Análisis del Desempeño de una SDN IoT al Implementar un Algoritmo de Control de Tráfico basado en *Machine Learning*

Laura Camila Artunduaga Sánchez
Jhon Hamilton Charo Orozco

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telecomunicaciones
Grupo I+D Nuevas Tecnologías en Telecomunicaciones (GNTT)
Popayán, Cauca.
2022

Análisis del Desempeño de una SDN IoT al Implementar un Algoritmo de Control de Tráfico basado en Machine Learning

Laura Camila Artunduaga Sánchez
Jhon Hamilton Charo Orozco

Trabajo de grado para optar al título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Director:
Ing. Catalina Muñoz Collazos MsC

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telecomunicaciones
Grupo I+D Nuevas Tecnologías en Telecomunicaciones (GNTT)
Popayán, Cauca.
2022

Dedico este logro a mi más grande tesoro,
mis padres Héctor Mario Artunduaga y María Yanith Sánchez,
y a mi Hermanito Héctor Mario Artunduaga Sánchez porque gracias
a su apoyo incondicional y ejemplo de perseverancia me impulsaron
a luchar y no rendirme para alcanzar este objetivo en mi vida profesional.

Laura Camila Artunduaga Sánchez

Le dedico este proyecto a toda mi familia
que ha contribuido a la consecución de este logro,
especialmente a mis padres que han creído en mí siempre,
quienes me criaron con mucho amor, humildad y me han dado
ejemplo de superación, me ayudaron con mucho cariño a no desfallecer
ante los obstáculos y son él pilar de mi formación como persona.

Jhon Hamilton Charo Orozco

Agradecimientos

Después de Dios, los autores expresan su agradecimiento a la Magister Catalina Muñoz Collazos quien tomo la dirección de este proyecto y compartió sus conocimientos y empuje para llevarlo a término.

A la Universidad del Cauca, en especial al programa de ingeniería Electrónica y Telecomunicaciones y al departamento de Telecomunicaciones por los años y recursos dedicados a la formación de excelentes profesionales y seres humanos.

A cada uno de sus familiares, compañeros y amigos por el apoyo y acompañamiento durante todo este proceso, pues sin ello no hubiese sido posible culminar con éxito esta etapa académica.

CONTENIDO

Lista de figuras

Lista de tablas

Lista de Anexos

Listado de acrónimos

APIs	*Application Programming Interface,* Interfaz de Programación de Aplicaciones.
BGP	*Border Gateway Protocol,* Protocolo de Puerta de Enlace de Frontera.
BSD	*Berkeley Software Distribution,* Distribución de software Berkeley.
CLI	*Command-Line Interface,* Interfaz de Línea de Comandos.
EPL	*Eclipse Public License,* Licencia Pública Eclipse.
FTTx	*Fiber to the x,* Fibra hasta x.
GPL	*General Public License,* Licencia Pública General.
HTC	*Human Type Communications,* Comunicación de Tipo Humano.
IoT	*Internet of Things,* Internet de las Cosas.
IP	*Internet Protocol,* Protocolo de Internet.
IPv4	*Internet Protocol Version 4,* Protocolo de Internet versión 4.
IPv6	*Internet Protocol Version 6,* Protocolo de Internet versión 6.
M2M	*Machine to Machine,* Máquina a Máquina.
ML	*Machine Learning,* Aprendizaje de Máquinas.
MTC	*Machine Type Communications,* Comunicaciones Tipo Máquina.
NAT	*Network Adress Translation ,* Traducción de Direcciones de Red.

NFC	*Near-Field Communication,* Comunicación de Campo Cercano.
QoS	*Quality of Service,* Calidad de Servicio.
RFID	*Radio Frecuency Identification,* Identificación por Radiofrecuencia.
SD-IoT	*Software Defined Based Internet of Things*, Internet de las cosas definido Por Software.
SDN	*Software Define Network,* Redes Definidas por Software.
TC	*Traffic Control,* Control de Tráfico.
TCP	*Transmission Control Protocol,* Protocolo de Control de Transmisión.
UDP	*User Datagram Protocol,* Protocolo de Datagramas de Usuario.
Wi-Fi	*Wireless Fidelity,* Fidelidad inalámbrica.
Wimax	*Worldwide Interoperability for Microwave Access,* Interoperabilidad Mundial para Acceso por Microondas
xDSL	*X Digital Suscriber Line,* Línea de Suscriptor Digital

INTRODUCCIÓN

Inicialmente cuando se creó Internet su objetivo era proporcionar un camino para que la información viajara de un lugar a otro; sin embargo, la trivialidad de esta red no llenaba las expectativas de algunos usuarios. En los últimos años se ha evidenciado el alto crecimiento de las comunicaciones, resultado de ello, hoy por hoy se tienen acceso a millones de dispositivos conectados que están haciendo uso de miles de aplicaciones, desde la más simple como el correo electrónico hasta grandes aplicaciones distribuidas, que requieren en ocasiones de la capacidad de monitorear y administrar la red. El Internet de las cosas (IoT, *Internet of Things*) es un paradigma revolucionario que llega para transformar las comunicaciones convencionales, permitiendo por medio de servidores la interconexión de millones de dispositivos de bajo costo con el mundo físico en cualquier momento y en cualquier lugar, dando paso a nuevos conceptos como ciudades, hogares, vehículos y salud inteligentes entre muchos otros entornos.

Cuando se habla de entornos inteligentes, también se debe tener en cuenta los obstáculos e inconvenientes que trae consigo la tecnología IoT. Si bien, la función de los dispositivos utilizados es detectar y recopilar datos para el correcto funcionamiento de IoT, esto también puede generar que las cargas de tráfico integradas a la red se vuelvan extremadamente densas, generando problemas de interoperabilidad, congestión, perdida de paquetes, entre otros, pues cada aplicación tiene propiedades diferentes y genera su propio tráfico con características únicas.

Una vez claras las dificultades del Internet de las Cosas, es necesario realizar la búsqueda de una tecnología que al combinarla con IoT permita dar solución a todas las falencias que esta presenta y facilite la administración de la red; las Redes definidas por Software (SDN, *Software Defined Networking*) es la tecnología que permitirá hacer de IoT un sistema más escalable y dinámico, ya que al realizar la separación del plano de datos del plano de control se pasará a tener una única entidad de control que facilitará el manejo de los dispositivos incorporados a la red y ayudará a mejorar la

calidad del servicio, de esta integración nace el concepto del Internet de las Cosas Definido por Software (SD-IoT, *Software Define-Internet of Things*).

Una forma de evitar la sobrecarga o congestión en las redes IoT, es realizar un Control de Tráfico (TC, *Traffic Control*), ya que de esta manera se puede tener un control del rendimiento de estas, y se puede contribuir al mejoramiento de la QoS para los usuarios. Una de las técnicas utilizadas para hacer TC es agrupar todos los datos similares y relacionarlos en una misma categoría para brindarles un tratamiento diferenciado y de esta forma poder dar mayor prioridad a los datos que son más importantes para el usuario, o para garantizar que no circulen paquetes maliciosos que afecten la seguridad de la información.

