDN han conseguido un cambio

significativo al paradigma de las redes de comunicación, tienen como propósito la agilidad y facilidad de implementar servicios de forma remota y escalable, en comparación con las redes tradicionales que son bastante complejas y rígidas, SDN se caracteriza por su facilidad de programación en la arquitectura de red (Bizanis & Kuipers, 2016). La red definida por software al ser una nueva tecnología de red, se está utilizando en múltiples aplicaciones para la gestión de recursos, las aplicaciones de Internet de las Cosas, seguridad, entre otras.

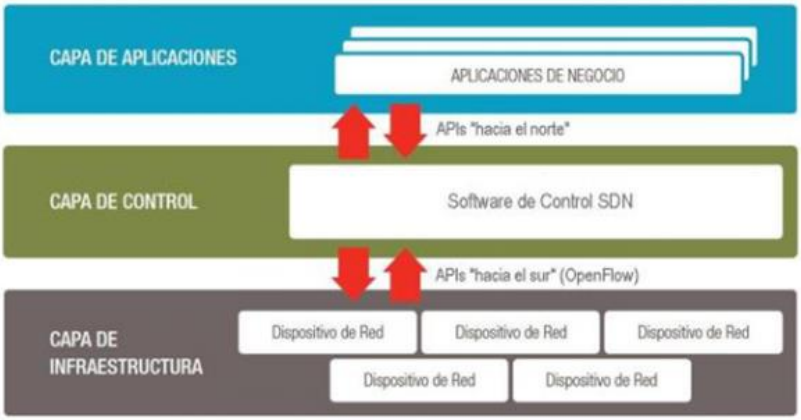
La arquitectura de SDN está diseñada para simplificar y mejorar la gestión de la red con alta flexibilidad al dividir el plano de control del plano de datos. Por lo tanto, la programabilidad de la red se mejora, lo que a su vez genera más oportunidades de innovación e investigación. SDN logra estas funcionalidades de gestión centralizando la inteligencia en el controlador SDN. El controlador ofrece los módulos adecuados para admitir la gestión centralizada, la movilidad de los nodos y la topología (Hakiri et al., 2017). En la Figura 4, se presenta la arquitectura SDN, misma que se compone de tres capas: capa de aplicación, capa de control y capa de datos. Cada una de estas capas se comunican con sus capas contiguas a través de interfaces norte y sur. La interfaz norte permite la comunicación de la capa de aplicación con el plano de control y la interfaz sur sirve para comunicar el plano de control con el plano de datos o capa de infraestructura (Santamaria, 2017).

La capa de aplicación es donde se ejecutan diversas aplicaciones SDN, permite la interacción con toda la arquitectura de una forma rápida, incluye servicios de redes, monitor de herramientas y la configuración de equipos de forma remota junto con el apoyo de las API’s hacia el norte para la comunicación con el controlador SDN. La capa de control basa su funcionamiento en el controlador SDN o controladores SDN que se pueden coordinar entre si a través de interfaces este u oeste para la toma efectiva de decisiones sobre los dispositivos de reenvío, una particularidad de estos controladores es la visión completa o parcial de la red para asegurar su gestión y administración de forma efectiva. Finalmente, el plano de datos o de infraestructura es donde se alojan los diversos dispositivos de conmutación SDN que pueden ser físicos o virtuales conectados por medios alámbricos o

inalámbricos, estos dispositivos no cuentan con una funcionalidad predefinida, sino que obedecen a las instrucciones dadas por el plano de control (Montaño et al., 2021).

Figura 4

Arquitectura SDN



Nota. Adaptado de (Santamaria, 2017)

La arquitectura SDN hace uso ampliamente del protocolo OpenFlow (McKeown et al., 2008), para la gestión del plano de control sobre el plano de datos, mismo que permite gestionar y configurar los dispositivos de reenvío de forma remota. OpenFlow se ha convertido en el protocolo más utilizado para implementar los escenarios SDN. La función principal de OpenFlow es permitir la comunicación entre los controladores SDN y los nodos. Por lo tanto, el controlador será el que se encarga de orquestar toda la red convirtiéndose en pieza fundamental de la arquitectura SDN, será el que centraliza las funciones de red y sondeo del estado global de la red. En la Tabla 1, se presenta un cuadro resumido de los controladores habilitados para SDN, haciendo referencia a la compatibilidad con el protocolo OpenFlow y con sus respectivos lenguajes de programación soportados.

Tabla 1

Controladores SDN

Controlador	Lenguaje de programación	Protocolo
NOX	C++, REST, Phython	OpenFlow 1.0, 1.3
Opendaylight	Java, Python, REST	OpenFlow 1.0, 1.3
POX	Python, REST	OpenFlow 1.0
ONOS	Java	OpenFlow 1.0, 1.3, 1.4

Ryu	Python	OpenFlow 1.0, 1.2, 1.3, 1.4, 1.5
Beacon	Java	OpenFlow 1.0
HiperFlow	Java, Python, REST	OpenFlow 1.0, 1.3, 1.4
Maestro	Java	OpenFlow 1.0
Kandoo	Python, C++, C	OpenFlow 1.0
MUL	C	OpenFlow 1.0, 1.3, 1.4
Trema	Ruby, C	OpenFlow 1.0
DISCO	Java	OpenFlow 1.1
ElastiCon	Java	OpenFlow 1.0
DropController	C, python	OpenFlow 1.0, 1.3
LearningController	C, python	OpenFlow 1.0, 1.3

Nota. Esta tabla muestra un resumen de los controladores habilidatos para SDN

Las redes SDN, al desacoplar el plano de datos del plano de control, ofrecen mayor control de red a través de programación, esta característica conlleva beneficios potenciales entre ellos, mejora de la configuración, mejora el desempeño y fomenta la innovación sobre la mejora de la arquitectura, gestión de red y aplicaciones varias. Por ejemplo, el controlador SDN pueden romper las barreras de las capas del modelo TCP/IP, puede incluir el reenvío de paquetes a nivel de conmutación, también trabajar sobre las conexiones a nivel de enlace de datos. Así mismo, SDN cuenta con la capacidad de obtener el estado real e instantáneo de la red, con ello, permite el control central en tiempo real basado tanto en estado temporal como en políticas acordadas por el usuario. Esto conduce directamente a beneficios y mejoras sobre diferentes tipos de redes con el uso de SDN mejorando optimización de configuración de red y sobre todo el rendimiento de la red (Xia et al., 2014).

2.3 Internet de las Cosas (IoT)

El Internet de las Cosas (IoT), en la actualidad es considerado un área de investigación emergente donde los objetos cotidianos son capaces de comunicarse entre ellos y a su vez con los usuarios. Esta evolución rápida de IoT representar arduos retos, en las áreas de recolección, estadística, distribución y visualización de datos, con el objetivo de mejorar la interacción de humanos con los objetos. El IoT implica la administración de dispositivos sensores distribuidos en la red, con el propósito de notificar al usuario posibles

eventualidades en tiempo real. Estos dispositivos cuentan con un cierto grado de inteligencia para actuar de forma autónoma, poseen recursos limitados: de potencia, memorias, procesamiento y ancho de banda, en vista de esta problemática los protocolos tradicionales de operación de red no son aceptados en los ecosistemas IoT. Hoy en día, existen muchas aplicaciones de IoT, como monitoreo ambiental, automatización del hogar, sistemas médicos y de salud, transporte, administración de energía e infraestructura, etc. Los desarrollos recientes de las aplicaciones de IoT nos brindan una amplia gama de oportunidades para crear valor agregado para los sectores de la industria y los usuarios (Ngu et al., 2016).

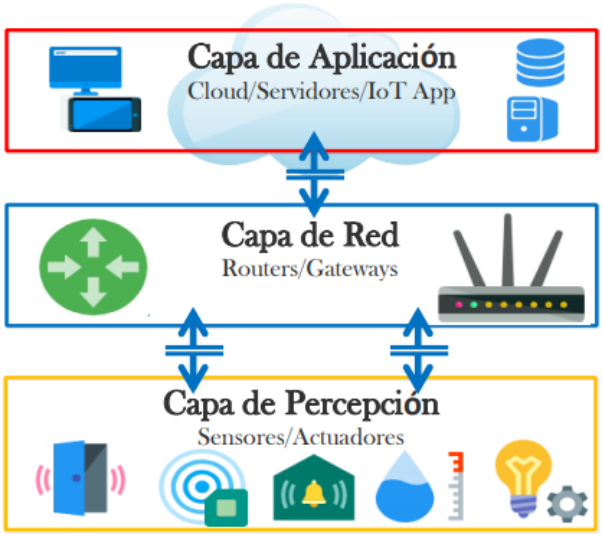
2.3.1 Arquitectura IoT

Se han desarrollado múltiples arquitecturas para IoT, como el modelo NIST para Smart Grid, el modelo ITU-T (ITU-T, 2012), el modelo M2M de ETSI (Alam et al., 2013), etc. En la actualidad, no se evidencia una arquitectura IoT que se adapta a todos los aplicativos de forma universal. Diferentes arquitecturas se han propuesto en el trascurso del tiempo, para el análisis se hace referencia al diseño más común para lograr plasmar el el funcionamiento de IoT(Jing et al., 2014), esta arquitectura consta de tres capas, capa de percepción, capa de red y capa de aplicación, en la Figura 5 se presenta la arquitectura.

Capa de percepción: es la capa física donde se ubican los dispositivos como sensores, permitiendo la comunicación con el gateway IoT. El objetivo principal de esta capa es detectar y recopilar la información del entorno como humedad, temperatura, ubicación, movimiento etc., además de identificar objetos con la ayuda de diferentes elementos como lectores RFID, tarjetas inteligentes y redes de sensores.

Figura 5

Arquitectura IoT



Nota. Adaptado de (Chiliquinga Rodríguez, 2020)

Capa de red: tiene como objetivo principal la transmisión de datos de la capa de percepción a la capa de aplicación. Para llevar a cabo la transferencia de datos, la capa de red utiliza distintos tipos de tecnologías y protocolos para garantizar la comunicación, por ejemplo; redes celulares, Bluetooth, Zigbee, Wi-Fi, 6LowPAN, ad hoc, dependiendo de los dispositivos sensores.

Capa de aplicación: el objetivo principal de la capa es la de incluir las aplicaciones IoT, por ejemplo, ciudades inteligentes, casas inteligentes, por consiguiente, también aborda servidores para compartir información y proporcionar servicios en tiempo real. Dicha capa cumple con la función medular de IoT que se encarga de adquirir, procesar y almacenar la información recibida de los dispositivos que comprenden la red IoT.

2.3.2 Protocolos IoT

Los dispositivos que intervienen en una red de IoT necesitan diferentes protocolos para permitir la interconexión entre la parte física y digital, permitiendo que la información proveniente de los nodos se recopile y procese en tiempo real. Hay varios protocolos de comunicaciones que se pueden utilizar en entornos de IoT (Tabla 2). IoT de acuerdo con la pila de protocolos TCP/IP contempla sus protocolos de red, basa su funcionamiento en un

modelo de 4 capas que son útiles para sus diversas aplicaciones, a continuación, se muestran una clasificación exhaustiva de los protocolos referentes al modelo TCP/IP. Como se evidencia algunos de los protocolos son conocidos y otras son desarrollados propios para aplicaciones IoT.

Tabla 2

Protocolos IoT

Capas TCP/IP	Protocolos
Nivel de aplicación Interfaz entre el usuario y el dispositivo	**AMQP:** Nivel de software que crea interoperabilidad entre el middleware de mensajería. **CoAP:** Protocolo diseñado para la comunicación máquina – máquina, con el objetivo de interconectar dispositivos con capacidades limitadas. **DDS:** Protocolo que se puede ejecutar en dispositivos pequeños, puede conectarse a redes extensas de rendimientos altos, con comunicación punto a punto. **MQTT:** Protocolo para la comunicación ligera entre equipos con conexiones de poco ancho de banda.
Nivel de transporte Se encarga de iniciar y resguardar los datos en el trascurso de la comunicación.	**TCP:** Protocolo más usado en Internet por su confiabilidad, permite segmentar grandes conjuntos de datos en pequeños paquetes que reenvía según sea necesario, es orientado a la conexión. **UDP:** Protocolo que permite enviar datos en la red sin antes haber establecido conexión, los datagramas contienen la información necesaria para el direccionamiento.
Nivel de red Permite el enrutamiento de los paquetes entre emisor y receptor.	**IP:** pueden usar IPV4 e IPV6, convirtiéndolo a IPV6 un habilitante para IoT, esta actualización de IP logra dirigir el tráfico en la red de Internet logrando darle un valor agregado como, localización e identificación de dispositivos conectados. **6LoWPAN:** Protocolo IoT de consumo de energía bajo, construido con el fin de manejar dispositivos de baja potencia, con capacitad restringida en el procesamiento.
Nivel físico Canal de comunicación entre los dispositivos	Tecnología de acceso como: WiFi, Bluetooth, Zigbee, NFC, red celular, LoraWan, LowPan, Z-Ware, Sigfox.

Nota. Esta tabla muestra un resumen de los protocolo IoT de acuerdo al modelo TCP/IP

2.4 Simulador NS-3

El software NS-3 es un simulador de redes de eventos discretos, ampliamente utilizado en el campo de la investigación y enseñanza. Es un sistema de código abierto de uso gratuito amparado bajo la licencia GNU GPLv2. Cuenta con una API opcional de script de python, permite adentrarse a profundidad en el estudio de los protocolos de Internet y redes de gran tamaño sobre entornos controlados. No es compatible con su antecesor NS-2, es un nuevo y reformado simulador codificado desde cero, bajo la programación del lenguaje de C++ (ns3, n.d.). A través de NS-3 se puede fomentar el desarrollo de sistemas de simulación realistas que trabajan en tiempo real, permitiendo reusar muchas implementaciones de código de protocolos de existente. NS-3 es usado en una gran variedad de investigaciones, que se enfocan en el análisis de redes inalámbricas sobre el protocolo IP, incluyen diferentes modelos como, WIFI, LTE, WIMAX, con simulaciones sobre las capas del modelo de red y en la gran variedad de protocolos de enrutamiento estáticos o dinámicos como OLSR y AODV para aplicaciones basadas en IP.

Capítulo tres

Sdn-Manet y Olsr-Manet

3.1 Arquitecturas basadas en SDN para MANET

Las investigaciones relacionas al presente trabajo (Aslam et al., 2017; Bellavista et al., 2020; Correia et al., 2017; De Gante et al., 2014; Detti et al., 2013; Hans et al., 2017; He et al., 2016; Ku et al., 2014; Truong et al., 2015; Zhu et al., 2015) presentan arquitecturas centralizadas basadas en SDN para MANET, donde el controlador SDN es estacionario y se mantiene centralizado en la arquitectura, emplea en la mayoría de los casos un solo salto directo para salir a una red diferente o conexiones de IP diferentes sobre la misma red, con el objetivo de controlar la comunicación con los nodos, donde se evidencia arquitecturas para nodos sensores (De Gante et al., 2014), nodos vehículos (He et al., 2016; Ku et al., 2014; Truong et al., 2015; Zhu et al., 2015) y arquitectura para distribución de nodos en tipo malla (Detti et al., 2013). No obstante algunas propuestas (De Gante et al., 2014; He et al., 2016), aplican servicios de seguimiento de ubicación, con el propósito de facilitar el aprendizaje de la topología de la red al controlador y mejorar las comunicaciones de múltiples saltos, aunque, en otras propuestas (Bellavista et al., 2020; Detti et al., 2013) son objetivos para evaluar la calidad de servicio (QoS por sus siglas en inglés) y la carga de tráfico. Sin embargo, es un trabajo de interés establecer una arquitectura que se adapte a una red sin infraestructura, sobre todo se centra la problemática en la capa de red y en la inteligencia del controlador SDN.

En la propuesta (Dely et al., 2011) se da a conocer una arquitectura para redes malladas, donde los autores sugieren el uso de puertas de enlace y puntos de acceso habilitados para OpenFlow, lo que permite la comunicación del controlador con el plano de datos, el controlador en este aspecto mallado se encarga de activar y gestionar el traspaso de los nodos móviles entre los puntos de acceso.

Un caso práctico de uso en la gestión de tráfico se evidencia en (Detti et al., 2013) donde configuran reglas de forma dinámica para logra una eficiencia en el equilibrio de tráfico de Internet, esto aplicando diversas puertas de enlace. Sin embargo, en ambas

investigaciones hacen uso del protocolo OLSR para el enrutamiento de los paquetes, que se lo realiza a través de los nodos haciendo uso de varios saltos. En (De Gante et al., 2014) los autores proponen una arquitectura basada en SDN para redes de sensores, donde el controlador de la red lo despliegan en una base fija en una topología estrella donde se asume que cada nodo sensor cuenta con un enlace directo con el controlador, con ello, el controlador aprende la topología de la red haciendo uso del rastreo de la posición de cada nodo aplicando servicios de ubicación, con la ubicación de los nodos el controlador define las rutas de la red.

3.2 Arquitecturas basadas en SDN para IoT

En la literatura se encuentran muchas propuestas de arquitecturas SDN para redes IoT, se han seleccionado algunas que sirven como referentes para tratar el problema de investigación. En (Ojo et al., 2016), los autores presentan una arquitectura IoT basada en SDN, donde el controlador SDN se encarga de gestionar y administrar los servicios de la red, la capa de red en IoT tiene diferentes acceso a redes, con la finalidad de proporcionar conectividad a los dispositivos a través de redes heterogenias, en la propuesta se emplea el protocolo OpenFlow para la conectividad del controlador al plano de datos con el objetivo de facilitar la comunicación con los nodos finales y con ello mantener la visión general de toda su topología, el controlador cumple con las funciones de actualización de las tablas de ruteo, QoS, movilidad de los nodos y equilibro de tráfico.

Sin embargo, el uso de un solo controlador en casos reales se tornaría un cuello de botella por la cantidad de tráfico que generan las redes IoT, por esta razón (Bizanis & Kuipers, 2016) proponen un esquema distribuido con el uso de múltiples controladores, en su propuesta enfatizan las característica típicas de las redes IoT, por tal razón, toman en consideración la movilidad de los nodos entre puntos de acceso, se basan en las características de una red con áreas geográficas diferentes, donde realizan una subdivisión de la red en sub partes para hacer uso de un controlador en cada área remota, donde el controlador aprende solo la topología de su red, y se evita la sobrecarga a la red. Con este enfoque los autores dan a conocer que se puede solventar la problemática de escalabilidad

de la red, un ejemplo para este tipo de arquitectura son las redes VANET, donde los vehículos se convertirían en los conmutadores SDN, reciben los mensajes desde el controlador para realizar las funciones de enrutamiento.

Otro enfoque en relación con la movilidad de los nodos en la red se presenta en (Wu et al., 2015), donde los autores dan a conocer su propuesta bajo el nombre de UbiFlow, dicha arquitectura se basa en SDN para IoT, hacen uso del concepto de división de la red IoT para mejorar la movilidad, donde manifiestan el control eficiente del flujo y movilidad haciendo uso de subdivisión de la red y aplicando controladores SDN distribuidos, con esta arquitectura lograron que los dispositivos IoT tenga la movilidad entre diferentes puntos de acceso donde los controladores pueden administrar la movilidad y con ello seleccionar el punto de acceso donde se va a conectar el dispositivo.

Otro aspecto importante a considerar es la rigidez que manejan las redes tradicionales, con ello se genera la problemática con la heterogeneidad de redes, en base a este problema, en (Bedhief et al., 2016) proponen la integración de SDN en la arquitectura para manejar la heterogeneidad de los dispositivos IoT, plantea la solución mediante el uso de Docker implementado en cada dispositivo de la red, al Docker lo utilizan para la implementación de aplicaciones necesarias en los nodos, da la posibilidad de agregar o crear aplicaciones sin mayor complejidad sobre los dispositivos. Las validaciones de la propuesta la realizaron mediante la comunicación entre dispositivos a través de la red SDN implementada, hacen uso de Mininet para la implementación de los escenarios, utilizaron un solo controlador centralizado, con ello, lograron evidenciar la conectividad de dispositivos heterogéneos enlazados de diferentes redes de acceso y con diferentes tráficos, con esta arquitectura demostraron la facilidad de agregar, cambiar o dar de baja a nodos dentro de la red, aplicando el concepto de Docker.

3.3 Arquitectura SDN-MANET e IoT

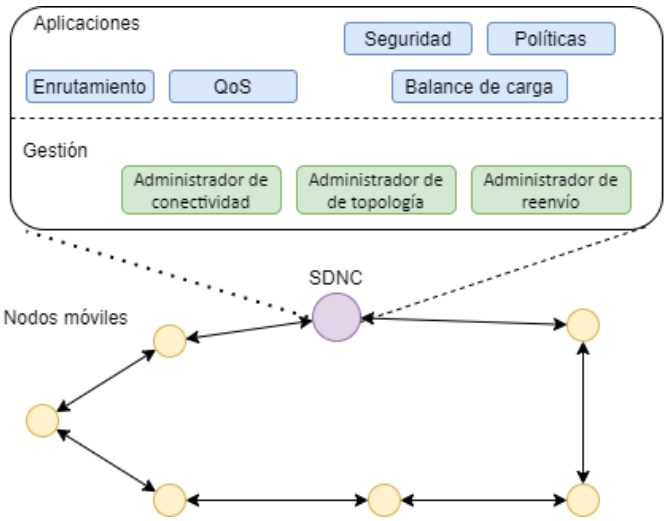
Las consideraciones de diseño para una arquitectura IoT MANET sobre SDN se basan en los criterios redundantes que se manejan en ciertas redes móviles como: mantener una infraestructura para alojar el controlador SDN, manejar enlaces de control de saltos

directos entre nodos y controlador, los servicios de localización para el aprendizaje de la topología por parte de controlador y la conectividad permanente entre la etapa de control. Por ejemplo, si se hace referencia a una arquitectura de estación base LTE para la instalación del controlador SDN, este controlador se convierte en un elemento inmóvil, sin embargo, todos los dispositivos conectados a él son móviles, esto ocasiona que el uso de comunicación a través de un solo salto requiera que el controlador SDN tenga una cobertura de transmisión mejor para cubrir todo el alcance de los nodos móviles.

La Figura 6 se muestra la arquitectura SDN-MANET para IoT, donde uno de los nodos móviles aloja al controlador SDN. Este nodo no está predeterminado y su selección se basa en un procedimiento de elección. Tiene funcionalidades similares a otros nodos de la red, incluido el alojamiento de aplicaciones de datos y la transmisión de paquetes de datos. No hay dispositivos de reenvío dedicados, todos los dispositivos pueden conmutar los paquetes. Es decir, cada nodo actúa como dispositivo de reenvío y como nodo final. Cada nodo tiene un alcance de transmisión inalámbrica limitado, por lo que tanto el control como las comunicaciones de datos se realizan a través de rutas de varios saltos.

Figura 6

Arquitectura SDN-MANET



Nota. Modificado de (Dusia, 2019)

En la arquitectura MANET tradicional, las funciones del plano de datos y del plano de control están agrupadas en cada nodo, y las funciones de enrutamiento dentro de los

nodos se comunican entre sí para seleccionar rutas individualmente. Por el contrario, en la arquitectura SDN-MANET propuesta, que es similar a la arquitectura SDN estándar, la función de enrutamiento se traslada de todos los nodos al controlador SDN. Dado que el controlador tiene la función de enrutamiento, tiene la responsabilidad de seleccionar rutas para todos los nodos de la red. El controlador selecciona las rutas utilizando la vista global de la red, que aprende y mantiene periódicamente la topología de la red. Una forma de aprender la topología de la red es conocer la información de vecindad de cada nodo. Cada nodo necesita una ruta al controlador para enviar su tabla de nodos vecinos. Por lo tanto, para establecer una arquitectura viable de SDN-MANET se requiere de tres funcionalidades dentro de controlador SDN, que se muestran en la Figura 6.

- **Administrador de conectividad:** en una red dinámica, la topología de la red cambia con frecuencia, pero los nodos necesitan aprender y mantener su ruta al controlador para enviar mensajes de control. El Administrador de conectividad ayuda a los nodos a realizar esta tarea.
- **Administrador de topología:** El controlador necesita aprender la topología de la red para seleccionar las rutas. El administrador de topología es responsable de recopilar los mensajes que tienen la información de topología.
- **Administrador de reenvío**: El controlador necesita enviar las actualizaciones de ruta a todos los nodos. Es responsable del reenvío para preparar los mensajes y difundirlos en la red.

Los tres factores ayudan al controlador SDN a administrar la red de manera centralizada y abordar los desafíos tradicionales de MANET: topología de red dinámica, comunicación de múltiples saltos e implementación sin infraestructura y los requisitos de IoT: conectividad, confiablidad en la comunicación, gestión de dispositivos y escalabilidad de red. El protocolo de enrutamiento usado es OLSR compatible con la arquitectura MANET. Además, de la aplicación de gestión de enrutamiento, el controlador también puede trabajar con otras aplicaciones, como balanceador de carga y QoS, que pueden determinar e influir

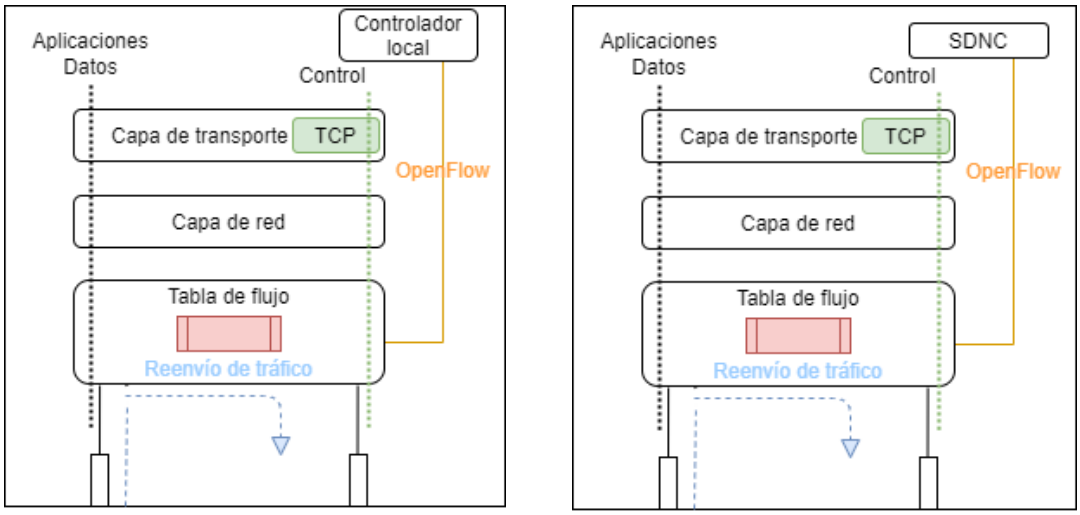
en la selección de la ruta, estas aplicaciones se las puede trabajar como línea de investigación futura y complemento a la propuesta.

3.3.1 Estructura interna del nodo MANET

La Figura 7 se muestra la estructura interna de un nodo SDN-MANET. Cada nodo tiene una tabla de flujo, que es similar a la tabla de flujo de un dispositivo de reenvío habilitado para OpenFlow. Los paquetes de datos se reenvían utilizando las entradas (rutas de red) configuradas en la tabla de flujo.

Figura 7

Estructura interna de los nodos SDN-MANET



a) Nodo con controlador local b) Nodo controlador SDN

Nota. Modificado de (Dusia, 2019)

La función de enrutamiento dentro de un nodo MANET tradicional se comunica con las funciones de enrutamiento en otros nodos para actualizar la tabla de enrutamiento intercambiando mensajes de control típicamente como cargas útiles UDP. En la arquitectura SDN estándar, mostrada en el Capítulo dos, Figura 4, cada dispositivo de reenvío tiene un agente para comunicarse con el controlador usando el protocolo OpenFlow e instalar rutas en la tabla de flujo. La arquitectura SDN-MANET, en la cual cada nodo tiene un agente, llamado controlador local Figura 7 (a), que se comunica con el controlador SDN mediante un protocolo de comunicación. El controlador local mantiene una ruta al controlador SDN

para enviar mensajes de control. La comunicación entre el controlador local y el controlador SDN no utiliza OpenFlow, hace uso del protocolo OLSR. El controlador local convierte la información de enrutamiento enviada por el controlador SDN en mensajes de OpenFlow para instalar rutas en la tabla de flujo. Esta conversión reduce la sobrecarga de comunicación y hace que la red sea programable, lo que permite que la arquitectura facilite varios beneficios de SDN.

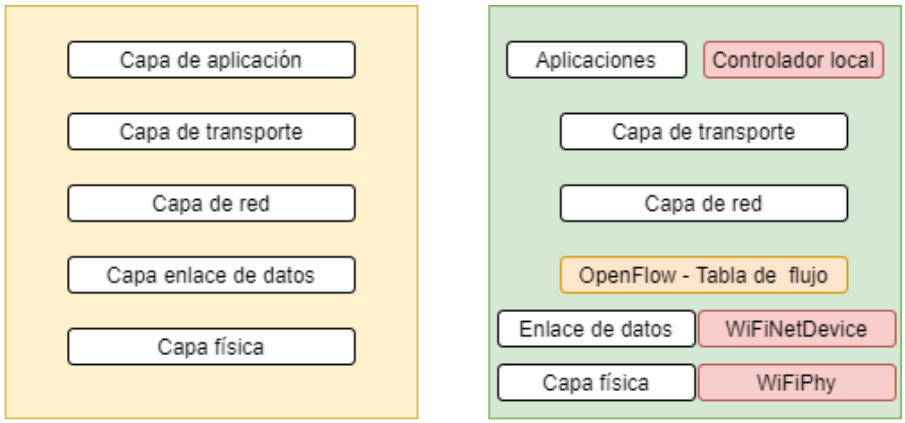
3.3.2 Compatibilidad de SDN con el simulador NS-3

En este apartado se presenta las modificaciones necesarias sobre el simulador NS-3 (referirse al apéndice 1) para la implementación en primera instancia del protocolo OpenFlow (referirse al apéndice 2) y con ello, lograr emular una arquitectura SDN-MANET. El simulador NS-3 tiene un módulo disponible para simulaciones SDN y OpenFlow. Este módulo incluye componentes para un controlador elemental y un conmutador habilitado para OpenFlow que admite la especificación OpenFlow v0.8.9 versión preliminar para simular redes cableadas con enlaces Ethernet entre el host y el conmutador.

La Figura 8 (a) se muestra la pila de IP tradicional de un host en NS-3. Se modifica esta pila para incluir el componente de conmutador OpenFlow entre la capa de red y la capa de enlace como se muestra en la Figura 8 (b). Dado que el componente del conmutador original funciona solo con dispositivos de red Ethernet, se realizan las modificamos para que funcione con dispositivos de red WiFi. Y con ello, se pueda configurar los dispositivos para que funcionen en modo ad hoc, es decir, modo de red sin infraestructura fija y utilizar CSMA para el control de acceso al medio.

Figura 8

Modificaciones a la pila de IP en el simulador ns3



Nota. Modificado de (Dusia, 2019)

3.4 Protocolo de enrutamiento OLSR – MANET

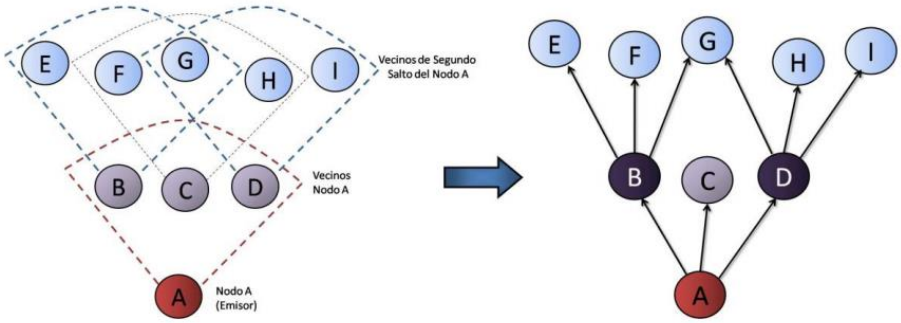
El protocolo de enrutamiento de estado de enlace optimizado (OLSR) se desarrolló para redes móviles ad hoc. Actúa como un protocolo proactivo orientado a tablas. Es decir, intercambia continuamente información de topología con sus vecinos de la red. Cada nodo selecciona un conjunto de nodos adyacentes como relés multipunto (MPR por sus siglas en inglés). En OLSR, solo los nodos seleccionados como MPR son los responsables de reenviar el tráfico de control que se difundirán por toda la red. MPR proporciona un mecanismo eficiente para el control del flujo de inundaciones al reducir el número de transmisiones.

El o los nodos seleccionados como MPR también tiene la responsabilidad de reportar información sobre el estado del enlace en la red. Por tanto, el requisito para que OLSR suministre rutas más cortas a todos los destinos es que el nodo MPR declare la información del estado del enlace. Por lo tanto, los nodos MPR seleccionados por los vecinos, a través de los mensajes de control anuncian de forma periódica esta información. Además, el protocolo usa los nodos MPR para facilitar la transferencia eficiente de mensajes de control a través de la red (Clausen et al., 2003).

Un ejemplo de la selección del nodo MPR se muestra en la Figura 9, todos los nodos a dos saltos de A pueden ser alcanzados a través de B y D. Como resultado, el nodo A elige los nodos B y D como MPR, cuando el nodo A envíe un mensaje de solicitud, solo los nodos B y D reenviarán el mensaje a todos sus vecinos.

Figura 9

Selección del nodo MPR.



Nota. Modificado de (Cavero Miranda & Sánchez Gómez, 2011)

El concepto clave de OLSR es el uso de los nodos MPR. Esta técnica reduce la sobrecarga de mensajes en comparación con otros mecanismos de transmisión, donde cada nodo retransmite cada mensaje cuando recibe la primera copia del mensaje, con esto se logra la primera optimización de protocolo. En OLSR, la información del estado del enlace es creada solo por nodos elegidos como MPR, por tanto, se logra una segunda optimización, minimizando el número de mensajes de control en la red. Como tercera optimización, un nodo MPR puede optar por informar sólo los enlaces entre él y sus selectores MPR, a diferencia del algoritmo de estado de enlace clásico, la información de estado de enlace parcial se distribuye en la red. Luego, esta información es utilizada para el cálculo de la ruta. OLSR proporciona rutas óptimas en términos de número de saltos. El protocolo es particularmente adecuado para redes grandes y densas, ya que la técnica de MPR funciona bien en este contexto (Clausen et al., 2003). En base a estas características de optimización se toma como base al protocolo OLSR para la implementación de la arquitectura SDN-MANET y para validar la propuesta se utiliza el protocolo OLSR para su comparativa en el proceso de enrutamiento y control.

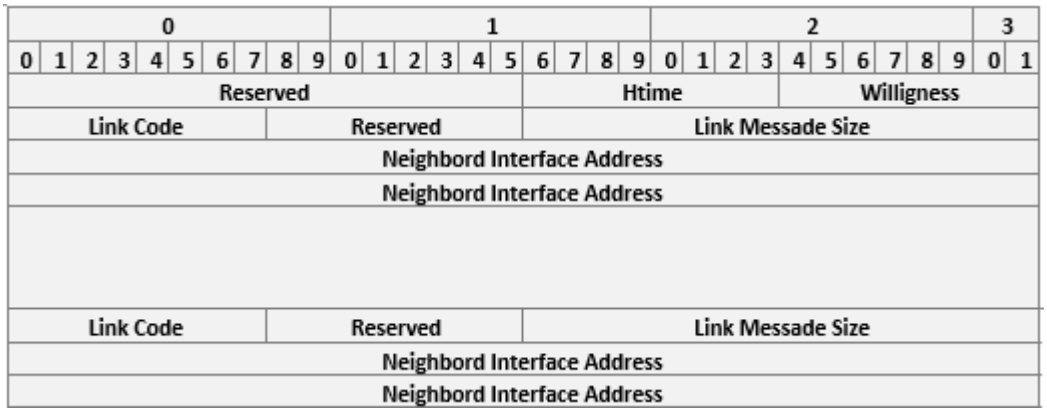
3.4.1 Mensaje de control OLSR

Mensaje HELLO: este paquete de mensaje es el que necesitan los nodos para detección de enlaces, detección de vecindad y señalización de selección de MPR, así como para adaptarse a futuras extensiones. Los mensajes HELLO cuentan con un campo TTL establecido de 1, lo que posibilita la transmiten solamente a un salto, con el objetivo de evitar retransmisión desde el nodo que genera el HELLO, se puede configurar el tiempo para la transmisión continua desde el nodo, en la Figura 10 se muestra el formato del paquete HELLO de OLSR.

Los nodos envían de forma periódica los mensajes HELLO a todos sus vecinos, este mensaje lleva la información de todos los vecinos detectados por el nodo emisor, y adicional, contiene la información de la identidad de los MPR elegidos por el transmisor, este intercambio de mensajes de control, permite a los nodos de la red identificar los nodos ubicados a uno y dos saltos de distancia, aquellos que se puede llegar con una transmisión directa, o haciendo uso de los MPR mediante una transmisión y una retrasmisión, con esto el nodo que retrasmite puede saber si ha sido elegido como nodo MPR por algún vecino.

Figura 10

Mensaje HELLO



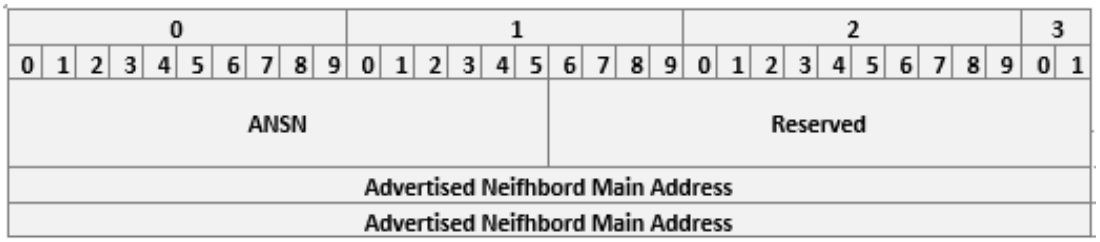
Nota. Modificado de (Clausen et al., 2003)

Mensaje TC: los mensajes de control de cambio de topología (TC) son construídos por todos los nodos de la red, pero solo son reenviados por los nodos seleccionados como MPR. Para cumplir con esta problemática, el campo TTL es igual a 255, y por cada reenvío

se irá restando en una unidad. Este mensaje TC contiene la información de los nodos vecinos que puedan tener el emisor de mensaje, también, la calidad del enlace a cada nodo. En la Figura 11 se muestra el formato de un mensaje TC de OLSR.