Para efectos del desarrollo de este trabajo, se abordará el problema de la densidad de tráfico en las redes IoT mediante el desarrollo de un mecanismo que permita analizar el tráfico que circula por la red y luego por medio de un algoritmo reforzado seleccionar la ruta menos congestionada para enviarlo al destino; centrando todos los esfuerzos en los métodos de aprendizaje automático, pues al tener una intervención humana mínima se puede garantizar un resultado más confiable. Inicialmente los dispositivos emulados envíen ráfagas de datos, una vez inicia el controlador ONOS procede a analizar la red para aplicar el algoritmo por refuerzo para la selección de ruta y posterior a ello instala la regla de flujo indicándole a los nodos por cual camino debe enviar los paquetes del dispositivo seleccionado; esto con el fin agilizar el envío de información prioritaria. Finalmente se hará la comparación de los resultados obtenidos con los arrojados por la red sin implementar dicho mecanismo.

El presente documento de trabajo de grado describe lo realizado a través de los siguientes capítulos:

El capítulo 1, presenta los conceptos generales y contextualiza las redes SD-IoT.

El capítulo 2, explica las metodologías implementadas para el desarrollo del proyecto.

El capítulo 3, muestra la instalación y explica brevemente el manejo de las herramientas utilizadas para el desarrollo del trabajo.

El capítulo 4, describe la simulación tanto de la red como del algoritmo de aprendizaje automático y la aplicación con la que trabaja el controlador ONOS.

EL capítulo 5, presenta los resultados obtenidos y el análisis de estos.

El capítulo 6, expone las conclusiones y retroalimentación del desarrollo, y presenta los posibles trabajos futuros a partir del trabajo desarrollado.

1. CONSIDERACIONES GENERALES

Este capítulo presenta una descripción de los temas principales que se deben comprender para el desarrollo del presente trabajo de grado, como el Internet de las Cosas, Redes Definidas por Software, Tráfico en Redes y *Machine Learning*, también se indica las ventajas, aplicaciones e integración de cada uno de estos conceptos para dar cumplimiento a cada uno de los objetivos planteados. Finalmente se muestran las generalidades de las distintas herramientas de desarrollo necesarias para llevar a cabo las demostraciones de la investigación planteada.

1.1. Internet de las Cosas

El Internet de las Cosas (IoT, *Internet of Things)*, es considerado como una infraestructura global de información definida por la conexión de millones de dispositivos y sensores inteligentes conectados a través Internet; dicho de otro modo, es la conexión de cualquier objeto, en cualquier momento y en todo lugar, como se muestra en la Figura 1-1. La función principal de estos dispositivos y sensores que hacen parte de estas redes es recopilar y compartir datos para evaluar los diferentes entornos donde son utilizados, dentro de los cuales se encuentran empresas, ciudades e individuos. Esta tecnología ha sido posible en gran parte debido a la aparición de procesadores y redes inalámbricas asequibles; pues los objetos que antes eran inanimados, como las manijas de las puertas o las luces, ahora pueden equiparse con sensores inteligentes que pueden recopilar datos y transmitirlos a la red. Los investigadores estiman que en los próximos tres años, es decir, que para el 2025 habrá aproximadamente un total cuarenta punto dos billones de dispositivos conectados en todo el mundo, de los cuales treinta punto nueve billones son dispositivos IoT [1], [2].

El tener tantos dispositivos conectados y tantos datos circulando en la red de manera constante genera algunas falencias en esta tecnología; una de ellas es la seguridad, la cual es considerada unas de las mayores desventajas de los sistemas IoT; ya que se encuentran vulnerables a los ataques cibernéticos y si se presenta un mal diseño en la interconexión de los dispositivos la información del usuario se puede ver

afectada. Otro inconveniente que se presenta de forma frecuente en las redes de entornos inteligentes es el manejo de la información, pues al hacer uso de múltiples tecnologías para conectar millones de dispositivos, donde cada uno tiene su propia funcionalidad, la interconexión de estos se vuelve compleja debido al gran volumen de datos que manejan; por tanto, se hace necesario tener una infraestructura de red adecuada que permita tener una alta escalabilidad y a su vez una excelente administración de la información [1].

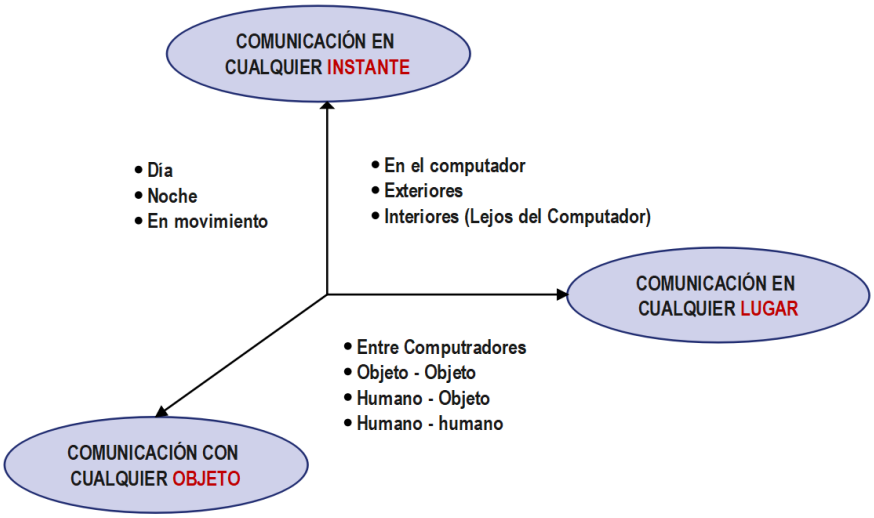

Figura 1-1 La nueva dimensión que introduce Internet a los Objetos [2]

1.1.1. Características

Desde la década de los 2000 la industria IoT le ha apostado a generar tecnología de costos asequibles, de consumo energético bajo y al alcance de cualquier persona. Con el tiempo se han establecido ciertas características de alto nivel propias de los sistemas IoT, las cuales son:

Interconectividad: Para que los dispositivos se interrelacionen con los usuarios o con otros sistemas ya sea a través de sensores, softwares, inteligencia artificial, sistemas computarizados entre otros [2], [3].

Servicios relacionados con objetos: Es capaz de proporcionar servicios en relación con los objetos dentro de los límites establecidos por estos mismo, es decir, para ofrecer los servicios se debe garantizar la protección de la privacidad [2], [3].

Heterogeneidad: Dado que los dispositivos para esta tecnología IoT se basan en diferente hardware y estos interactúan con múltiples dispositivos o plataformas de servicios a través de diversas redes [2], [3].

Cambios dinámicos: En IoT todo se mantiene en constante cambio, el estado, el contexto y el número de dispositivos conectados [2], [3].