Figura 11

Mensaje TC



Nota. Modificado de(Clausen et al., 2003)

La parte de detección de enlaces y de vecinos del protocolo básicamente ofrece TC, a cada nodo, una lista de vecinos con los que puede comunicarse directamente y, en combinación con la parte de formato de paquetes y reenvío, un mecanismo de comunicación optimizado a través de MPR. Sobre esta base, la información de topología se difunde a través de la red (Clausen et al., 2003). Las rutas se construyen a través de enlaces con los vecinos. Un nodo debe al menos difundir enlaces entre sí mismo y los nodos en su conjunto selector de MPR, con el fin de proporcionar información suficiente para permitir el enrutamiento.

Capítulo cuatro

Simulación, análisis y discusión de resultados

La validación del modelo de enrutamiento SDN-MANET se lo realiza mediante un proceso de simulación, para lo cual, se ha adaptado el protocolo de enrutamiento MANET OLSR sobre SDN. El aporte de la propuesta se enfoca en la capa de red para mejorar el proceso de enrutamiento y gestión de red. La implementación se la realizó en el software de simulación NS-3, tomando en consideración que es altamente utilizado en investigación sobre protocolos de enrutamiento MANET, así mismo, es de código abierto y gratuito permitiendo la implementación y modificación de nuevos protocolos o funcionalidades.

Para validar la propuesta, se plantean varios escenarios tomando en consideración la densidad de la red, para ello, se han propuesto la cantidad de: 20, 30, 40, 50 y 100 nodos. Para verificar su eficacia se ha planteado la comparación entre SDN-MANET y OLSR-MANET, utilizando el modelo de movilidad *Random Waypoint*, así mismo, a los dos sistemas de enrutamiento se los simula mediante la modificación del tráfico de red, escenarios con tráfico normal, carga de tráfico y generador de tráfico PPBP (Ammar et al., 2011).

Para implementar el protocolo OLSR en NS-3 se incluyen los archivos con su código fuente y archivos de cabecera, posterior se implementan en el simulador. Estos archivos se los detalla en la Tabla 3.

Tabla 3

Archivos y códigos fuente OLSR

Archivo	Descripción
olsr.cc	En este script están definidos constructores, métodos de envío de paquetes, enrutamiento, direccionamiento, entre otros.
olsr.h	En este script se definen las propiedades y métodos de OLSR.
olsr-routing-protocol.cc	Este script es el principal se encuentra el comportamiento de OLSR.
olsr-routing-protocol.h	Este script documenta la API del módulo OLSR.

olsr-state.cc	Este script implementa todas las funciones necesarias para manipular el estado interno de un nodo OLSR.
olsr-state.h	Este script encapsula todas las estructuras de datos necesarios para mantener el estado interno de un nodo OLSR.

Nota. Esta tabla muestra los achivos y cabeceras para implementa OLSR sobre NS-3

Para la implementación de OpenFlow y SDN se hace uso de la estructura de OLSR, se incluye los archivos con su código fuente de los controladores, se detalla a continuación en la Tabla 4.

Tabla 4

Archivos y códigos fuente OpenFlow

Archivo	Descripción
openflow-switch-helper.cc	En este script se instala los controladores en cada nodo CDMA.
openflow-switch-helper.h	En este script añade la capacidad de conmutar varios segmentos de LAN.

Nota. Esta tabla muestra los achivos y cabeceras para implementa OpenFlow sobre NS-3

El generador de tráfico de proceso de ráfaga de Poisson Pareto (PPBP por sus siglas en inglés) se basa en un proceso de Poisson subyacente que representa puntos en el tiempo en los que cualquiera de un gran número de usuarios comienza a transmitir una ráfaga de tráfico. En el artículo (Zukerman et al., 2003) examinan a PPBP como modelo para el tráfico de Internet siendo muy prometedor en esta función. Así mismo, dan a conocer que PPBP cumple con criterios para un modelo de tráfico simple y preciso. Lo utilizan a PPBP para predecir futuros niveles de eficiencia de multiplexación y enlace dentro de una red. Han demostrado que se puede utilizar eficientemente para futuras aplicaciones del Internet. Por esta razón se utiliza este generador de tráfico para corroborar la eficiencia de la propuesta.

4.1 Proceso de simulación

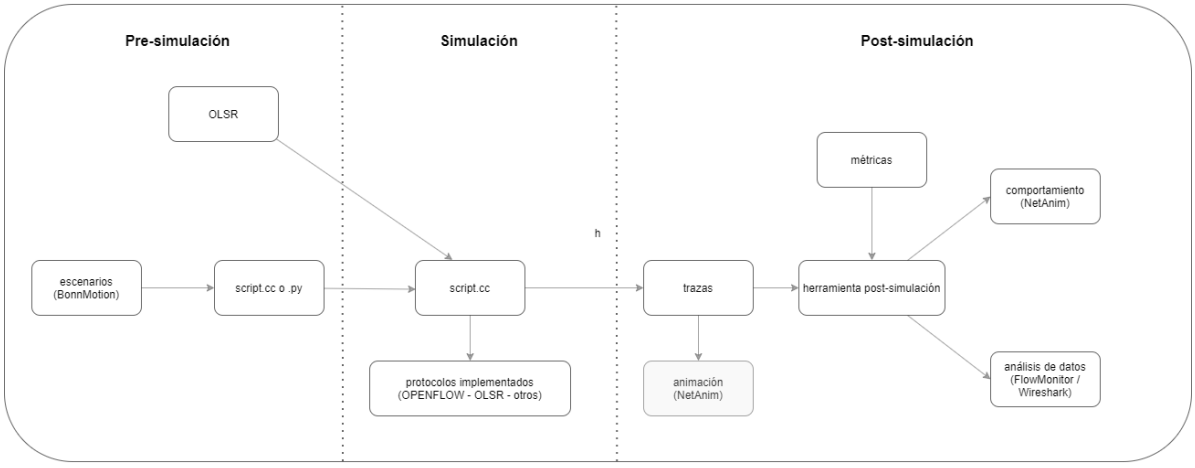
Para llevar a cabo el proceso de simulación se toma como base el proceso que define (Torres, 2011), donde propone tres fases que son: flujo de información, procesos, entradas y salidas, que se ven representadas en la Figura 12. Tomando en consideración que esta

metodología se adapta para el simulador de NS-3 ya que el proceso de simulación (Torres, 2011) hace uso de simulador NS-2.

- Flujo de información: indica la dirección de la información origen al destino, en la Figura 12, se ve representado por las flechas.
- Procesos: los rectángulos de la Figura 12, representan los procesos que se van a ejecutar, cuentan con argumentos de entrada para posterior generar una salida para los demás procesos.
- Entradas y salidas: los rectángulos de la Figura 12, definen la información necesaria para otro proceso o el resultado de algún proceso.

Figura 12

Proceso de simulación en NS-3



Nota. Modificado de (Torres, 2011)

Como se observa en la Figura 12, para realizar la simulación (Torres, 2011) define etapas a seguir donde destaca, la pre-simulación, la simulación y la post-simulación. En la etapa de pre-simulación se realiza un proceso exhaustivo para conocer y especificar los requisitos de entrada de la simulación, se puede mencionar, la edición de los campos del protocolo OLSR como OpenFlow para crear un enrutamiento proactivo MANET a través de SDN, fue necesario analizar la estructura y funcionamiento de la herramienta de simulación NS-3, sin embargo, con el objetivo de verificar el comportamiento de la estructura planteada, se crea los escenarios de simulación donde se destaca la densidad de red y el tráfico de red, para ello, se modificó, el número de nodos, área de simulación, tiempo de simulación,

tamaño del paquete entre otros, todo basado en el modelo de movilidad *Random Waypoint*. En la etapa de simulación, se ejecutan los scripts generados (referirse al apéndice 3, 4, 5) de manera secuencial para obtener los resultados. Por último, en la etapa de post-simulación es donde se va a realizar el análisis de los resultados obtenidos, se lo realizó mediante la generación de archivos con extensión *.pcap* dentro del script de simulación, archivos que contienen diversidad de información para obtener métricas como, paquetes enviados, recibidos, perdidos, el origen y destino del flujo de red, etc. Para el análisis de datos se utiliza un software de análisis de tráfico denominado wireshark (Wireshark, n.d.).

4.2 Definición de escenarios

En esta sección se especifica en forma general los escenarios propuestos para validar el protocolo, véase Tabla 5, se definen las características, atributos de los nodos para ejecutar las pruebas de simulación en 5 escenarios diferentes con tamaños de red de 20, 30, 40, 50 y 100 nodos respectivamente utilizando el protocolo de enrutamiento OLSR-MANET y OpenFlow para la propuesta SDN-MANET, se utiliza un área de simulación de 300 x 300 metros para colocar los nodos de forma aleatoria, el flujo de datos de la conexión es del nodo 6 como emisor al nodo 1 como receptor, aplicando el modelo de movilidad *Random Waypoint* que presenta un comportamiento estable en los rangos de velocidad en un tiempo de simulación de 100 segundos definidos para la simulación. Se utiliza conexión TCP para garantizar la entrega de los paquetes a su destino, estos escenarios fueron creados con la herramienta *Bonnmotion* (BonnMotion, n.d.).

Tabla 5

Escenarios de simulación

Parámetros	Valores
Simulador	NS3
Protocolo	OLSR - OpenFlow
Controlador	LearningController - DropController
Número de nodos	20 - 30 - 40 - 50 - 100
Área de simulación	300 x 300 metros

Transporte	TCP
Ubicación de nodos	Despliegue aleatorio
Modelo de movilidad	Random Waypoint
Tiempo de simulación	100 segundos
Tipo de tráfico	Normal – Carga - PPBP

Nota. Esta tabla muestra los parámetros de configuación de los escenarios

Se evalúan los escenarios tomando en consideración 2 aspectos, cantidad de nodos y tráfico en la red. El parámetro tráfico en la red se lo realiza modificando la carga de tráfico en el canal, tomando en consideración un tráfico normal con el valor de 500Kbps, carga de tráfico un valor de 500Mbps. Para comprobar la carga de tráfico se emplea el generador de tráfico de proceso de ráfaga de Poisson Pareto (PPBP) con el objetivo de encontrar similitudes y comprobar el funcionamiento de la red. Con el parámetro carga de tráfico se satura en canal el flujo entre el emisor y el receptor, mientras que, con el generador de tráfico PPBP se sobrecarga toda la red.

4.3 Análisis de resultados

Para analizar los datos obtenidos se toma como base lo que propone Engineering Task Force (IETF) (Corson et al., 1999), que da a conocer las métricas cuantitativas que se pueden utilizar para validar la eficiencia de los protocolos de enrutamiento en redes móviles. Por lo tanto, la evidencia de los resultados de los protocolos OLSR-MANET y SDN-MANET se los realiza tomando en consideración las métricas de: rendimiento, tasa de envío de paquetes de aplicación, tasa promedio de envío de paquetes, retardo promedio, tasa de retardo de paquetes y porcentaje de pérdida de paquetes. Estas métricas serán utilizadas para comparar los protocolos en los diferentes escenarios planteados.

4.3.1 Rendimiento

El rendimiento por concepto hace referencia a la calidad del servicio de la red desde el cliente. Según (Torres, 2011) define al rendimiento como el total de paquetes recibidos en el destino. El objetivo de medir el rendimiento es para demostrar la eficacia de los protocolos de enrutamiento, tomando en consideración que es más confiable cuan más estable sea en

el tiempo, se lo calcula mediante la fórmula (1), se lo realiza en el nodo destino donde se incluye todos los paquetes que circulan en la red tanto de datos como de protocolos de enrutamiento, etc.

$$R = \frac{Pr}{Ts} \qquad (1)$$

Donde:

R: denota el rendimiento.

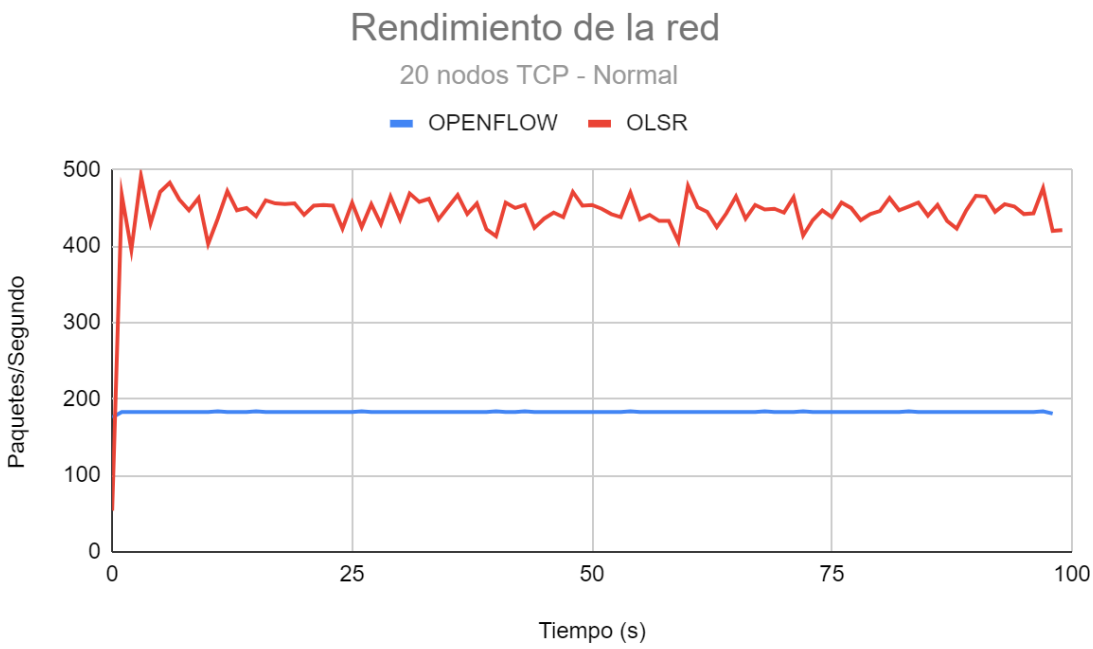
Pr: denota la cantidad de paquetes recibidos en el nodo destino.

Ts: denota el tiempo de simulación del script en segundos.

Se ha medido el rendimiento de la red para conexiones TCP, variando los 2 parámetros de simulación detallados en la sección de escenarios. En la Figura 13, 14 y 15, se muestra el rendimiento de SDN-MANET representada por el protocolo OpenFlow y OLSR-MANET con 20 nodos bajo un modelo de movilidad *Random Waypoint*, donde se puede observar que el protocolo OLSR enviada muchos más paquetes que OpenFlow, considerando que OLSR en nativo para enrutamiento MANET se adapta mucho mejor que las red SDN, su rendimiento es mejor pero cuenta con variaciones caso que en SDN no sucede el rendimiento es estable esto se debe a la lógica e inteligencia que le agregan los controladores desde el plano de control.

Figura 13

Rendimiento con 20 nodos tráfico normal



La validación de carga en la red se puede observar en la Figura 14 y 15 donde el comportamiento del rendimiento es similar cuando la red es sometida a carga de 500Mbps y con el generador de tráfico PPBP. Por lo tanto, OLSR no presentan muchas variaciones en comparación con la Figura 13 (tráfico normal), su rendimiento sigue una tendencia, esto se debe a la pequeña cantidad de nodos que contiene la red. Sin embargo, OpenFlow en la Figura 14 y 15 si presenta mínimas variaciones, por la cargar de tráfico a la red, esto conlleva a pérdida de algunos paquetes que se ven evidenciados en los picos descendentes de la gráfica.

Figura 14

Rendimiento con 20 nodos con carga de tráfico

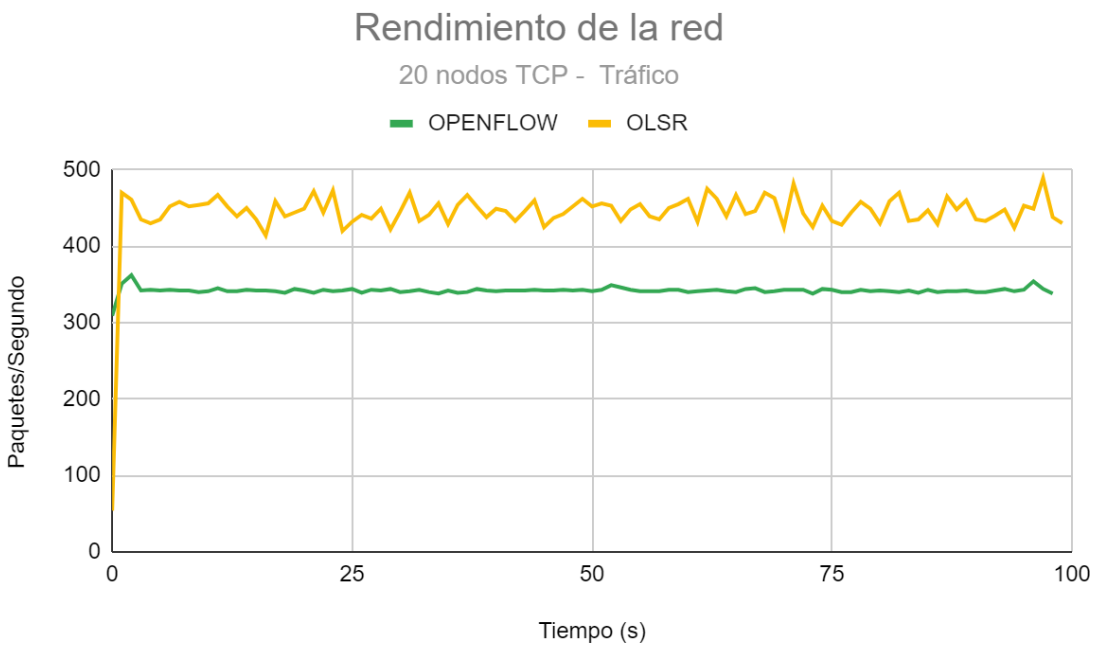
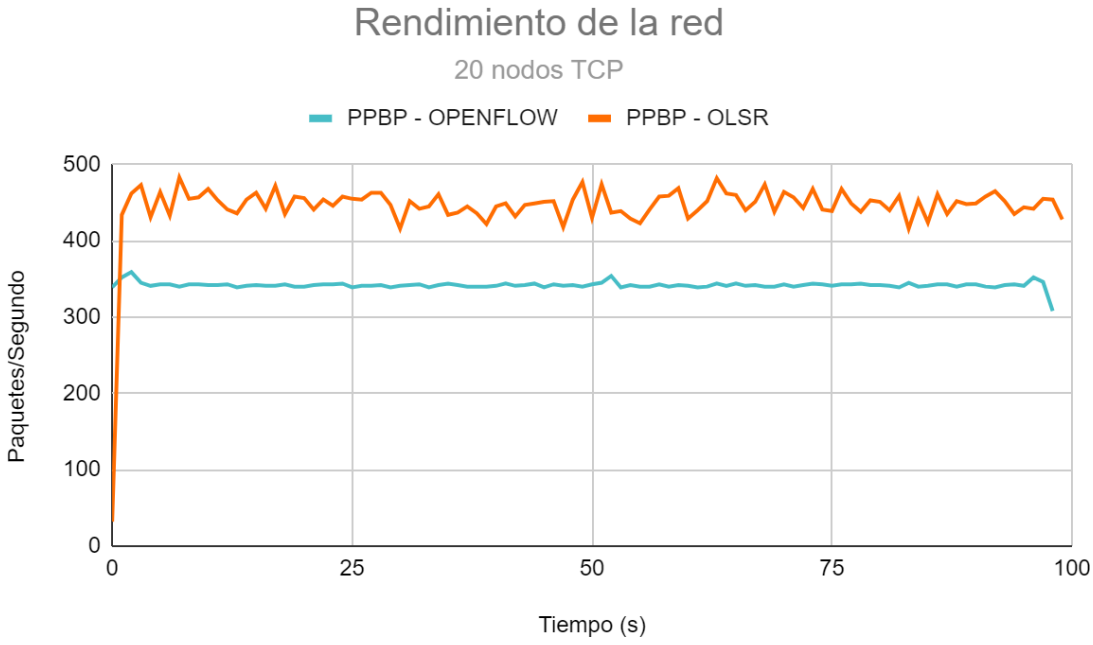


Figura 15

Rendimiento con 20 nodos con tráfico PPBP

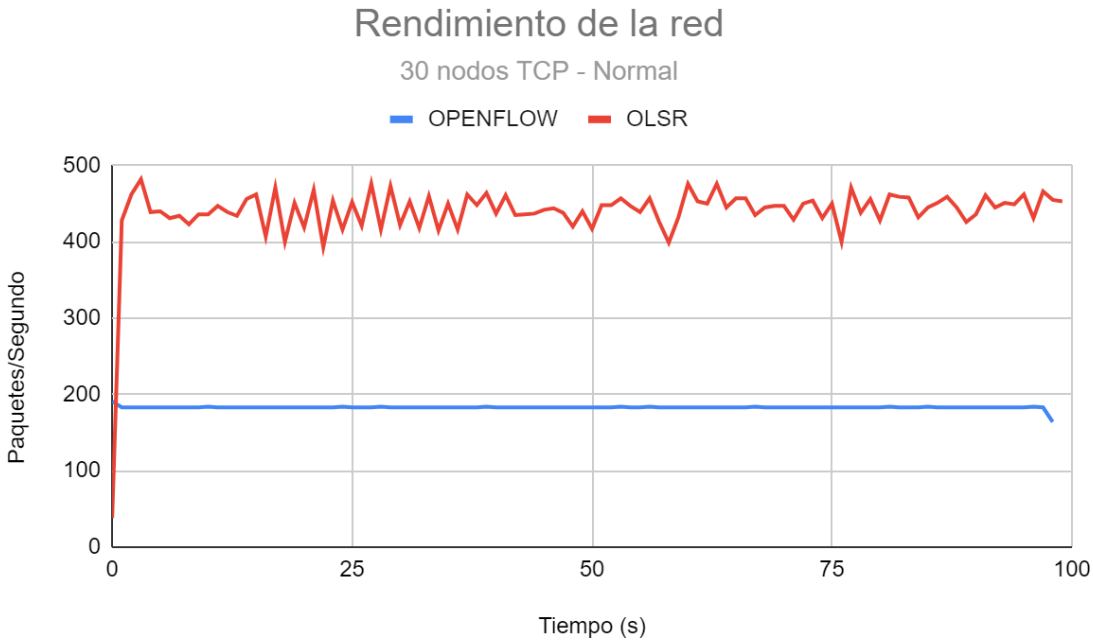


En la Figura 16, 17 y 18 se puede apreciar el rendimiento con 30 nodos, donde se evidencia que en la Figura 16 el protocolo OLSR se ve afectado con un mayor número de

oscilaciones en su gráfica que se puede tornar en muchos casos perjudiciales para una red, sin embargo, SDN mantiene su rendimiento constante siendo menor a OLSR.

Figura 16

Rendimiento con 30 nodos tráfico normal



Cabe denotar que en la Figura 17 en la gráfica del protocolo OpenFlow se aprecia un pico descendente de la red esto se debe a la saturación del canal lo que conlleva que la red pierda paquetes y baja su rendimiento, el tiempo de oscilación es de aproximadamente 3 segundos en el transcurso de 0 a 10 segundos que tarda la red para establecer los nuevos caminos de conexión con los nodos, posterior a este tiempo se puede observar que el rendimiento se mantiene estable, mientras que el protocolo OLSR sigue una similitud con la gráfica de tráfico normal.

En la Figura 18 en el protocolo OpenFlow no sucede el pico descendente, el generador de tráfico afecta a toda la red y no a un flujo determinado como el que se genera con el parámetro carga de tráfico. En este escenario el protocolo OLSR presenta mayor número de variaciones en comparación con la Figura 17 que representa carga de tráfico, esto se debe a la lógica que maneja PPBP el tráfico afecta a toda la red por ende se observa

mayor número de picos descendentes dando como resultado mayor cantidad de paquetes perdidos.

Figura 17

Rendimiento con 30 nodos con carga de tráfico

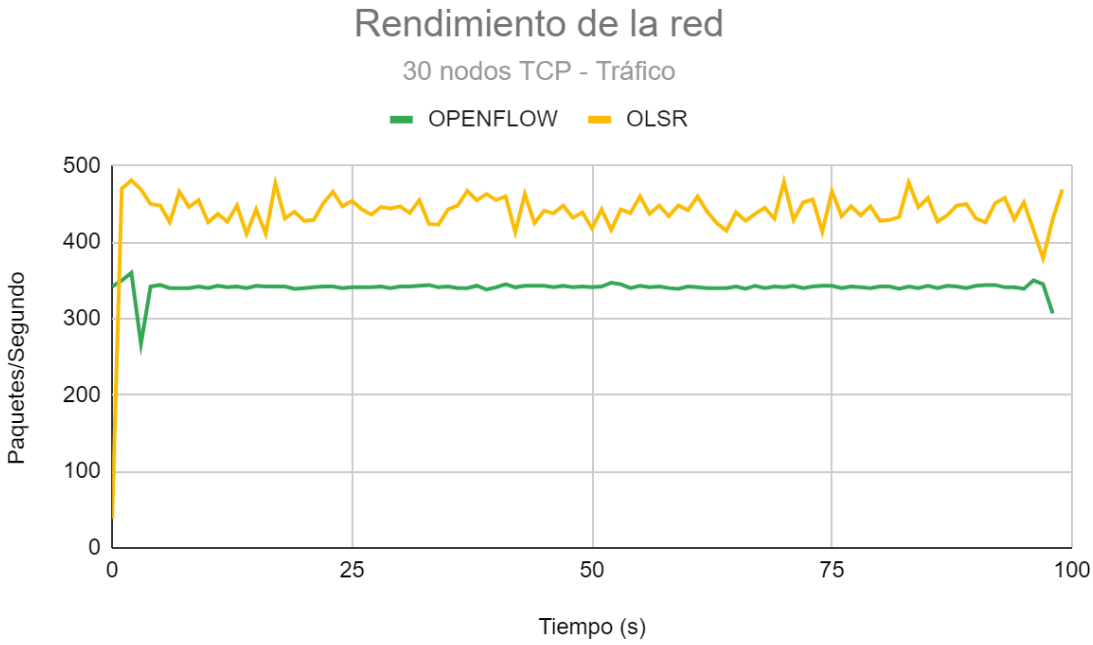
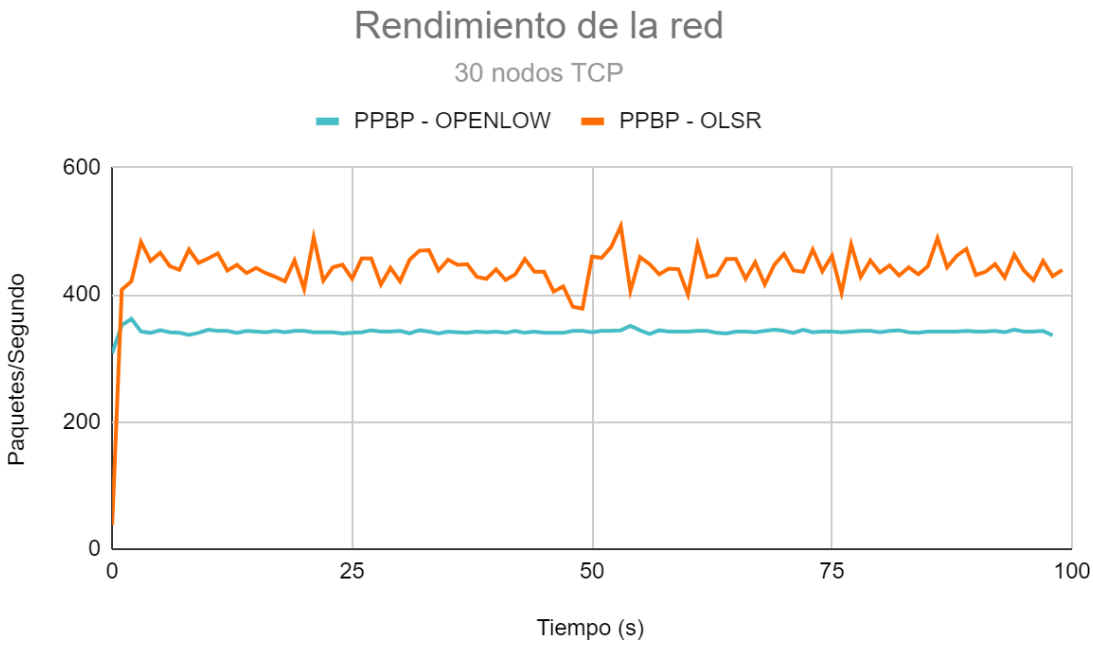


Figura 18

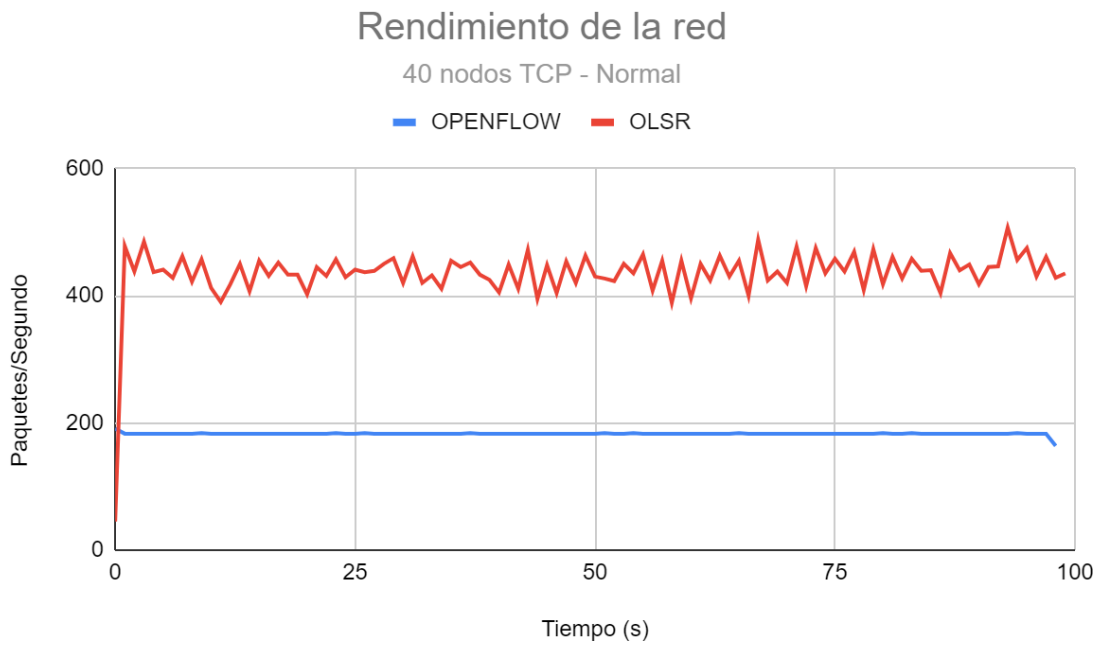
Rendimiento con 30 nodos con tráfico PPBP



El rendimiento de la red con 40 nodos se puede apreciar en la Figura 19, 20 y 21 donde se evidencia que el comportamiento del rendimiento tanto del protocolo OLSR como OpenFlow mantiene una tendencia similar a los escenarios anteriores en cuanto a las oscilaciones de OLSR y la constancia de OpenFlow, tomando en consideración que los escenarios de 20, 30 y 40 nodos son considerados como redes pequeñas de acuerdo con la investigación de (Mishra et al., 2018), por tal razón el comportamiento del rendimiento sigue una tendencia muy similar con ciertas variaciones que se producen por la naturales aleatoria de los nodos y el modelo de movilidad.

Figura 19

Rendimiento con 40 nodos tráfico normal



En la Figura 20 el protocolo OpenFlow denota una caída de rendimiento en el tiempo de simulación de 3 segundos, donde se evidencia gran número de paquetes perdidos, debido a la sobrecarga de tráfico en el canal entre el transmisor y receptor, estos paquetes perdidos son paquetes ACK que ocupa TCP como confirmación de la entrega exitosa, si se saturo el canal estos paquetes se pierden en la red por ende se vuelven a retransmitir, en el segundo 5 el rendimiento se estabiliza. En el caso de OLSR se puede observar que la

sobrecarga del canal se ve reflejada en un rendimiento inestable con mayor número de oscilaciones.

Figura 20

Rendimiento con 40 nodos con carga de tráfico

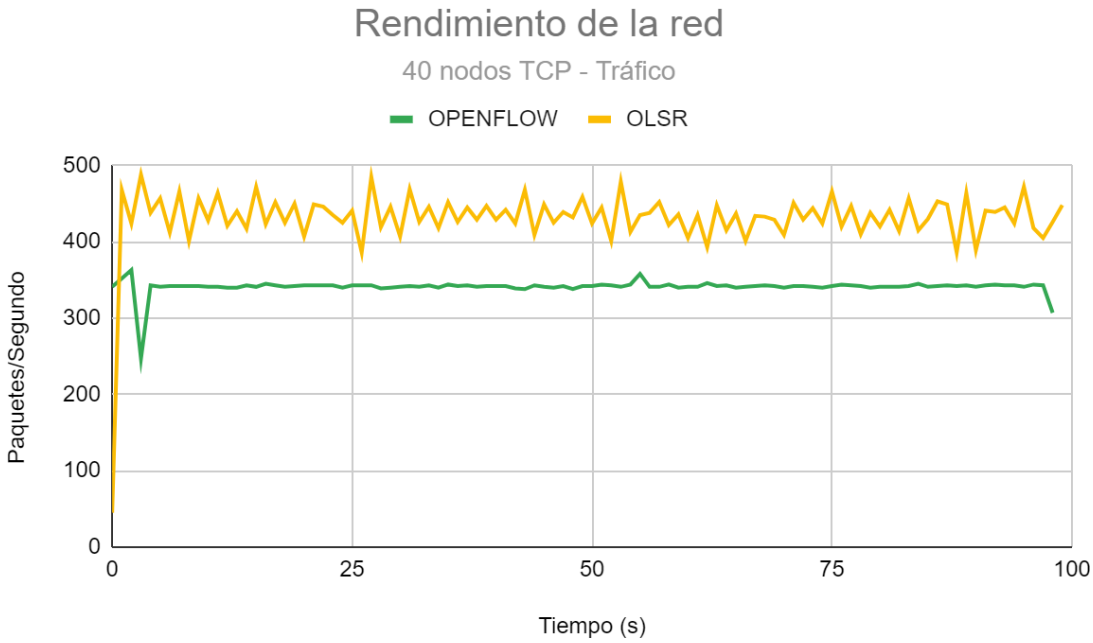
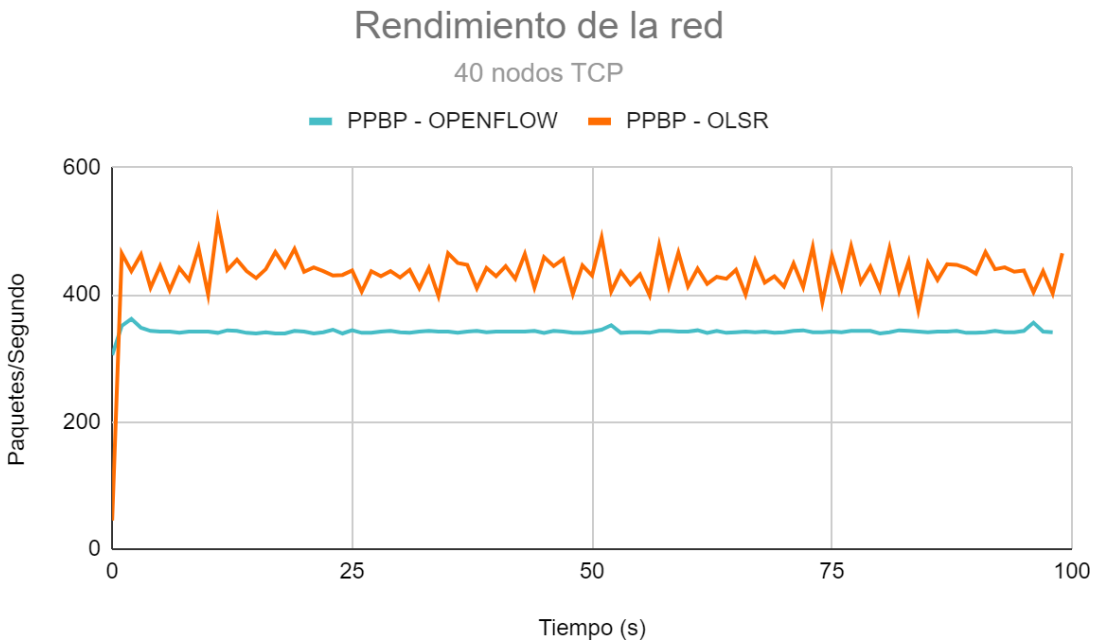


Figura 21

Rendimiento con 40 nodos con tráfico PPBP



En la Figura 21 en el protocolo OpenFlow se evidencia un rendimiento estable favorable para la red MANET, se denota que SDN maneja tráfico de manera eficiente cuando este es administrado a toda la red como se logra bajo el generador de tráfico PPBP, sin embargo, OLSR presenta mayores complicaciones su rendimiento es variable se puede observar que entre el segundo 15 al 60 el rendimiento es mejor.

De acuerdo con (Mishra et al., 2018), las redes grandes se establecen de 50 a 100 nodos, por tal razón, se efectúa una comparación entre el extremo inferior y superior para poder evidenciar el comportamiento de la red con grandes cantidades de nodos. En la Figura 22, 23 y 24 se muestra el rendimiento con 50 nodos, donde se evidencia que el protocolo OLSR en tráfico normal no sufre muchas afectaciones, así mismo, el protocolo OpenFlow se mantiene constante, sin embargo, en la Figura 23 el protocolo OLSR sufre mayores afectaciones en cuanto a las oscilaciones constantes que no favorecen al rendimiento de la red, con ello, también se puede observar que el rendimiento cae un 10% en consideración con los escenarios de redes pequeñas, el protocolo OpenFlow no sufre afectaciones en el rendimiento la tendencia se mantiene similar a los escenarios anteriores.