Escala enorme: Con el tiempo, el número de dispositivos a gestionar es mucho mayor al número de dispositivos conectados a Internet, por lo que es esencial la administración de los datos generados y la interpretación que se le hace [2], [3].

Sensibilidad: Los dispositivos deben ser capaces de detectar los tipos de parámetros para los que estén configurados [2], [3].

Interacción: Para establecer la relación entre las personas, el mundo físico y los dispositivos, IoT debe tener interfaces y sistemas de comunicaciones óptimas [2], [3].

Redes Automáticas: IoT debe tener funciones de control automáticas, como la configuración, restablecimientos, optimización, y todos estos deben adaptarse a las diferentes aplicaciones en los múltiples contextos [2], [3].

Configuración automática de servicios: Es preciso poder configurar los servicios a partir de los datos obtenidos, y de esa manera poder personalizar las configuraciones de los dispositivos a las necesidades de cada usuario [2], [3].

Seguridad: Todo lo que esté conectado a una red de Internet ya sea cableada o inalámbrica siempre tendrá el riesgo de amenazas en la confidencialidad, autenticidad e integridad de sus datos; por tanto, en IoT cada día se trabaja e investiga más en cómo mejorar en este aspecto para que sea más seguro para los usuarios de esta tecnología [2], [3].

Para lograr la integración de todas las características detalladas anteriormente, el Internet de las Cosas tiene a su disposición múltiples herramientas que se muestran en la Tabla 1-1 y las cuales facilitan que esta tecnología no solo cuente con una interfaz

amigable con sus usuarios, sino también garanticen la interconexión entre dispositivos y personas [2], [3].

Tabla 1-1 Herramientas que complementan IoT [3]

• Comunicación inalámbrica • Etiquetado Geográfico • Interfaces M2M • Interfaces tangibles • Máquinas de visión • Microcontroladores • Protocolos de comunicación electrónica • Realidad aumentada	• Robótica • Sensores • Software • Tecnología de almacenamiento de energía • Tecnología de localización • Tecnología RFID • Tecnologías limpias

1.1.2. Arquitectura

Con el tiempo el concepto de IoT ha venido madurando, sin embargo, aún no se tiene una arquitectura estándar para estos sistemas, ya que cada proveedor adapta la arquitectura acorde con los servicios que presta y a las necesidades de sus usuarios. No obstante, cada propuesta hecha debe adaptarse a un conjunto de principios, los cuales establecen que la arquitectura debe ser: escalable, horizontal, flexible, segura, gestionable y con facilidad de controlar los dispositivos [1], [2].

La arquitectura generalmente se describe en términos de múltiples "capas", donde cada una de estas tiene sus propias funciones específicas y sus propios protocolos. Actualmente, se encuentran varias propuestas de modelos; los más mencionados son los de tres, cuatro, cinco y siete capas o niveles; se puede decir que todas las arquitecturas conservan una misma base que es el modelo de tres niveles; lo que difiere de los demás es que en algunos de ellos profundizan un poco más en los componentes de la tecnología; según las necesidades del proveedor se pueden agregar capacidades de procesamiento en la nube, o una capa exclusiva para la seguridad entre otros; sin embargo, a continuación se presentarán los detalles solo del modelo de tres niveles, no solo porque este es la base de todos los demás sino también porque permite o facilita la adaptación al incorporar SDN para posibles soluciones [1], [2].

Arquitectura de 3 niveles

Figura 1-2 Modelo de Referencia de 3 niveles [4]

El modelo de referencia de la Figura 1-2 es el más básico de todos y es una de las primeras en ser utilizada, sus tres niveles principales se definen como:

Nivel de Percepción: Es la encargada de identificar todos los objetos cuya función es recaudar información e interactuar con el mundo físico; además de identificar los dispositivos inteligentes que se puedan encontrar en el entorno. En este nivel estarán ubicados los sensores, actuadores y dispositivos digitales cada uno con un identificador único universal que permitirá su rastreo en el medio digital [2], [4].

Nivel de Red: Se considera la capa central de IoT, ya que aquí se encuentran ubicadas las diferentes redes que dan soporte a esta tecnología; su función principal es procesar y transmitir la información obtenida de los dispositivos. Adicionalmente hace posible la interconexión de los objetos a dispositivos de red o servidores, para ello dispone de herramientas que permitan soportar conexiones ethernet, red celular, red satelital, Wi-Fi, WiMAX, FTTx, xDSL, Cable modem [2], [4].

Nivel de Aplicación: Es la responsable de entregar los servicios a los usuarios finales de IoT; la esencia de esta capa es utilizar la información brindada por el sistema para darle valor a la misma. Aquí se pueden encontrar desde las aplicaciones domésticas sobre el uso de recursos hasta aplicaciones logísticas, es decir, todas las que hacen posibles tener las *Smart home*, *cities*, *health* entre otras [2], [4].

1.1.3. Conexiones y Modelos de Comunicación

Máquina a Máquina (M2M, *Machine to Machine*): es la conexión que se da entre máquinas y permite el intercambio de información entre los dispositivos conectados de manera automática facilitando así el desarrollo de las actividades [3], [4].

Máquina a Persona (M2P, *Machine to Person*): este tipo de conexión permite que los sistemas técnicos interactúen con las personas y las organizaciones para así tener un intercambio de información en ambas direcciones [3], [4].

Persona a Persona (P2P, *Person to Person*): aquí se hace un alto consumo de los recursos del ecosistema IoT, ya que la conexión de datos es bidireccional y se aprovechan capacidades de los dispositivos e infraestructura para lograr una colaboración exitosa entre personas [3], [4].

Comunicación de dispositivos a puerta de enlace: para poder acceder a la nube los dispositivos tienen que operar el *software* o las aplicaciones y tener un intermediario que les permita la conexión y la puerta de enlace es la que permite realizar la traducción de protocolos y garantiza la seguridad [3], [4].

Comunicación de dispositivo a la nube: la comunicación aquí se da directamente del dispositivo a un servicio en la nube lo que permite tener un acceso remoto a los dispositivos con facilidad de actualización de *software* y en la compatibilidad de este [3], [4].

Comunicaciones de dispositivo a dispositivo: dos o más dispositivos se comunican directamente entre sí por medio de diferentes redes [3], [4].

Intercambio de datos de *Back-End*: se presenta cuando un dispositivo se conecta a la nube para que una persona autorizada pueda acceder a los dispositivos o a los datos de los sensores y así poder exportar los datos de los objetos [3], [4].

1.2. Redes Definidas por Software

Las Redes Definidas por Software (SDN, *Software Defined Networking*) llegan para renovar la arquitectura tradicional, donde los elementos de red son administrados de forma individual y propone un nuevo modelo, en el cual se hace la separación del plano de datos del plano de control, dejando el plano de datos únicamente con la función de reenvió de información y concentrando toda la inteligencia de la red en el plano control [5].