En la Figura 23 y 24 se presenta el rendimiento de la red con carga de tráfico y generador de tráfico PPBP. Se evidencia que el protocolo OLSR mantiene su tendencia en cuando a las oscilaciones, el envío de paquetes es similar en los dos escenarios de prueba, en la Figura 23 el protocolo OLSR en el lapso de 40 a 45 segundos se evidencia mayor número de envío de paquetes. El protocolo OpenFlow mantiene un rendimiento estable y sigue la tendencia de las redes pequeñas, en la Figura 23 mantiene el pico descendente en el tiempo de simulación de 3 segundos.

Figura 22

Rendimiento con 50 nodos tráfico normal

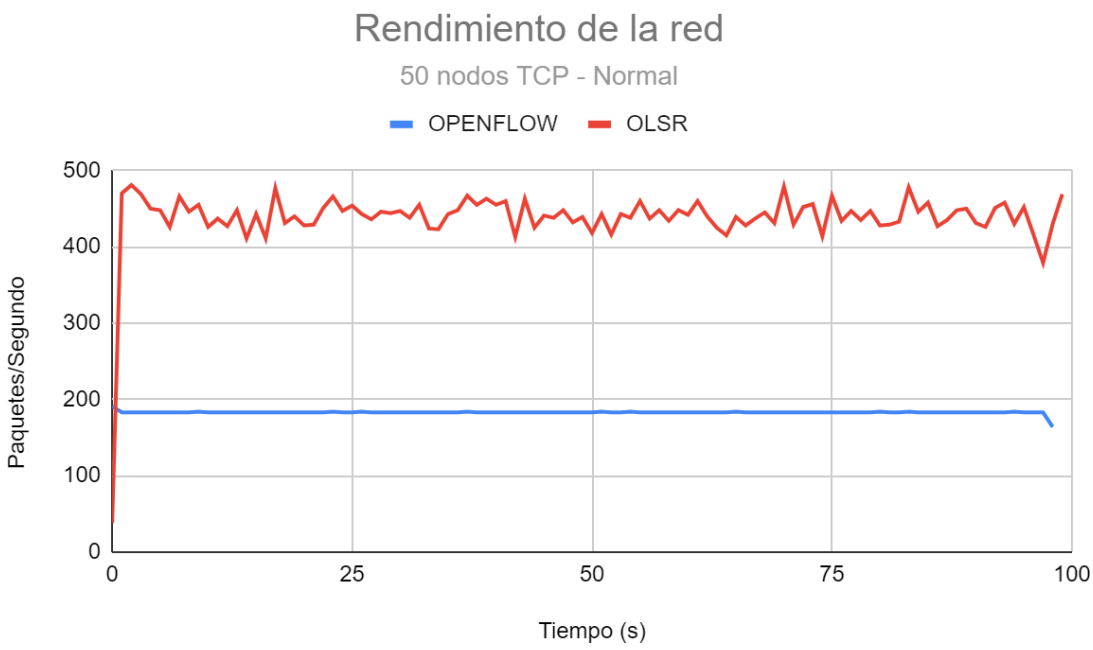


Figura 23

Rendimiento con 50 nodos con carga de tráfico

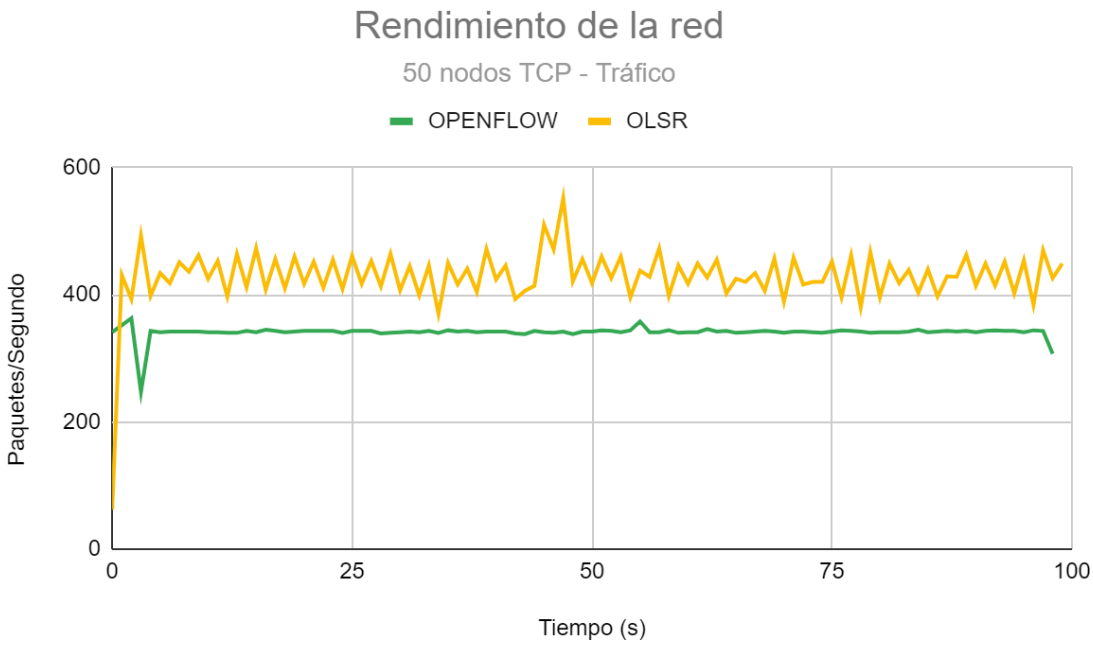
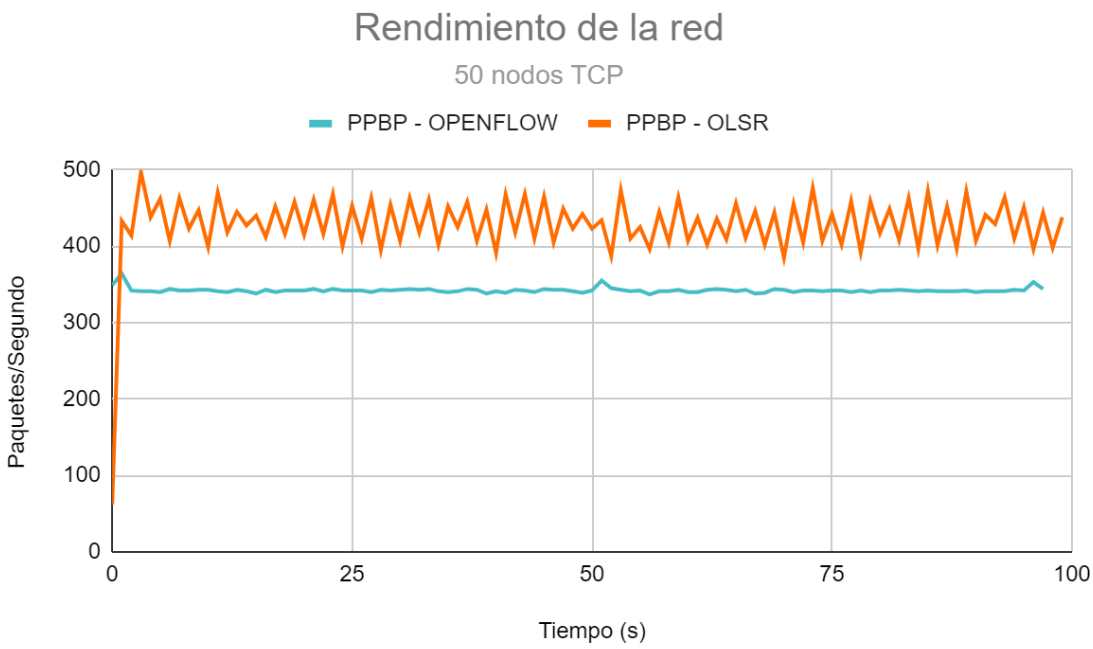


Figura 24

Rendimiento con 50 nodos con tráfico PPBP



Al evaluar el rendimiento con un total de 100 nodos, se evidencia que el rendimiento con tráfico normal (Figura 25), con carga de tráfico (Figura 26) y con generador de tráfico BBPB (Figura 27), en el protocolo OLSR el rendimiento se ve afectado mayormente que en el escenario de 50 nodos, su rendimiento cae de forma drástica las oscilaciones aumentan generando mayor pérdida de paquetes y por ende la eficacia de la red baja un 40% aproximadamente.

Sin embargo, el protocolo OpenFlow para la red SDN-MANET mantiene su rendimiento y en el caso de carga de tráfico pierde el pico descendente debido al mayor número de nodos, donde el controlador a través de la gestión puede encontrar muchos mas caminos hacia el nodo destino, como resultado se evidencia que el rendimiento de SDN-MANET en redes pequeñas y grandes sige una tendencia y su rendimiento es constante, no ocupa demasiados paquetes de control para la entregar de los paquetes de aplicación, sin embargo, el protocolo OLSR-MANET se ve afectado de manera significativa con el aumento de nodos y sobre todo con carga a la red dando como resultado que para redes grandes bajo los escenarios planteados su rendimiento es menor a SDN-MANET.

Figura 25

Rendimiento con 100 nodos tráfico normal

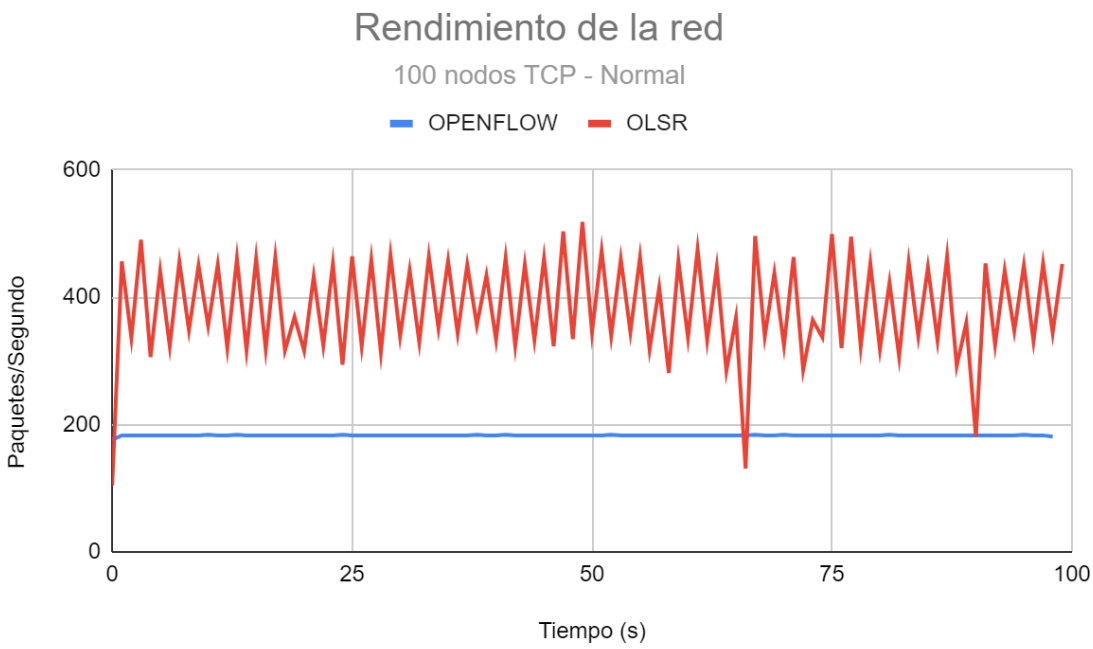
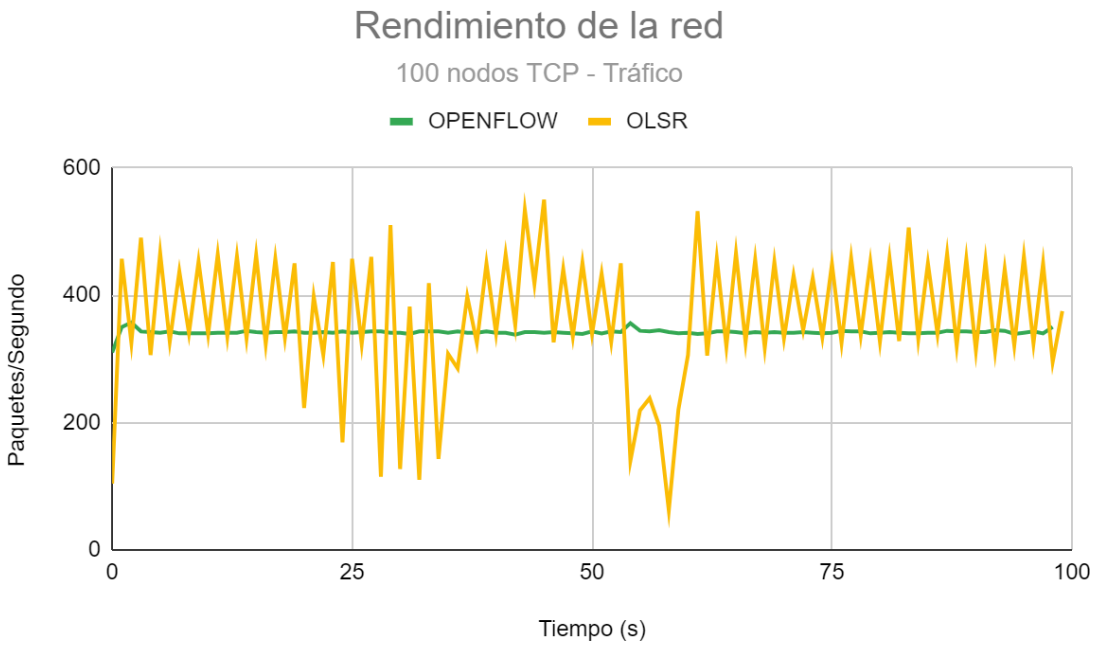


Figura 26

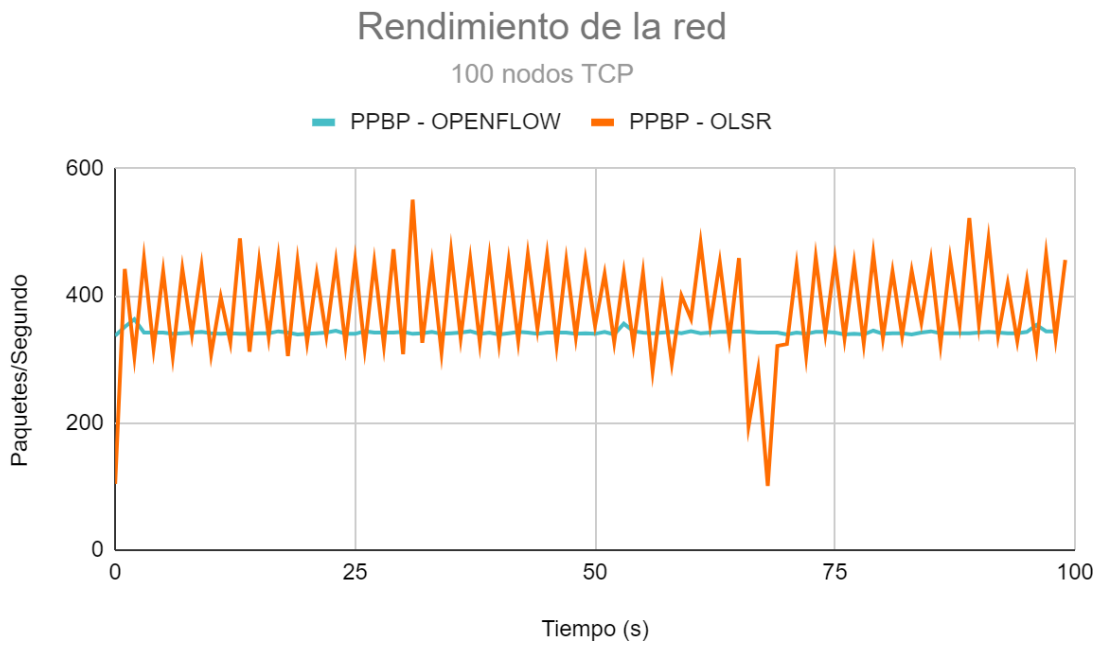
Rendimiento con 100 nodos con carga de tráfico



Es importante analizar los puntos críticos que se evidencian en las gráficas de OLSR en los tres parámetros de prueba. En la Figura 25 se observa una caída de rendimiento en el segundo 66 y 90 de simulación, estos picos descendentes se ven reflejados en la pérdida de paquetes de enrutamiento (paquetes OLSR) esto se debe a la movilidad aleatoria de los nodos. En la Figura 26 en el lapso de 54 a 58 segundos se denota una caída mayor del rendimiento de red donde se pierden paquetes de aplicación, paquetes de enrutamiento y paquetes ACK, esto se debe a la saturación del canal. En la Figura 27 se evidencia una caída de redimiendo similar a la Figura 26 con un ligero desplazamiento del lapso de 66 a 68 segundos, de igual forma como resultado en este tiempo de simulación pierde mayor número de paquetes de aplicaciones, enrutamiento y ACK.