El principal cambio que presenta el enfoque SDN es la alteración en la capa de red que es donde se hace el control de tráfico; esta capa pasa a estar distribuida entre el hardware y la capa de software de tal manera que los elementos de red pasan a ser los responsables únicamente del encaminamiento físico de los paquetes mientras que el control de enrutamiento es realizado por el software en la capa superior. Este desvinculamiento permite tener un control programable y hace de la nueva propuesta de arquitectura de red un modelo dinámico, manejable, rentable y adaptable; en la Figura 1-3 se observa el paralelo de las redes tradicionales con el modelo SDN, resaltando el cambio que se presenta en el plano de datos: de tener un controlador por cada dispositivo, a tener un solo controlador para todos ellos, dejando así que los dispositivo trabajen únicamente en el reenvío de información y permitiéndoles tener interfaces de programación de aplicaciones en su sistema operativo, facilitando la creación de redes controladas por programación [6].

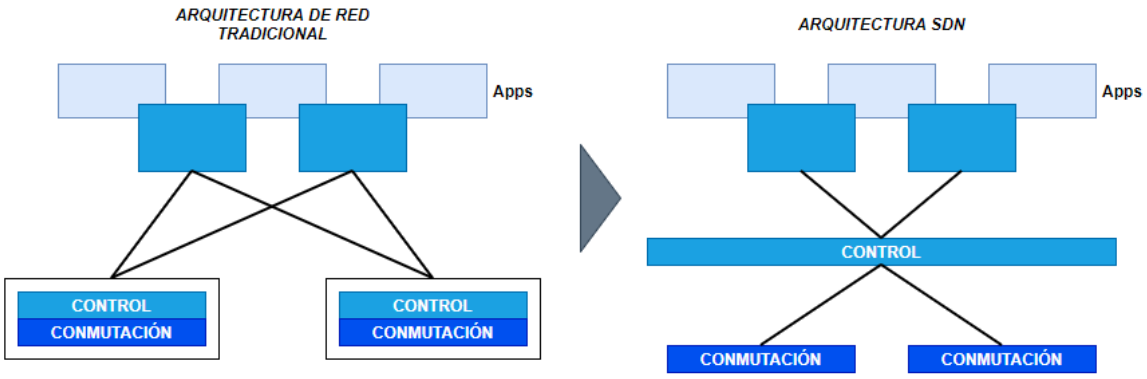

Figura 1-3 Comparación del Modelo Tradicional y el Modelo SDN [5]

1.2.1. Arquitectura

La arquitectura de las SDN está basada en un modelado de tres capas mostrado en la Figura 1-4, el nivel de infraestructura, el nivel de control y el nivel de aplicación; con lo que se reinventa y automatiza la infraestructura de red de manera óptima. Adicional a esto la arquitectura también se compone de API's las que permiten comunicar los tres niveles [5], [7], [8].

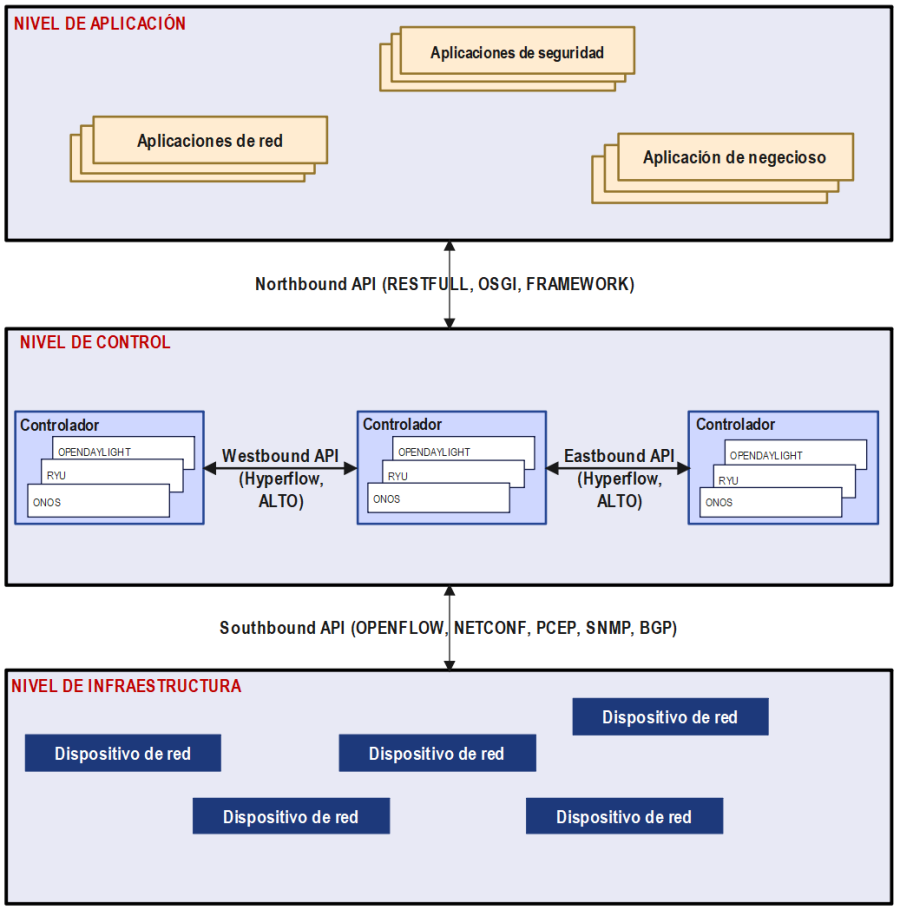

Figura 1-4 Arquitectura SDN [Elaboración Propia]

Nivel de infraestructura: Este nivel está conformado por todos los dispositivos físicos o virtuales, como los switches, enrutadores y puntos de acceso, los cuales son los encargados de transportar toda la información de los usuarios. Hay que tener en cuenta que en esta capa los dispositivos ya no poseen una funcionalidad predefinida y fija, sino que por el contrario aquí se caracterizan por tener un conjunto de instrucciones dadas por el plano de control y de esa manera un mismo hardware puede

funcionar como enrutador o como cortafuego según como se haya definido en el gestor de la red [5], [7], [8].

Nivel de control: Esta es la parte de la arquitectura donde se considera reside la inteligencia de la red, ya que automatiza las funcionalidades dentro de esta, es decir, eliminar o restaurar circuitos una vez solucionado el fallo; también abarca los protocolos de señalización, cálculos de enrutamiento, entre otros. En este nivel también aparece un nuevo componente, conocido como controlador el cual es el encargado de implementar de forma centralizada la administración de la red, permitiendo manejar flujos teniendo en cuenta los distintos patrones; adicional a esto también contiene las políticas de reenvió de datos, tablas de flujo, y tiene una visión general de toda la red; dentro de la gran variedad de controladores se encuentran RYU, Floodlight, OpenDaylight, ONOS, entre otros [5], [7], [8].

Nivel de aplicación: aquí se localizan las aplicaciones, las cuales comunican sus requisitos mediante el uso de API, que son las que permiten conectarse con la capa de control y están diseñadas para satisfacer las necesidades de los usuarios y permite la interacción con toda la arquitectura de forma rápida, ya que esta capa tiene el conocimiento de la cantidad y distribución de los dispositivos conectados hasta la recolección de estadísticas de comportamiento de la red, que es donde se toman decisiones respecto a la administración. Finalmente hay que tener en cuenta que el desarrollo de estas aplicaciones se realiza con código abierto para aportar a la estandarización agregando características de seguridad y portabilidad, en lenguajes como JavaScript, JSON7, Python, entre otros [5], [7], [8].

Para establecer una comunicación entre las diferentes capas descritas anteriormente, se hace uso de aplicaciones o interfaces, para comunicar la capa de control con la capa de infraestructura se utiliza una interfaz hacia el sur o Southbound, y para comunicar el nivel de control con el nivel de aplicación se usa una interfaz hacia el norte o una Northbound; ahora si se requiere implementar un sistema distribuido el cual se explica más abajo en esta sección, se requiere incorporar dos interfaces más que permitan realizar la conexión hacia los costados, estas interfaces son las Eastbound y Westbound, estas 4 interfaces adicionales se describen a continuación.

Interfaz *Southbound*: ayuda a la comunicación entre el controlador SDN y los dispositivos de red en la capa de infraestructura. Su objetivo principal es permitirle al controlador comunicarse con los elementos de conmutación de la red y programas la lógica de las comunicaciones en el hardware. El protocolo usado para este tipo de interacción es OpenFlow, el cual es un protocolo emergente y abierto de comunicaciones estándar definido entre los planos de datos y control [5], [7], [8].

Interfaz *Northbound*: Permite la interacción entre las aplicaciones externas y el controlador; ahora bien, el tipo de información que intercambian estas dos capas dependerá de cada aplicación en cuestión, es decir, no hay una estandarización para esta interacción [5], [7], [8].

Interfaz *Eastbound* y *Westbound*: Este tipo de interfaces se usan cuando se tienen controladores distribuidos y ayudan a facilitar la comunicación entre los controladores y compartir la información de control; también permiten la interconexión entre redes convencionales y las redes SDN; entre sus funciones se encuentra exportar, importar datos entre controladores, proveer algoritmos para modelos de consistencia de datos y capacidades de monitoreo para tener compatibilidad e interoperabilidad [5], [7], [8].

1.2.2. Ventajas

Las redes SDN permiten realizar la configuración, administración, optimización y protección de la red de manera flexible, todo esto debido a una serie ventajas que tiene esta tecnología sobre las demás y las cuales se mencionan a continuación:

Automatización: da la facilidad a las redes de realizar cambios y actuar según los servidores, es decir, se podrán inicializar o desactivar de forma ágil según se requiera con una intervención humana mínima [8], [9].

Escalabilidad: al hacer uso de túneles y redes virtuales se pueden contener un número razonable de dispositivos en un dominio de Broadcast, ayudando así a mejorar las limitaciones que trae consigo el tamaño de las tablas de direcciones [8, 9].

Seguridad: las SDN vienen acompañadas de una seguridad centralizada; el controlador proporciona un punto central de control para distribuir las políticas de seguridad de manera óptima [8], [9].

Virtualización: esto implica tener una abstracción virtual de una red ejecutándose arriba de la red física, lo que permite que el administrador sea capaz de manejar la red dese cualquier parte y en cualquier momento [8], [9].

Flexibilidad: al centralizar el control se mejora la flexibilidad en la gestión de los datos, permitiendo la modificación de la red en tiempo real para hacer frente a todo tipo de demanda [8], [9].

En la Tabla 1-2 se realiza un paralelo donde se resaltan las ventajas que tienen las SDN sobre la arquitectura de red tradicional.

Tabla 1-2 Cuadro comparativo entre las redes tradicionales y SDN [5]

REDES TRADICIONALES	***SDN***
Con políticas difíciles de operar y es relativamente estática	Redes completamente dinámicas
Tablas de enrutamiento cerradas	Tablas de enrutamiento abierta
La configuración de los dispositivos se hace de forma individual.	La configuración es centralizada
Es de difícil automatización	Fácil automatización
Alto costo operativo y financier**o**	Bajos costo operativo y financiero
Él envió de paquetes se hace basado en las coincidencias de la tabla	Usa formato y acciones de las tablas claramente especificadas
Interfaces propietarias	Interfaces definidas y abiertas
No es escalable	Escalable
Requiere de mucho tiempo para crear nuevas aplicacione**s**	Se tienen nuevas aplicaciones en un tiempo corto
No son flexibles	Son flexibles

1.2.3. Elementos que Integran una SDN

A. Dispositivos

Los dispositivos de las SDN están compuestos por una interfaz de comunicaciones con el controlador, una capa de abstracción y una función de procesamiento de

paquetes, esta estructura se observa en la Figura 1-5; donde se ve que cuando entra un paquete al dispositivo, este realiza una búsqueda en las tablas de flujo, actividad a cargo de la función de procesamiento; luego selecciona la entrada que tenga la mayor prioridad y realiza las acciones que están especificadas en la tabla de flujo. Si por casualidad no se encuentra la entrada para procesar el paquete, este es copiado al controlador para que realice las acciones correspondientes [9], [10].

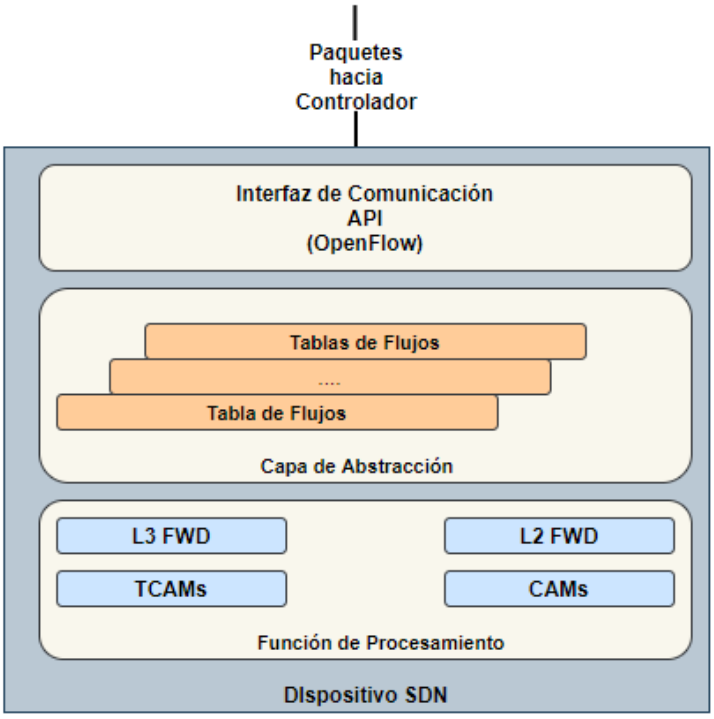

Figura 1-5 Estructura de un dispositivo SDN [10]

B. Protocolo *OpenFlow*

Esta herramienta surge de la necesidad de abstraer el control de los datos de hardware de los equipos de conmutación e implementarlo en una única entidad de control dentro de la red para luego aplicarlas a todos los dispositivos, esto con el fin de eliminar la administración individual de cada elemento y seguir el concepto de SDN de administrar la red como un todo [5], [8], [9].