Figura 27

Rendimiento con 100 nodos con tráfico PPBP



4.3.2 Tasa de envío de paquetes de aplicación

De acuerdo con el concepto de (Syarif & Sari, 2011), la tasa de envío de paquetes es la relación entre los paquetes entregados al nodo destino y la cantidad de paquetes emitidos desde el nodo origen, esta métrica se la utiliza para analizar el envío y recepción de paquetes de un protocolo de enrutamiento. La fórmula 2 se utiliza para el cálculo

matemático, misma que especifica que la tasa de envío de paquetes se divide entre el total de paquetes recibidos para los paquetes enviados. En relación con el script de prueba los paquetes de aplicación son emitidos por el nodo 6 y receptados por el nodo 1, esta métrica es mejor cuando el valor obtenido de la fórmula 2 se acerca a 1, por consiguiente, si se tiene un cálculo de 1 se puede mencionar que todos los paquetes generados por el nodo emisor son entregados satisfactoriamente al nodo receptor.

$$TEP = \frac{Pr}{Pe} \qquad (2)$$

Donde:

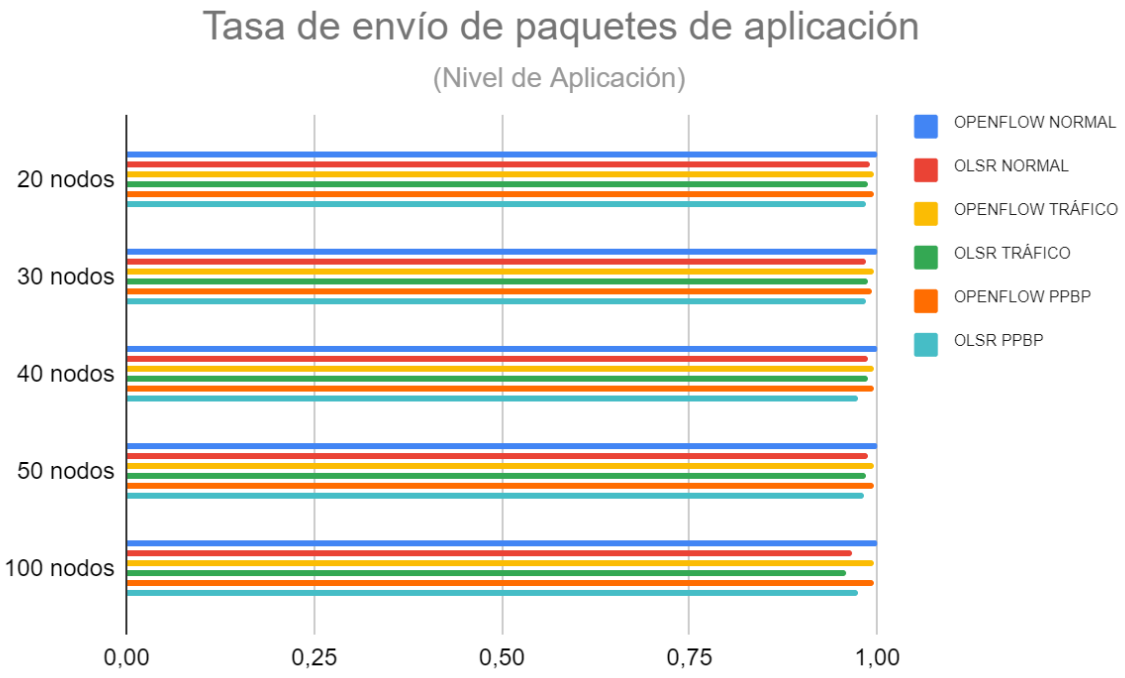
TEP: tasa de envío de paquetes,

Pr: paquetes de aplicación recibidos

Pe: paquetes enviados.

Figura 28

Tasa de envío de paquetes de aplicación con tráfico, normal, carga y PPBP



En la Figura 28 se puede evidenciar la tasa de envío de paquetes a nivel de aplicación, esta métrica es evaluada bajo 2 aspectos, cantidad de nodos y tráfico en la red, dando como resultado que a mayor cantidad de nodos y tráfico en la red el protocolo OLSR pierde mayor cantidad de paquetes, debido a la movilidad de los nodos y el descubrimiento de la topología, mientras que OpenFlow cuenta con una eficacia mayor sin pérdida de paquetes en los dos aspectos, cantidad de nodos y tráfico en la red, el protocolo OpenFlow mantiene un comportamiento similar en todos los escenarios, logra su eficacia gracias a la gestión de los controladores SDN que permiten gestionar la topología de una manera global, por lo tanto, el controlador conoce de todos los nodos conectados en la red con el objetivo de llevar los paquetes generados a su destino.

4.3.3 Tasa promedio de envío de paquetes

El promedio de paquetes recibidos se calcula entre los paquetes de datos enviados para el total de paquetes de datos recibidos. Mientras más cercano a 1 es mejor. El cálculo se lo realiza a través de la fórmula 3.

$$Tpr = \frac{pde}{dr} \qquad (3)$$

Donde:

Tpr: total de paquetes recibidos.

pde: total de paquetes de datos enviados

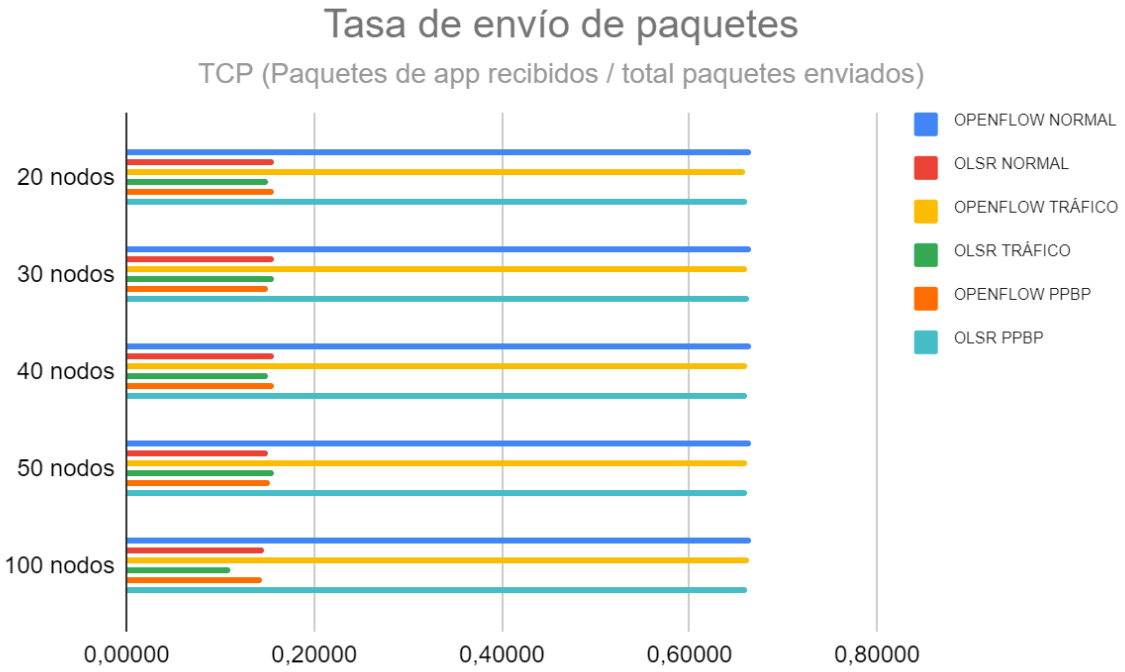
dr: datos recibidos

En la Figura 29 se muestra la tasa promedio de envío de paquetes, donde, la tasa de envío del protocolo OpenFlow es mayor al protocolo OLSR. OpenFlow a través de SDN separa el plano de datos del plano de control, logrando tener una gestión controlada desde los controladores SDN, en los escenarios se puede evidenciar una tendencia positiva de la tasa de envío de la propuesta SDN-MANET planteada, mientras que, OLSR experimenta una tasa menor debido al protocolo, y descubrimiento de vecinos, mismo que conlleva complicaciones por la movilidad de los nodos, con ello, se puede evidenciar que necesitan más paquetes de control para el descubrimiento de la topología generando sobrecarga a la

red que se repercute en la pérdida de paquetes. Mientras mayor es la densidad de la red y tráfico en la misma, menor es la tasa de envío de paquetes.

Figura 29

Tasa de envío de paquetes de datos con tráfico, normal, carga y PPBP



4.3.4 Retardo promedio

El retardo se lo conceptualiza mediante el tiempo entre la generación de paquetes del nodo origen a un nodo destino con entrega exitosa, es una métrica significativa para corroborar el tiempo desde el envío y recepción de paquetes. Por lo tanto, es el tiempo de un paquete que se transmite sobre la red. Para calcular este parámetro se utiliza la fórmula 4, que indica la diferencia del tiempo de recepción del paquete en el nodo destino (nodo 1) menos el tiempo de generación del paquete en el nodo de origen (nodo 6). Este parámetro puede definir si el protocolo es eficiente.

$$Ad = \frac{Sd}{Pr} \qquad (4)$$

Donde:

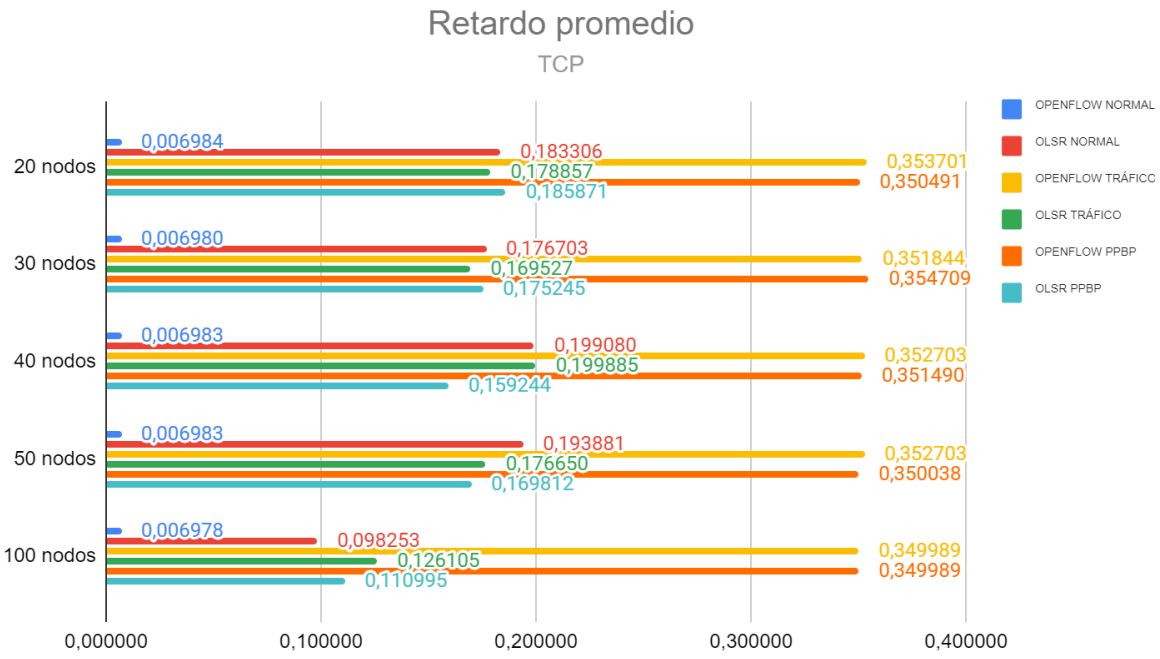
Ad: retardo promedio.

Sd: suma de retardo.

Pr: paquetes recibidos.

Figura 30

Retardo promedio con tráfico, normal, carga y PPBP



En la Figura 30 se observa el retardo promedio entre OLSR y OpenFlow con tráfico normal, carga de tráfico y generador de tráfico, donde se evidencia que OLSR tiene un retardo menor a OpenFlow, excepto en los escenarios donde OpenFlow maneja un tráfico normal. El retardo en OpenFlow se ve reflejado por el trabajo que deben realizar los controladores sobre los nodos, por lo tanto, conlleva mas tiempo para enviar cantidad de tráfico, OLSR lo hace más rápido porque busca sus nodos vecinos, crea sus tablas de ruteo hasta llegar al nodo destino. Sin embargo, ambos protocolos tienen un retardo bastante bajo, que no puede definir la eficiencia del protocolo.

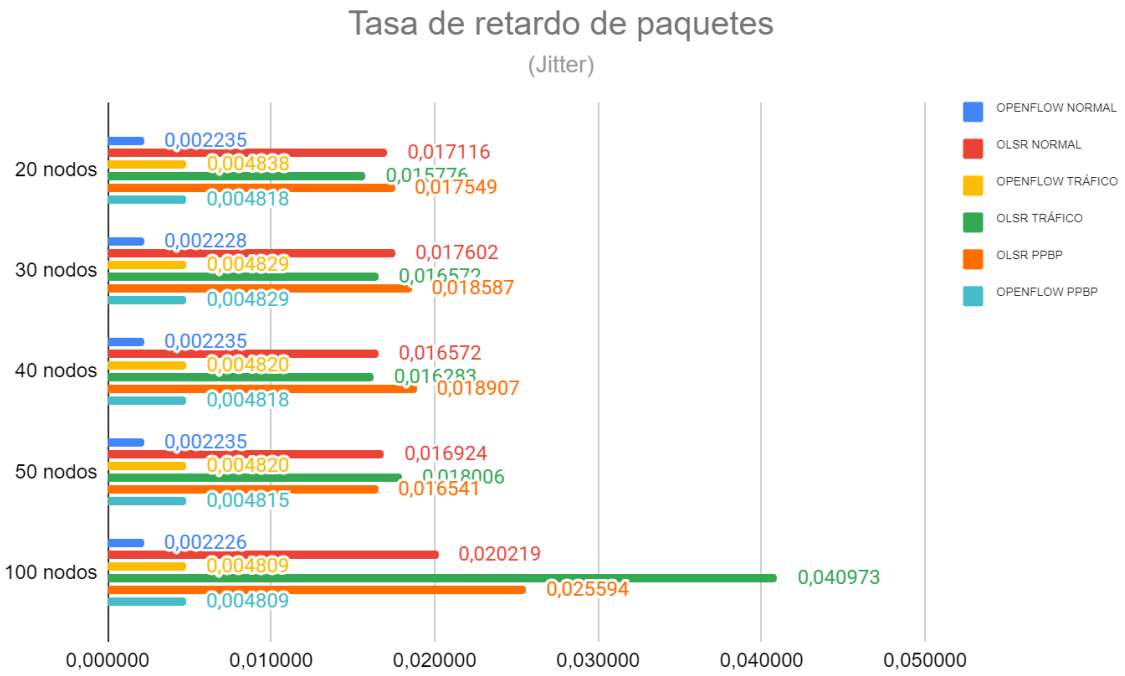
4.3.5 Tasa de retardo de paquetes

También conocido como *jitter*, se considera como la diferencia del retardo de la comunicación entre el origen y destino de un flujo seleccionado, por lo tanto, mientras el jitter tenga valores más cercanos a 0 la red es más estable, esta métrica se la obtiene del nodo destino (nodo 1).

Esta métrica permite medir la convergencia y estabilidad de la red móvil, en la Figura 31 se muestra la tasa de retardo de paquetes en el protocolo OLSR y OpenFlow, con tráfico de red, normal, carga y generador BBPB. El protocolo OpenFlow presenta menor retardo, sus valores se acercan a 0, por ende, la red bajo estas condiciones de densidad de nodos y tráfico es estable en todos los escenarios, mientras que el protocolo OLSR presenta un mayor retardo, que puede indicar que la comunicación no es estable entre el nodo inicio y el destino. Es menester mencionar, que mientras se aumenta la densidad de la red y la carga de tráfico el protocolo OLSR tendrá mas retardo.

Figura 31

Tasa de retardo de paquetes con tráfico, normal, carga y PPBP



4.3.6 Porcentaje de pérdida de paquetes

Por definición, según (Torres, 2011) es la cantidad de paquetes eliminados debido a: el movimiento de los nodos, el expirar los temporizadores, destinos están fuera de rango o paquetes borrados por el protocolo ARP. Los datos fueron obtenidos del nodo destino (nodo 1). Para el cálculo del porcentaje de pérdida de paquetes se multiplica los paquetes perdidos por 100 todo dividido para los paquetes enviados, usando la fórmula 5.

$$\%Pp = \frac{Pp*100}{Pt}\% \qquad (5)$$

Donde:

$\%Pp$: Porcentaje de pérdida de paquetes,

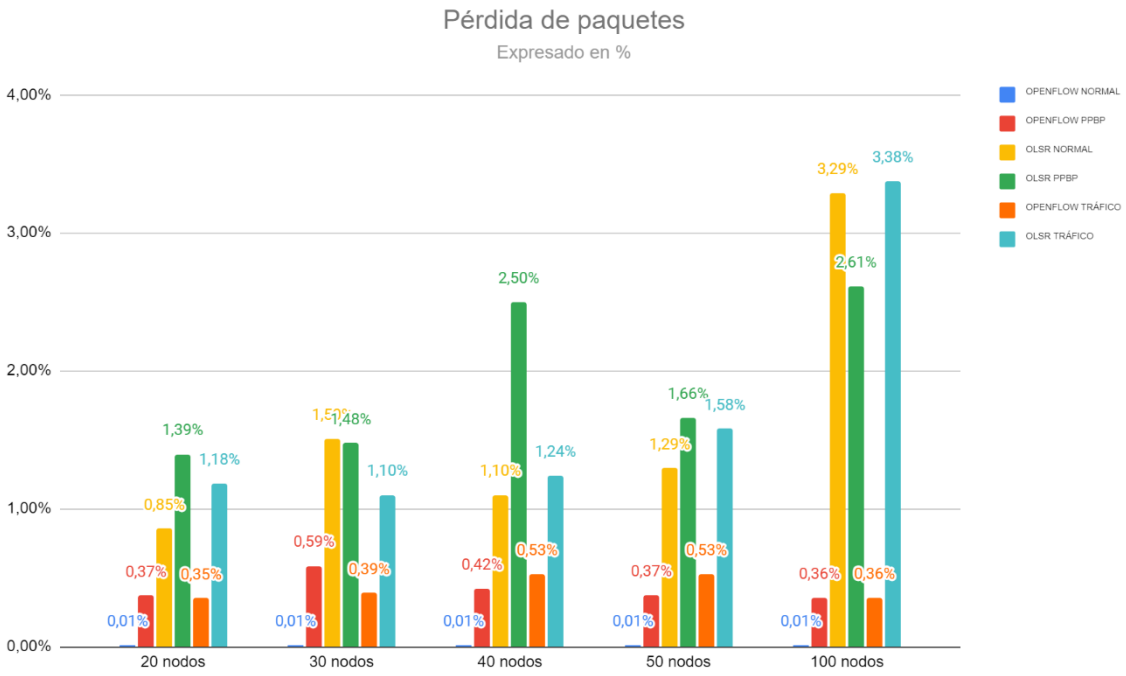
Pp: Paquetes de datos perdidos

Pt: Paquetes de datos enviados

En la Figura 32 se muestra el porcentaje de pérdida de paquetes de los protocolos OLSR y OpenFlow en los diversos escenarios propuestos. Tomando en consideración que la red MANET con el protocolo OLSR su rendimiento es menor a la red SDN-MANET. Con esta métrica se puede constatar que el protocolo OLSR pierde mayor cantidad de paquetes en todos los escenarios de prueba, debido a la característica de la red móvil, el no contar con una infraestructura o topología de red fija, el protocolo sobrecarga la red con mensajes de control para la búsqueda del destino, afectado por la movilidad aleatoria de los nodos y por ende pérdida de paquetes. Sin embargo, SDN maneja la lógica e inteligencia de la red, el protocolo OpenFlow tiene mayor retardo en la entrega de paquetes, esta métrica de retardo promedio beneficia positivamente en la entrega exitosa de los paquetes, por lo tanto, la pérdida de paquetes en menor. Los controladores son los que se encargan de verificar y controlador a los nodos dentro de la red, crean las posibles rutas para que los nodos conmutadores envíen los paquetes hacia el destino de forma exitosa.

Figura 32

Pérdida de paquetes con tráfico, normal, carga y PPBP



Conclusiones

Luego de la revisión de literatura de las arquitecturas propuestas para SDN-MANET, la descripción de la solución propuesta, la verificación a través de simulación y la presentación de los resultados obtenidos, en esta sección se presentan las conclusiones de proyecto de investigación.

Como producto de esta investigación, se describió y discutió una propuesta de red que mostró cómo se puede aplicar SDN en MANET. Además, el documento presenta una implementación a través de la herramienta de simulación NS-3. La solución propuesta se centraliza en la modificación de la capa de red, donde se propone un enfoque de enrutamiento proactivo tomando como base al protocolo OLSR, cada nodo en la red se modifica con el objetivo de desacoplar el plano de control con el plano de datos, donde se hace uso de protocolo OpenFlow para la comunicación. De acuerdo con este enfoque, los nodos MPR de la red serán los encargados de alojar el controlador SDN. OLSR se utiliza en la fase inicial cuando toda la topología y la información del canal se envía al controlador. Las decisiones de enrutamiento para toda la red se toman en el controlador con OpenFlow y las reglas de enrutamiento se distribuyen a todos los nodos de la red. Si la topología cambia, OLSR se usa nuevamente para enviar la información de red actualizada a los controladores.

Para validar la propuesta se generaron varios escenarios tomando en consideración, densidad de red y carga de tráfico con el objetivo de evaluar la propuesta SDN-MANET. Donde se demostró mediante los resultados y análisis de los datos obtenidos que, el rendimiento, la tasa de envíos de paquetes de aplicación, la tasa promedio de envío de paquetes, el retardo promedio, la tasa de retardo de paquetes, la pérdida de paquetes, son comparables con los de OLSR para redes pequeñas de aproximadamente 40 nodos, mientras que estas métricas siguen una tendencia positiva en SDN-MANET propuesta, los resultados dan a conocer que para redes grandes de aproximadamente 50 a 100 SDN-MANET es mejor que OLSR.

El rendimiento del protocolo OLSR se ve directamente afectado por la cantidad de nodos o densidad de red y la carga de tráfico, a mayor cantidad de nodos y mayor tráfico el rendimiento cae, para redes grandes de 50 a 100 nodos el rendimiento decrece en un 40% aproximadamente en comparación con las redes pequeñas, la caída del rendimiento afectada a las demás métricas de evaluación. El rendimiento del protocolo OpenFlow es constante para tráfico normal y generador de tráfico, se ve afectado cuando se lo evalúa con carga de tráfico causando afectaciones al rendimiento por el pico descendente generado en el segundo 3 de la simulación. Es importante destacar que el retardo en la entrega de paquetes de OpenFlow es mayor a OLSR, razón por la cual, OpenFlow tiene una demora adicional de 0.15 segundos aproximadamente por el análisis que deben realizar los controladores para la entrega de los paquetes.

Recomendaciones

Después de haber realizado el presente proyecto de investigación, y los conocimientos adquiridos duranta la ejecución, se recomienda:

Para la instalación de NS-3 y la ejecución de SDN, bajo el protocolo OpenFlow, se requiere un ordenador con características y recursos mínimos 16GB RAM, 1TB HDD, procesador CORE i7 o superior, por la densidad de red (cantidad de nodos) y la carga de tráfico elevada, conlleva, a la necesidad de recursos de ordenador, tomado en consideración que bajo estas caracterizas y para la simulación de 100 nodos bajo el protocolo OpenFlow se obtuvo una demora entre un rango de 2 a 5 horas.

Para obtener mejores resultados y sobre todo potenciar al máximo la herramienta de simulación NS-3, es fundamental que se opere con una herramienta para emulación de escenarios, como es *BonnMotion*, y herramientas para la obtención de datos como lo es *FlowMonitor*, y los archivos *pcap* que ayuda a captar las métricas para el posterior análisis de los datos.

Seria de gran importancia evaluar la propuesta implementada bajo el protocolo de transporte UDP, para corroborar su rendimiento y confiablidad de comunicación de red MANET basada en redes definidas por software para IoT.

Referencias

Ade, S. A., & Tijare, P. A. (2010). Performance comparison of AODV, DSDV, OLSR and DSR routing protocols in mobile ad hoc networks. *International Journal of Information Technology and Knowledge Management*, *2*(2), 545–548.

Aktas, T., Quer, G., Javidi, T., & Rao, R. R. (2016). From connected vehicles to mobile relays: Enhanced wireless infrastructure for smarter cities. *2016 IEEE Global Communications Conference (GLOBECOM)*, 1–6.

Alam, M., Nielsen, R. H., & Prasad, N. R. (2013). The evolution of M2M into IoT. *2013 First International Black Sea Conference on Communications and Networking (BlackSeaCom)*, 112–115.

Ammar, D., Begin, T., & Guerin-Lassous, I. (2011). A new tool for generating realistic internet traffic in ns-3. *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, 81–83.

Aslam, M., Hu, X., & Wang, F. (2017). Sacfir: Sdn-based application-aware centralized adaptive flow iterative reconfiguring routing protocol for wsns. *Sensors*, *17*(12), 2893.

Basagni, S., Conti, M., Giordano, S., & Stojmenovic, I. (2004). *Mobile ad hoc networking*. John Wiley & Sons.

Bedhief, I., Kassar, M., & Aguili, T. (2016). SDN-based architecture challenging the IoT heterogeneity. *2016 3rd Smart Cloud Networks & Systems (SCNS)*, 1–3.

Bellavista, P., Dolci, A., Giannelli, C., & Montenero, D. D. P. (2020). SDN-based traffic management middleware for spontaneous WMNs. *Journal of Network and Systems Management*, *28*(4), 1575–1609.

Bhardwaj, P. K., Sharma, S., & Dubey, V. (2012). Comparative analysis of reactive and proactive protocol of mobile ad-hoc network. *International Journal on Computer Science and Engineering*, *4*(7), 1281.

Bizanis, N., & Kuipers, F. A. (2016). SDN and Virtualization Solutions for the Internet of Things: A Survey. In *IEEE Access* (Vol. 4, pp. 5591–5606). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2016.2607786

BonnMotion. (n.d.). *BonnMotion - A mobility scenario generation and analysis tool*. Retrieved August 1, 2021, from http://sys.cs.uos.de/bonnmotion/

Cavero Miranda, J., & Sánchez Gómez, F. (2011). *Banco de pruebas para redes ad-hoc*. Universitat Politècnica de Catalunya.

Chiliquinga Rodríguez, C. S. (2020). *Internet de las cosas basado en redes definidas por software*. Universidad Técnica de Ambato. Facultad de Ingeniería en Sistemas ….

Clausen, T., Jacquet, P., Adjih, C., Laouiti, A., Minet, P., Muhlethaler, P., Qayyum, A., & Viennot, L. (2003). *Optimized link state routing protocol (OLSR)*.

Correia, S., Boukerche, A., & Meneguette, R. I. (2017). An architecture for hierarchical software-defined vehicular networks. *IEEE Communications Magazine*, *55*(7), 80–86.

Corson, S., Papademetriou, S., Papadopoulos, P., Park, V., & Qayyum, A. (1999). *An Internet MANET encapsulation protocol (IMEP) specification, draft-ietf-manet-imep-spec02. txt*. IETF.

De Gante, A., Aslan, M., & Matrawy, A. (2014). Smart wireless sensor network management based on software-defined networking. *2014 27th Biennial Symposium on Communications (QBSC)*, 71–75.

De Rango, F., Fotino, M., & Marano, S. (2008). EE-OLSR: Energy Efficient OLSR routing protocol for Mobile ad-hoc Networks. *MILCOM 2008-2008 IEEE Military Communications Conference*, 1–7.

Dely, P., Kassler, A., & Bayer, N. (2011). Openflow for wireless mesh networks. *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, 1–6.

Detti, A., Pisa, C., Salsano, S., & Blefari-Melazzi, N. (2013). Wireless mesh software defined networks (wmSDN). *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 89–95.

Dusia, A. (2019). *Software-defined architecture and routing solutions for mobile ad hoc networks*. University of Delaware.

Haas, Z. J. (1997). A new routing protocol for the reconfigurable wireless networks.

Proceedings of ICUPC 97-6th International Conference on Universal Personal Communications, *2*, 562–566.

Hakiri, A., Gokhale, A., & Patil, P. (2017). *A software defined wireless networking for efficient communication in smart cities*. Technical report.

Hans, C. Y., Quer, G., & Rao, R. R. (2017). Wireless SDN mobile ad hoc network: From theory to practice. *2017 IEEE International Conference on Communications (ICC)*, 1–7.

He, Z., Cao, J., & Liu, X. (2016). SDVN: Enabling rapid network innovation for heterogeneous vehicular communication. *IEEE Network*, *30*(4), 10–15.

Hoebeke, J., Moerman, I., Dhoedt, B., & Demeester, P. (2004). An overview of mobile ad hoc networks: applications and challenges. *Journal-Communications Network*, *3*(3), 60–66.

Hong, X., Xu, K., & Gerla, M. (2002). Scalable routing protocols for mobile ad hoc networks. *IEEE Network*, *16*(4), 11–21.

ITU-T. (2012). Overview of the Internet of things. *Series Y: Global Information Infrastructure, Internet Protocol Aspects and next-Generation Networks - Frameworks and Functional Architecture Models*, 22. https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=Y.2060

Jayakumar, G., & Gopinath, G. (2007). Ad hoc mobile wireless networks routing protocols–a review. *Journal of Computer Science*, *3*(8), 574–582.

Jing, Q., Vasilakos, A. V, Wan, J., Lu, J., & Qiu, D. (2014). Security of the Internet of Things: perspectives and challenges. *Wireless Networks*, *20*(8), 2481–2501.

Johnson, D., Hu, Y., & Maltz, D. (2007). *The dynamic source routing protocol (DSR) for mobile ad hoc networks for IPv4*. RFC 4728.

Khatkar, A., & Singh, Y. (2012). Performance evaluation of hybrid routing protocols in mobile ad hoc networks. *2012 Second International Conference on Advanced Computing & Communication Technologies*, 542–545.

Ku, I., Lu, Y., Gerla, M., Gomes, R. L., Ongaro, F., & Cerqueira, E. (2014). Towards software-defined VANET: Architecture and services. *2014 13th Annual Mediterranean Ad Hoc*

Networking Workshop (MED-HOC-NET), 103–110.

Liu, C., & Kaiser, J. (2005). *Survey of mobile ad hoc network routing protocols*.

Mauve, M., Widmer, J., & Hartenstein, H. (2001). A survey on position-based routing in mobile ad hoc networks. *IEEE Network*, *15*(6), 30–39.

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

Mishra, V. K., Dusia, A., & Sethi, A. (2018). *Routing in software-defined mobile ad hoc networks (sd-manet)*. US Army Research Laboratory Aberdeen Proving Ground United States.

Montaño-Blacio, M., Briceño-Sarmiento, J., & Pesántez-Bravo, F. (2020). 5G Network Security for IoT Implementation: A Systematic Literature Review. *International Conference on Innovation and Research*, 28–40.

Montaño, M., Torres, R., Ludeña, P., & Sandoval, F. (2021). IoT Management Analysis Using SDN: Survey. *Communications in Computer and Information Science*, *1388 CCIS*, 574–589. https://doi.org/10.1007/978-3-030-71503-8_45

Ngu, A. H., Gutierrez, M., Metsis, V., Nepal, S., & Sheng, Q. Z. (2016). IoT middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, *4*(1), 1–20.

ns3. (n.d.). *About | ns-3*. Retrieved June 29, 2021, from https://www.nsnam.org/about/

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

ONF, F. O. N. (n.d.). *Software-Defined Networking (SDN) Definition - Open Networking Foundation*. Retrieved June 28, 2021, from https://opennetworking.org/sdn-definition/

Pappalardo, I., Quer, G., Rao, B. D., & Zorzi, M. (2016). Caching strategies in heterogeneous networks with D2D, small BS and macro BS communications. *2016 IEEE International Conference on Communications (ICC)*, 1–6.

Santamaria, C. J. G. (2017). *Management of a heterogeneous distributed architecture with the SDN*. Universite de Reims Champagne Ardenne.

Syarif, A., & Sari, R. F. (2011). Performance analysis of AODV-UI routing protocol with energy consumption improvement under mobility models in hybrid ad hoc network. *International Journal on Computer Science and Engineering*, *3*(7), 2904–2918.

Torres, R. (2011). Contribución a los modelos de gestión de las redes móviles Ad Hoc. *Universidad Politécnica de Madrid. Madrid*.

Truong, N. B., Lee, G. M., & Ghamri-Doudane, Y. (2015). Software defined networking-based vehicular adhoc network with fog computing. *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 1202–1207.

Wireshark. (n.d.). *Wireshark User's Guide*. Retrieved August 1, 2021, from https://www.wireshark.org/docs/wsug_html_chunked/

Wu, D., Arkhipov, D. I., Asmare, E., Qin, Z., & McCann, J. A. (2015). UbiFlow: Mobility management in urban-scale software defined IoT. *Proceedings - IEEE INFOCOM*, *26*, 208–216. https://doi.org/10.1109/INFOCOM.2015.7218384

Xia, W., Wen, Y., Foh, C. H., Niyato, D., & Xie, H. (2014). A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, *17*(1), 27–51.

Yick, J., Mukherjee, B., & Ghosal, D. (2008). Wireless sensor network survey. *Computer Networks*, *52*(12), 2292–2330.

Zhu, M., Cao, J., Pang, D., He, Z., & Xu, M. (2015). SDN-based routing for efficient message propagation in VANET. *International Conference on Wireless Algorithms, Systems, and Applications*, 788–797.

Zukerman, M., Neame, T. D., & Addie, R. G. (2003). Internet traffic modeling and future technology implications. *IEEE INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, *1*, 587–596 vol.1. https://doi.org/10.1109/INFCOM.2003.1208709

Apéndice

Apéndice 1: Configuración NS-3 para instalación OpenFlow

Paso1. Tiene los siguientes paquetes instalados en su sistema

$ sudo apt-get install build-essential gcc g ++ python git mercurial descomprimir cmake

$ sudo apt-get install pkg-config autoconf libtool libboost-dev

Paso 2. Descargue un código ns-3 estable reciente (usando el repositorio mercurial para ns-3.32)

$ wget https://www.nsnam.org/releases/ns-allinone-3.33.tar.bz2

Paso 3. Descomprimir archivo manual o en terminal

$ tar archivo.tar.bz2

Paso 4. Desde la carpeta ns-3-dev en la terminal (para c ++)

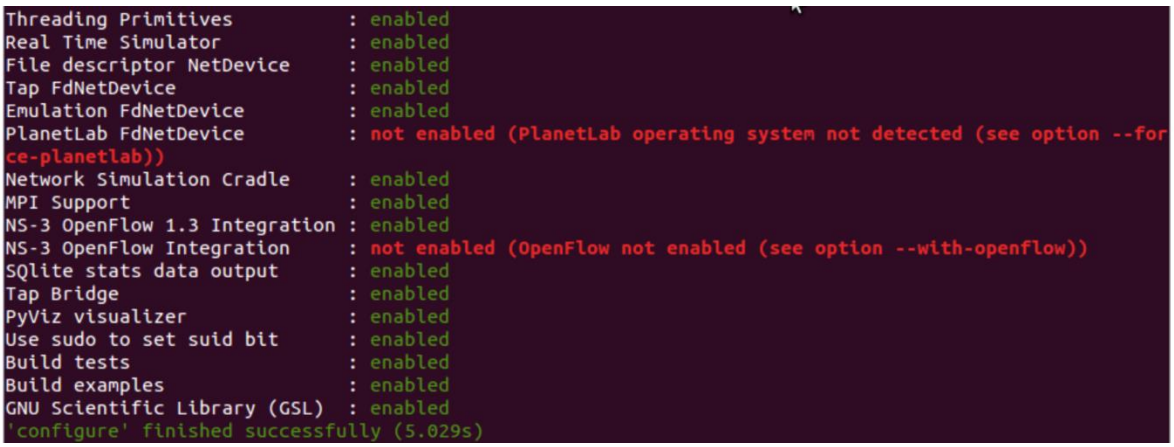
$ ./waf configure --enable-tests --enable-examples

$ ./waf build

Paso 5. Resultado OpenFlow integrado a NS-3.

Figura 33

Integración OpenFlow a NS-3



Apéndice 2: Script OLSR

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/olsr-routing-protocol.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/flow-monitor.h"
#include "ns3/olsr-helper.h"
#include "ns3/yans-wifi-helper.h"

#include "ns3/log.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("OlsrHna");

void ReceivePacket (Ptr<Socket> socket)
{
  NS_LOG_UNCOND ("Received one packet!");
}

static void GenerateTraffic (Ptr<Socket> socket, uint32_t pktSize,
                             uint32_t pktCount, Time pktInterval )
{
  if (pktCount > 0)
    {
      socket->Send (Create<Packet> (pktSize));
      Simulator::Schedule (pktInterval, &GenerateTraffic,
                           socket, pktSize,pktCount - 1, pktInterval);
    }
  else
    {
      socket->Close ();
    }
}

int main (int argc, char *argv[])
{
  std::string phyMode ("DsssRate1Mbps");
  double rss = -80;
```

```
  uint32_t packetSize = 1000;
  uint32_t numPackets = 1;
  double interval = 1.0;
  bool verbose = false;
  bool assocMethod1 = false;
  bool assocMethod2 = false;

  CommandLine cmd (__FILE__);

  cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
  cmd.AddValue ("rss", "received signal strength", rss);
  cmd.AddValue ("packetSize", "size of application packet sent", packetSize);
  cmd.AddValue ("numPackets", "number of packets generated", numPackets);
  cmd.AddValue ("interval", "interval (seconds) between packets", interval);
  cmd.AddValue ("verbose", "turn on all WifiNetDevice log components", verbos
e);
  cmd.AddValue ("assocMethod1", "Use SetRoutingTableAssociation () method", a
ssocMethod1);
  cmd.AddValue ("assocMethod2", "Use AddHostNetworkAssociation () method", as
socMethod2);

  cmd.Parse (argc, argv);

  Time interPacketInterval = Seconds (interval);

  Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold"
, StringValue ("2200"));

  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", Strin
gValue ("2200"));

  Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
                      StringValue (phyMode));

  NodeContainer olsrNodes;
  olsrNodes.Create (30);

  NodeContainer csmaNodes;
  csmaNodes.Create (4);
  WifiHelper wifi;
  if (verbose)
    {
      wifi.EnableLogComponents ();
  wifi.SetStandard (WIFI_STANDARD_80211b);

  YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
  wifiPhy.Set ("RxGain", DoubleValue (0) );
  wifiPhy.SetPcapDataLinkType (WifiPhyHelper::DLT_IEEE802_11_RADIO);
```

```
  YansWifiChannelHelper wifiChannel;
  wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel")
;

  wifiChannel.AddPropagationLoss ("ns3::FixedRssLossModel","Rss",DoubleValue
(rss));
  wifiPhy.SetChannel (wifiChannel.Create ());

  WifiMacHelper wifiMac;
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode",StringValue (phyMode),

  wifiMac.SetType ("ns3::AdhocWifiMac");
  NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, olsrNodes);

  CsmaHelper csma;

  csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate (5000000)));
  csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
  NetDeviceContainer csmaDevices = csma.Install (NodeContainer (csmaNodes.Get
 (0), olsrNodes.Get (1)));

  csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate (50000000000)
));
  csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (100)));
  NetDeviceContainer csmaDevices1 = csma.Install (olsrNodes.Get (1));

  std::string traceFile = "src/mobility/examples/escenario.ns_movements";
  MobilityHelper mobility;
  Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocat
or> ();
  positionAlloc->Add (Vector (0.0, 0.0, 0.0));
  positionAlloc->Add (Vector (5.0, 0.0, 0.0));
  mobility.SetPositionAllocator (positionAlloc);
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (olsrNodes);

  cmd.AddValue ("traceFile", "Ns2 movement trace file", traceFile);