Teniendo en cuenta que el tráfico de red se basa en reglas predefinidas, openflow permite que dichas reglas sean programas en el controlador, ya sea de forma estática o dinámica según sea el caso, lo que le permite a la red reaccionar en tiempo real a

nivel de aplicación, usuario y sesión, resaltando las funciones principales del protocolo que son: permitirles a los usuarios que definan los flujos de datos y determinen el camino que deben seguir en la red. Esta herramienta lanzó su primera versión en el año 2009, pero con el tiempo las funciones que tenía se han venido actualizando y mejorando, dando paso a que hoy en día existan alrededor de 5 versiones de este protocolo, los cuales se encuentran resumidos en la Figura 1-6 con sus respectivas características [5], [8], [9].

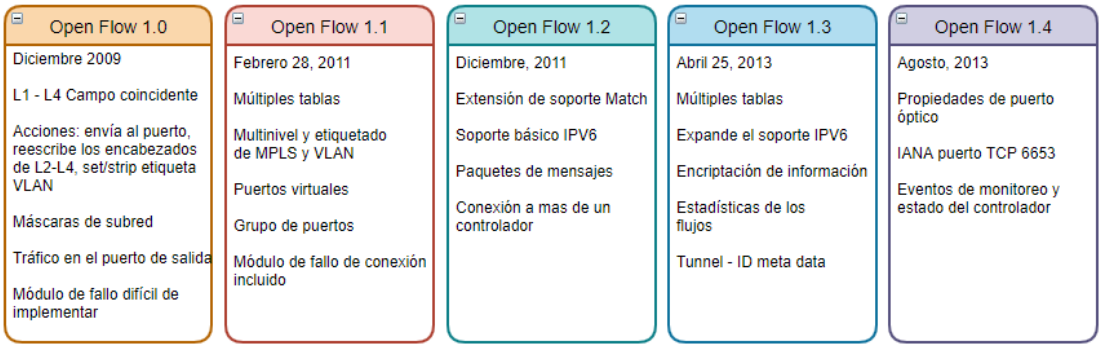

Figura 1-6 Versiones de OpenFlow [6]

C. Controladores

Cómo se presentó en la Figura 1-4 uno de los niveles que conforman la arquitectura SDN es el de control, aquí se encuentra ubicado el software controlador SDN, el cual es el cerebro de la red y controla todos los dispositivos que componen la infraestructura por medio de una interfaz en dirección sur, adicional a eso, también cuenta con una interfaz en dirección norte que permite la interacción con el nivel de aplicación, en la Figura 1-7 se aprecia cómo está compuesto este dispositivo [10], [11].

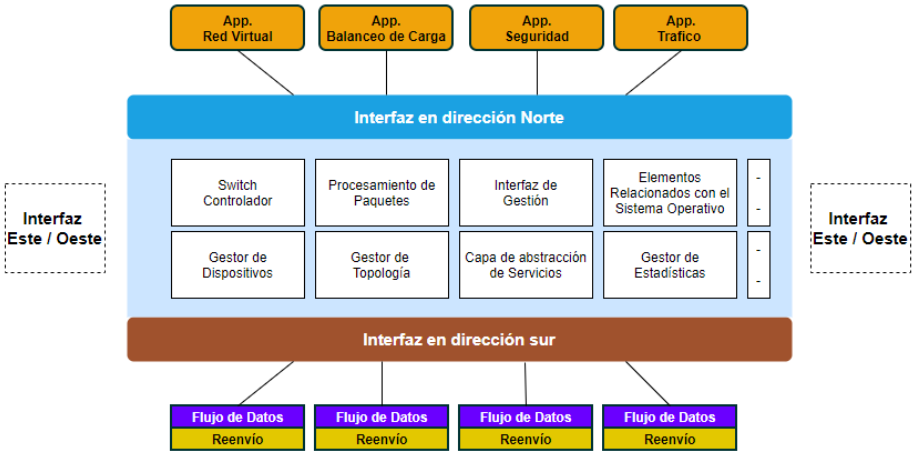

Figura 1-7 Arquitectura de un Controlador SDN [11].

El controlador es quien centraliza todas las configuraciones de los elementos de la red y funciona como una especie de sistema operativo, puesto que tiene la capacidad de tener una visión global de toda la red e implementa las políticas de seguridad, enrutamiento, redirección, balanceo de carga, entre otros. Una de sus funciones es traducir las necesidades o requerimientos de la capa de aplicaciones a los elementos de la red haciendo uso de interfaz norte y a su vez proporcionar la información relevante a las aplicaciones [10], [11].

Dependiendo de la dimensión de la red se pueden diseñar esquemas centralizados o distribuidos. Los diseños centralizados cuentan con un solo controlador que va a enviar información a todos los dispositivos y tiene la administración de toda la red, la gran ventaja que tiene este modelo es que la operabilidad se vuelve más sencilla, sin embargo, al afectarse se perdería el control de toda la red, por esta razón solo se recomienda su uso para modelos de red sencillos y pequeños. Por otra parte, en el control distribuido se tiene más de un controlador y todos ellos se comunican y envían toda la información funcionando como un solo sistema centralizado, en el que cada uno de ellos se concentrará en administrar los dispositivos a su cargo, en la Figura 1-8 se observa el esquema de ambos diseños [12].

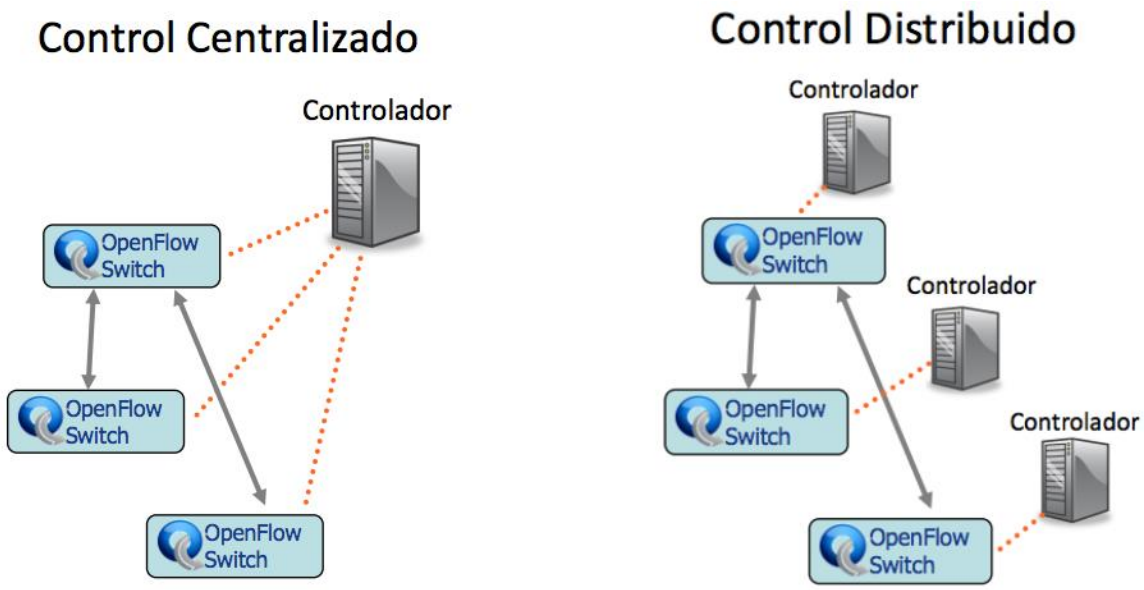

Figura 1-8 Esquema del Control Centralizado y Distribuido [12]