  Ns2MobilityHelper ns2 = Ns2MobilityHelper (traceFile);
  ns2.Install();

  OlsrHelper olsr;
  olsr.ExcludeInterface (olsrNodes.Get (1), 2);

  Ipv4StaticRoutingHelper staticRouting;
```

```
  Ipv4ListRoutingHelper list;
  list.Add (staticRouting, 0);
  list.Add (olsr, 30);

  InternetStackHelper internet_olsr;
  internet_olsr.SetRoutingHelper (list);
  internet_olsr.Install (olsrNodes);

  InternetStackHelper internet_csma;
  internet_csma.Install (csmaNodes);

  Ipv4AddressHelper ipv4;
  NS_LOG_INFO ("Assign IP Addresses.");
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  ipv4.Assign (devices);

  ipv4.SetBase ("172.16.1.0", "255.255.255.0");
  ipv4.Assign (csmaDevices);
  ipv4.Assign (csmaDevices1);

  uint16_t port = 9;

  OnOffHelper onoff ("ns3::TcpSocketFactory",
                     Address (InetSocketAddress (Ipv4Address ("10.1.1.2"), po
rt)));
  onoff.SetConstantRate (DataRate ("500Mb/s"));

  ApplicationContainer app = onoff.Install (olsrNodes.Get (6));
  app.Start (Seconds (1.0));
  app.Stop (Seconds (100.0));
  PacketSinkHelper sink ("ns3::TcpSocketFactory",
                         Address (InetSocketAddress (Ipv4Address::GetAny (),
port)));
  app = sink.Install (olsrNodes.Get (1));
  app.Start (Seconds (0.0));
  onoff.SetAttribute ("Remote",
                      AddressValue (InetSocketAddress (Ipv4Address ("10.1.1.1
"), port)));
  app = onoff.Install (olsrNodes.Get (3));
  app.Start (Seconds (1.1));
  app.Stop (Seconds (100.0));

  app = sink.Install (olsrNodes.Get (0));
  app.Start (Seconds (0.0));
  Ptr<Ipv4> stack = olsrNodes.Get (1)->GetObject<Ipv4> ();
  Ptr<Ipv4RoutingProtocol> rp_Gw = (stack->GetRoutingProtocol ());
  Ptr<Ipv4ListRouting> lrp_Gw = DynamicCast<Ipv4ListRouting> (rp_Gw);
```

```
  Ptr<olsr::RoutingProtocol> olsrrp_Gw;

  for (uint32_t i = 0; i < lrp_Gw->GetNRoutingProtocols ();  i++)
    {
      int16_t priority;
      Ptr<Ipv4RoutingProtocol> temp = lrp_Gw-
>GetRoutingProtocol (i, priority);
      if (DynamicCast<olsr::RoutingProtocol> (temp))
        {
          olsrrp_Gw = DynamicCast<olsr::RoutingProtocol> (temp);
        }
    }

  if (assocMethod1)
    {
      Ptr<Ipv4StaticRouting> hnaEntries = Create<Ipv4StaticRouting> ();
      hnaEntries-
>AddNetworkRouteTo (Ipv4Address ("172.16.1.0"), Ipv4Mask ("255.255.255.0"), u
int32_t (6), uint32_t (1));
      olsrrp_Gw->SetRoutingTableAssociation (hnaEntries);
    }

  if (assocMethod2)
    {
      olsrrp_Gw-
>AddHostNetworkAssociation (Ipv4Address ("172.16.1.0"), Ipv4Mask ("255.255.25
5.0"));
    }

  wifiPhy.EnablePcap ("resultados/olsr/olsr-hna", devices);
  csma.EnablePcap ("resultados/olsr/olsr-hna", csmaDevices, false);

  Ptr<FlowMonitor> flowMonitor;
  FlowMonitorHelper flowMonitorHelper;
  flowMonitor = flowMonitorHelper.InstallAll();

  Simulator::Stop (Seconds (100.0));
  Simulator::Run ();
  Simulator::Destroy ();

  flowMonitor->SetAttribute("DelayBinWidth", DoubleValue(0.01));
  flowMonitor->SetAttribute("JitterBinWidth", DoubleValue(0.01));
  flowMonitor->SetAttribute("PacketSizeBinWidth", DoubleValue(1));
  flowMonitor->CheckForLostPackets();
  flowMonitor-
>SerializeToXmlFile("resultados/olsr/flowMonNodes.xml", true, true);
  return 0;
}
```

Apéndice 3: Script OpenFlow

```
#include <iostream>
#include <fstream>

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/openflow-module.h"
#include "ns3/mobility-module.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/flow-monitor.h"
#include "ns3/log.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("OpenFlowCsmaSwitchExample");

bool verbose = false;
bool use_drop = false;
ns3::Time timeout = ns3::Seconds (100);
std::string phyMode ("DsssRate1Mbps");

bool
SetVerbose (std::string value)
{
  verbose = true;
  return true;
}

bool
SetDrop (std::string value)
{
  use_drop = true;
  return true;
}

bool
SetTimeout (std::string value)
{
  try {
      timeout = ns3::Seconds (atof (value.c_str ()));
      return true;
    }
  catch (...) { return false; }
  return false;
```

```
}

int
main (int argc, char *argv[])
{
  #ifdef NS3_OPENFLOW

  CommandLine cmd (__FILE__);
  cmd.AddValue ("v", "Verbose (turns on logging).", MakeCallback (&SetVerbose
));
  cmd.AddValue ("verbose", "Verbose (turns on logging).", MakeCallback (&SetV
erbose));
  cmd.AddValue ("d", "Use Drop Controller (Learning if not specified).", Make
Callback (&SetDrop));
  cmd.AddValue ("drop", "Use Drop Controller (Learning if not specified).", M
akeCallback (&SetDrop));
  cmd.AddValue ("t", "Learning Controller Timeout (has no effect if drop cont
roller is specified).", MakeCallback ( &SetTimeout));
  cmd.AddValue ("timeout", "Learning Controller Timeout (has no effect if dro
p controller is specified).", MakeCallback ( &SetTimeout));

  cmd.Parse (argc, argv);

  if (verbose)
    {
      LogComponentEnable ("OpenFlowCsmaSwitchExample", LOG_LEVEL_INFO);
      LogComponentEnable ("OpenFlowInterface", LOG_LEVEL_INFO);
      LogComponentEnable ("OpenFlowSwitchNetDevice", LOG_LEVEL_INFO);
    }


  NS_LOG_INFO ("Create nodes.");
  NodeContainer terminals;
  terminals.Create (30);

  NodeContainer csmaSwitch;
  csmaSwitch.Create (5);

  WifiHelper wifi;
  wifi.SetStandard (WIFI_STANDARD_80211b);
  YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
  wifiPhy.Set ("RxGain", DoubleValue (0) );
  wifiPhy.SetPcapDataLinkType (WifiPhyHelper::DLT_IEEE802_11_RADIO);
  YansWifiChannelHelper wifiChannel;
  wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel")
;
  wifiChannel.AddPropagationLoss ("ns3::FixedRssLossModel","Rss",DoubleValue
(-80));
```

```
  wifiPhy.SetChannel (wifiChannel.Create ());
  WifiMacHelper wifiMac;
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode",StringValue (phyMode),
                                "ControlMode",StringValue (phyMode));
  wifiMac.SetType ("ns3::AdhocWifiMac");
  NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, terminals);

  NS_LOG_INFO ("Build Topology");
  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", DataRateValue (5000000));
  csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));

  NetDeviceContainer terminalDevices;
  NetDeviceContainer switchDevices;

  std::string traceFile = "src/mobility/examples/escenario.ns_movements";

  for (int i = 0; i < 5; i++)
    {
      NetDeviceContainer link = csma.Install (NodeContainer (terminals.Get (i
), csmaSwitch));
      terminalDevices.Add (link.Get (0));
      switchDevices.Add (link.Get (1));
    }

  Ptr<Node> switchNode = csmaSwitch.Get (0);
  OpenFlowSwitchHelper swtch;

  if (use_drop)
    {
      Ptr<ns3::ofi::DropController> controller = CreateObject<ns3::ofi::DropC
ontroller> ();
      swtch.Install (switchNode, switchDevices, controller);
    }
  else
    {
      Ptr<ns3::ofi::LearningController> controller = CreateObject<ns3::ofi::L
earningController> ();
      if (!timeout.IsZero ()) controller-
>SetAttribute ("ExpirationTime", TimeValue (timeout));
      swtch.Install (switchNode, switchDevices, controller);
    }

  InternetStackHelper internet;
  internet.Install (terminals);

  cmd.AddValue ("traceFile", "Ns2 movement trace file", traceFile);
```

```
  Ns2MobilityHelper ns2 = Ns2MobilityHelper (traceFile);
  ns2.Install();

  NS_LOG_INFO ("Assign IP Addresses.");
  Ipv4AddressHelper ipv4;
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  ipv4.Assign (terminalDevices);

  NS_LOG_INFO ("Create Applications.");
  uint16_t port = 9;

  PPBPHelper ppbp = PPBPHelper ("ns3::TcpSocketFactory",
                    Address (InetSocketAddress (Ipv4Address ("10.1.1.2"), po
rt)));

  ApplicationContainer app = ppbp.Install (terminals.Get (6));

  app.Start (Seconds (1.0));
  app.Stop (Seconds (100.0));


  PacketSinkHelper sink ("ns3::TcpSocketFactory",
                         Address (InetSocketAddress (Ipv4Address::GetAny (),
port)));
  app = sink.Install (terminals.Get (1));
  app.Start (Seconds (0.0));


  ppbp.SetAttribute ("Remote",
                     AddressValue (InetSocketAddress (Ipv4Address ("10.1.1.1
"), port)));
  app = ppbp.Install (terminals.Get (3));
  app.Start (Seconds (1.1));
  app.Stop (Seconds (100.0));

  app = sink.Install (terminals.Get (0));
  app.Start (Seconds (0.0));

  NS_LOG_INFO ("Configure Tracing.");


  AsciiTraceHelper ascii;
  csma.EnableAsciiAll (ascii.CreateFileStream ("resultados/nw/openflow-
switch.tr"));

  csma.EnablePcapAll ("resultados/nw/openflow-ppbp-switch", false);
```

```
  NS_LOG_INFO ("Run Simulation.");

  Ptr<FlowMonitor> flowMonitor;
  FlowMonitorHelper flowMonitorHelper;
  flowMonitor = flowMonitorHelper.InstallAll();

  Simulator::Stop (Seconds(100));
  Simulator::Run ();
  Simulator::Destroy ();


  flowMonitor->SetAttribute("DelayBinWidth", DoubleValue(0.01));
  flowMonitor->SetAttribute("JitterBinWidth", DoubleValue(0.01));
  flowMonitor->SetAttribute("PacketSizeBinWidth", DoubleValue(1));
  flowMonitor->CheckForLostPackets();
  flowMonitor->SerializeToXmlFile("resultados/nw/flow-ppbp-
MonNodes.xml", true, true);

  NS_LOG_INFO ("Done.");
  #else
  NS_LOG_INFO ("NS-3 OpenFlow is not enabled. Cannot run simulation.");
  #endif
}
```

Apéndice 4: Script PPBP

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/olsr-routing-protocol.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/flow-monitor.h"
#include "ns3/olsr-helper.h"
#include "ns3/yans-wifi-helper.h"

#include "ns3/log.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("PPBPExample");

void ReceivePacket (Ptr<Socket> socket)
{
  NS_LOG_UNCOND ("Received one packet!");
}

static void GenerateTraffic (Ptr<Socket> socket, uint32_t pktSize,
                             uint32_t pktCount, Time pktInterval )
{
  if (pktCount > 0)
    {
      socket->Send (Create<Packet> (pktSize));
      Simulator::Schedule (pktInterval, &GenerateTraffic,
                           socket, pktSize,pktCount - 1, pktInterval);
    }
  else
    {
      socket->Close ();
    }
}


int main (int argc, char *argv[])
{
  std::string phyMode ("DsssRate1Mbps");
```

```
  double rss = -80;  // -dBm
  uint32_t packetSize = 1000; // bytes
  uint32_t numPackets = 1;
  double interval = 1.0; // seconds
  bool verbose = false;
  bool assocMethod1 = false;
  bool assocMethod2 = false;

  CommandLine cmd (__FILE__);

  cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
  cmd.AddValue ("rss", "received signal strength", rss);
  cmd.AddValue ("packetSize", "size of application packet sent", packetSize);
  cmd.AddValue ("numPackets", "number of packets generated", numPackets);
  cmd.AddValue ("interval", "interval (seconds) between packets", interval);
  cmd.AddValue ("verbose", "turn on all WifiNetDevice log components", verbos
e);
  cmd.AddValue ("assocMethod1", "Use SetRoutingTableAssociation () method", a
ssocMethod1);
  cmd.AddValue ("assocMethod2", "Use AddHostNetworkAssociation () method", as
socMethod2);

  cmd.Parse (argc, argv);

  Time interPacketInterval = Seconds (interval);


  Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold"
, StringValue ("2200"));

  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", Strin
gValue ("2200"));

  Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
                      StringValue (phyMode));

  NodeContainer olsrNodes;
  olsrNodes.Create (20);

  NodeContainer csmaNodes;
  csmaNodes.Create (4);


  WifiHelper wifi;
  if (verbose)
    {
      wifi.EnableLogComponents ();
    }
```

```
  wifi.SetStandard (WIFI_STANDARD_80211b);

  YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();

  wifiPhy.Set ("RxGain", DoubleValue (0) );

  wifiPhy.SetPcapDataLinkType (WifiPhyHelper::DLT_IEEE802_11_RADIO);

  YansWifiChannelHelper wifiChannel;
  wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel")
;

  wifiChannel.AddPropagationLoss ("ns3::FixedRssLossModel","Rss",DoubleValue
(rss));
  wifiPhy.SetChannel (wifiChannel.Create ());

  WifiMacHelper wifiMac;
  wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode",StringValue (phyMode),
                                "ControlMode",StringValue (phyMode));

  wifiMac.SetType ("ns3::AdhocWifiMac");
  NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, olsrNodes);

  CsmaHelper csma;

  csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate ("100Mbps")))
;
  csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
  NetDeviceContainer csmaDevices = csma.Install (NodeContainer (csmaNodes.Get
 (0), olsrNodes.Get (1)));

  csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate ("1000Mbps"))
);
  csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (100)));
  NetDeviceContainer csmaDevices1 = csma.Install (olsrNodes.Get (1));

  std::string traceFile = "src/mobility/examples/escenario.ns_movements";


  MobilityHelper mobility;
  Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocat
or> ();
  positionAlloc->Add (Vector (0.0, 0.0, 0.0));
  positionAlloc->Add (Vector (5.0, 0.0, 0.0));
  mobility.SetPositionAllocator (positionAlloc);
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (olsrNodes);
```

```
  cmd.AddValue ("traceFile", "Ns2 movement trace file", traceFile);

  Ns2MobilityHelper ns2 = Ns2MobilityHelper (traceFile);
  ns2.Install();

  OlsrHelper olsr;


  olsr.ExcludeInterface (olsrNodes.Get (1), 2);

  Ipv4StaticRoutingHelper staticRouting;

  Ipv4ListRoutingHelper list;
  list.Add (staticRouting, 0);
  list.Add (olsr, 20);

  InternetStackHelper internet_olsr;
  internet_olsr.SetRoutingHelper (list);
  internet_olsr.Install (olsrNodes);

  InternetStackHelper internet_csma;
  internet_csma.Install (csmaNodes);

  Ipv4AddressHelper ipv4;
  NS_LOG_INFO ("Assign IP Addresses.");
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  ipv4.Assign (devices);

  ipv4.SetBase ("172.16.1.0", "255.255.255.0");
  ipv4.Assign (csmaDevices);
  ipv4.Assign (csmaDevices1);

  uint16_t port = 9;

  PPBPHelper ppbp = PPBPHelper ("ns3::TcpSocketFactory",
                    Address (InetSocketAddress (Ipv4Address ("10.1.1.2"), po
rt)));

  ApplicationContainer app = ppbp.Install (olsrNodes.Get (6));

  app.Start (Seconds (1.0));
  app.Stop (Seconds (100.0));


  PacketSinkHelper sink ("ns3::TcpSocketFactory",
                         Address (InetSocketAddress (Ipv4Address::GetAny (),
port)));
  app = sink.Install (olsrNodes.Get (1));
```

```
  app.Start (Seconds (0.0));

  ppbp.SetAttribute ("Remote",
                     AddressValue (InetSocketAddress (Ipv4Address ("10.1.1.1
"), port)));
  app = ppbp.Install (olsrNodes.Get (3));
  app.Start (Seconds (1.1));
  app.Stop (Seconds (100.0));

  app = sink.Install (olsrNodes.Get (0));
  app.Start (Seconds (0.0));

  Ptr<Ipv4> stack = olsrNodes.Get (1)->GetObject<Ipv4> ();
  Ptr<Ipv4RoutingProtocol> rp_Gw = (stack->GetRoutingProtocol ());
  Ptr<Ipv4ListRouting> lrp_Gw = DynamicCast<Ipv4ListRouting> (rp_Gw);

  Ptr<olsr::RoutingProtocol> olsrrp_Gw;

  for (uint32_t i = 0; i < lrp_Gw->GetNRoutingProtocols ();  i++)
    {
      int16_t priority;
      Ptr<Ipv4RoutingProtocol> temp = lrp_Gw-
>GetRoutingProtocol (i, priority);
      if (DynamicCast<olsr::RoutingProtocol> (temp))
        {
          olsrrp_Gw = DynamicCast<olsr::RoutingProtocol> (temp);
        }
    }

  if (assocMethod1)
    {

      Ptr<Ipv4StaticRouting> hnaEntries = Create<Ipv4StaticRouting> ();

      hnaEntries-
>AddNetworkRouteTo (Ipv4Address ("172.16.1.0"), Ipv4Mask ("255.255.255.0"), u
int32_t (6), uint32_t (1));
      olsrrp_Gw->SetRoutingTableAssociation (hnaEntries);
    }

  if (assocMethod2)
    {

      olsrrp_Gw-
>AddHostNetworkAssociation (Ipv4Address ("172.16.1.0"), Ipv4Mask ("255.255.25
5.0"));
    }
  wifiPhy.EnablePcap ("resultados/olsr/ftp-olsr-hna", devices);
```

```
  csma.EnablePcap ("resultados/olsr/ftp-olsr-hna", csmaDevices, false);

  Ptr<FlowMonitor> flowMonitor;
  FlowMonitorHelper flowMonitorHelper;
  flowMonitor = flowMonitorHelper.InstallAll();
  Simulator::Stop (Seconds (100.0));
  Simulator::Run ();
  Simulator::Destroy ();
  flowMonitor->SetAttribute("DelayBinWidth", DoubleValue(0.01));
  flowMonitor->SetAttribute("JitterBinWidth", DoubleValue(0.01));
  flowMonitor->SetAttribute("PacketSizeBinWidth", DoubleValue(1));
  flowMonitor->CheckForLostPackets();
  flowMonitor->SerializeToXmlFile("resultados/olsr/ftp-
flowMonNodes.xml", true, true);
  return 0;
}
```