A lo largo del crecimiento y evolución de las redes definidas por software, se han desarrollado múltiples controladores; se debe tener en cuenta que independiente del fabricante, el software diseñado debe cumplir con un mínimo de características generales, las cuales se mencionan a continuación [13].

Versión de protocolo OpenFlow: establece que versión del protocolo maneja el controlador, de esta versión dependen funcionalidades como el soporte de IPV4 y/o IPV6 y soporte para enlaces ópticos [13].

API hacia el sur: esta interfaz da la posibilidad de tener conexiones con equipos de configuración híbrida, es decir que tengan ethernet y SDN, además de habilitar la conexión con redes virtuales con funcionalidades en la nube. También se indican los protocolos de comunicación con el nivel de infraestructura y de manera adicional también pueden soportar ARP, DHCP, BGP, entre otros [13].

API hacia el norte: señala que tipo de lenguajes soporta el controlador para establecer un puente de comunicación con la capa de aplicaciones [13].

Tipo de Interfaz: indica que clase de interfaz soporta el controlador para establecer la comunicación con los usuarios; dicha interfaz puede ser con comandos CLI o puede ser una interfaz gráfica [13].

Aplicaciones: establece si el software cuenta con aplicaciones de enrutamiento, monitoreo, medición, seguridad, integración con la nube, entre otras que faciliten el seguimiento de la red, los usuarios y servicios [13].

Máximo número de flujos soportados: este parámetro establece el número máximo de conexiones que puede soportar el controlador [13].

Código abierto o propietario: si el controlador está desarrollado en código privado o abierto, el cual da la posibilidad a los usuarios de tener acceso a su fuente para que pueda ser estudiado y modificado sin inconvenientes [13].

Soporta Multihilos: este parámetro le indica al usuario si el controlador soporta procesos simultáneos al trabajar bajo una arquitectura multihilo, lo cual permite reducir el tiempo de procesamiento de las tareas [13].

Consistencia de la información: permite al controlador garantizar la información de la red y asegurar desplegar la misma cuando esta sea necesaria para ejecutarla en todos los nodos de manera simultánea con el fin de evitar los errores de configuración al retomar el control de la red en caso de que se presente una pérdida de comunicación de los equipos [13].

Sistema Distribuido o Centralizado: establece si el software soporta un diseño centralizado permitiendo tener una alta consistencia de la información de la red; o si soporta el diseño distribuido dando a la red una mayor flexibilidad y resistencia ante una falla en el controlador [13].

Tolerancia a Fallos: establece si el controlador tiene la capacidad de responder ante las fallas y recuperación de los componentes, logrando las menores interrupciones posibles [13].

Tipo de licencia de uso: indica si el desarrollador le da permisos al usuario para acceder a él. Existen varios tipos de licencias, la primera en la licencia general pública o GPL el cual permite acceder al código fuente del software para realizar modificaciones y redistribuirlo bajo la misma licencia de uso; otra forma de licencia es la Apache y el software de distribución Berkeley BSD los cuales tienen los mismo

permisos que la anterior con la diferencia que esta no obliga a distribuir las modificación bajo la misma licencia; ahora bien, hay otras licencias que permiten tener una combinación de código abierto y privado los cuales exigen la publicación del código fuente solo cuando este es considerado una derivación del original, además prohíben la modificación y distribución del software, por tanto su uso solo se hará bajo las indicaciones del desarrollador, estas licencias son la licencia pública eclipse o EPL y la licencia pública general lesser o GNU [13].

Otras consideraciones para tener en cuenta a la hora de seleccionar un controlador son: el lenguaje en el que está desarrollado, así como la versión y actualizaciones que este haya tenido y si cuenta con buena documentación [13].

Dadas las características mencionadas anteriormente, las cuales se resumen para un mejor entendimiento en la Tabla 1-3, se encuentra aproximadamente 35 controladores diferentes; sin embargo, muchos de ellos están inactivos o son de poco uso debido a la falta de actualización por parte de sus desarrolladores o la baja consistencia. Lo que deja solo pocos software que cumplen con la mayoría de las características mencionadas anteriormente, entre ellos están OpenDaylight, ONOS y Ryu que resaltan por ser los más empleados para el desarrollo de soluciones SDN, en la Tabla 1-4 se encuentra una tabla comparativa de estas propuestas [13].

Tabla 1-3 Características técnicas generales de los Controladores [13]

Servicios del Controlador	Conexiones del Controlador	Vigencia del Controlador	Características del Controlador	Desarrollo del controlador
Número de Aplicaciones de enrutamiento	Interfaces de aplicaciones en dirección sur	Si el software tiene página web	Consistencia de la información encontrada	Fabricante
Número de Aplicaciones de Medición y Monitoreo	Interfaces de aplicaciones en dirección norte	Fecha de la última actualización de la página web	Versiones del protocolo OpenFlow soportadas	Lenguaje de Programación
Número de Aplicaciones de Seguridad y confianza	Ambientes de uso		Si presenta tolerancia a fallos	Nivel de Documentación
Tipos de Interfaz de usuario	Integración con la nube y virtualización		Si soporta Multihilos	Sistema Operativo

	Sistema es Distribuido o Centralizado		Cuál es el máximo de número de flujos soportados	Tipo de licencia de uso y permisos
				Si es código abierto o propietario

Tabla 1-4 Cuadro comparativo de controladores

Controlador	Licencia	Arquitectura	N. APP	S. APP	Lenguaje
Floodlight	Apache-2.0	Centralizada	Java API, REST API	OpenFlow	Java
Onix	Comercial	Distribuida	NIB API	OpenFlow, OVSDB	C++
OpenDaylight	EPL v1.0	Distribuida	REST API, JAVA API	OpenFlow, BGP, Netconf, OVSDB	Java
ONOS	GPLv3	Distribuida	REST API, JAVA API	OpenFlow, NETCONF, BGP, SNMP	Java
Ryu	Apache-2.0	Centralizada	Python API	OpenFlow, Netconf, OVSDB, BGP	Python
POX	GPLv3	Centralizada	Python	OpenFlow	Python

1.3. IoT basado en SDN

Dado al amplio número de dispositivos conectados a Internet, es necesario que la infraestructura de IoT sea escalable y a la vez permita la automatización de protocolos que permitan la administración de los datos obtenidos por medio de los dispositivos garantizando la seguridad. Las SDN permiten una solución alternativa para solventar los desafíos presentados por IoT. Enfocándose específicamente en los inconvenientes presentados a nivel de red en la tecnología IoT, algunos de los beneficios que tiene SDN son los siguientes:

Interoperabilidad

El tener tantos dispositivos conectados cada uno con diferente tecnología requiere el manejo de diversos protocolos para el intercambio de datos, lo que genera una falta de cooperación y coincidencia de capacidad entre los dispositivos obstaculizando el

rendimiento de la red. SDN ofrece la flexibilidad que se necesita, pues permite que diferentes objetos se comuniquen entre sí en redes heterogéneas logrando conexiones simultaneas y dejando decisiones de administración, enrutamiento, programación a cargo del controlador [14]–[17].

Descubrimiento

Otro inconveniente presentado en la tecnología IoT es la detección de dispositivos, el cual es un factor muy importante para implementar las aplicaciones de manera exitosa evitando las interrupciones y los errores de configuración. Las redes definidas por software le dan la capacidad al Internet de las cosas de adaptarse y autoconfigurarse al entorno con una intervención humana mínima o nula ya que configurar estos dispositivos manualmente es muy tedioso y no es factible [14]–[17].

Seguridad

En Internet existen muchos dispositivos conectados que al ser totalmente heterogéneos son propensos a ataques y en esta tecnología la protección de los datos es una característica para tomar en serio; SDN le permite a IoT afrontar estas dificultades de manera efectiva, permitiendo asegurar los datos al tener la opción de autentificación y autorización cuando se realiza una determinada entrega de datos; adicional a esto, las redes definidas dan la posibilidad de tener un modelo dinámico e inteligente de autoaprendizaje en temas de seguridad haciendo el Internet de las cosas más adaptable a medida que evoluciona [14]–[17].

Administración

Las SDN permiten tener un control generalizado de la red, ya que el controlador tiene la capacidad de tener la visión global de la topología y todas las características de la red como el enrutamiento, la calidad de servicio entre otros. Este software permite a la red tomar las mejores decisiones sobre el comportamiento de los datos y qué condiciones poner en las tablas de flujo permitiendo que los sensores se enfoquen en él envió de paquetes únicamente dependiendo de las condiciones establecidas previamente, reduciendo la latencia en los procesos y ahorrando energía [14]–[17].

Escalabilidad

A medida que IoT avanza se pueden observar la implementación de pequeños dispositivos como actuadores, sensores entre otros, y cada que la red crece, también se aumenta la cantidad de datos a procesar, por ende manejar el volumen de información es un desafío muy grande para esta tecnología; las SDN ayudan a mejorar en gran medida los inconvenientes de escalabilidad en las redes IoT, ya que es el software controlador quien se encargará de supervisar el dominio de la red ya sea de carácter centralizado o distribuido [14]–[17].

1.3.1. Arquitecturas SDN-IoT

Como se ha mencionado anteriormente, lo interesante de las SDN es la división del plano de control del plano de datos, facilitando la modificación de protocolos de red sin la necesidad de estar el dispositivo; con el paso del tiempo se han realizado varios esfuerzos por incorporar esta tecnología con IoT, centrándose en su mayoría en la recuperación y protección de los datos y en configurar dichas redes de manera remota. [16], [17].

Para poder lograr esta fusión, los autores en [16], [17] han desarrollado distintas arquitecturas, las cuales se caracterizan por hacer la recopilación de datos, para luego pasarlo por el nivel de procesamiento y finalmente entregarlos al usuario. Sin embargo, entre las propuestas encontradas tres de ellas sobresalen, puesto que su enfoque va más allá de la recopilación de datos y trabajan en función de mejorar el flujo de tráfico. La primera de ellas brinda la capacidad de proporcionar un objeto IoT para establecer una conexión con otro objeto que se encuentre en la red; aquí es el controlador quien establece la forma de reenvío y atiende las solicitudes de conexión. El objetivo de la segunda propuesta era implementar SDN para solventar problemas de big data, para ello, se analiza todo tipo de información que llega al controlador y así determina que paquete descartar, cabe a aclarar que las reglas de funcionamiento se pueden variar de forma dinámica. Finalmente, la última propuesta que se encontró interesante y se tomó como base para el desarrollo de este trabajo se explica a continuación [16], [17].

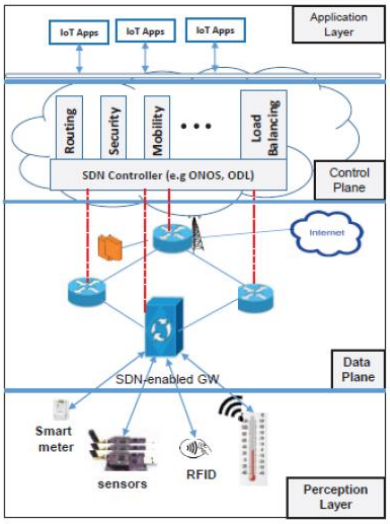

Figura 1-9 Arquitectura SDN-IoT [14]

La propuesta de arquitectura mostrada en la Figura 1-9, permite al administrador gestionar todos los servicios de la red a través del controlador SDN. Este modelo está divido en la capa de percepción, la capa de datos, la capa de control y la capa de aplicación. En la capa de datos se encuentran ubicados los enrutadores y conmutadores SDN, los cuales se encargan de recibir y transmitir la información que viene de la capa de percepción, aunque hay que aclarar que estos no toman ninguna decisión [16], [17].

La capa de control se comunica con los dispositivos por medio de interfaz en dirección sur haciendo uso del protocolo *OpenFlow*; adicional a ello, esta es la capa responsable de tomar las decisiones y tiene la visión global de toda la red por tanto es la de mayor importancia es esta arquitectura, otras de las funcionalidades de este nivel es el cálculo de reglas para el reenvío de paquetes, la planificación del enrutamiento, el equilibrio de carga entre otros. Lo que los autores en [14] querían conseguir con esta propuesta principalmente era tener flexibilidad para gestionar los datos, garantizar la seguridad y mantener la calidad de servicio, adicional a eso también busca garantizar la heterogeneidad de la red permitiendo el manejo de grandes volúmenes de tráfico que viene de los diferentes dispositivos conectados [16], [17].

1.4. Control de Tráfico

El concepto de IoT trae consigo la conexión a Internet de cientos de dispositivos a nivel mundial, desde juegos de cocina, electrodomésticos, automóviles, herramientas, dispositivos eléctricos y electrónicos, cuyo objetivo común es generar una relación entre el mundo físico y las tecnologías de la información; para poder lograr todo esto, IoT realiza una recopilación de toda clase de datos del entorno ya sea de fuentes únicas o múltiples, lo cual incide directamente sobre el rendimiento de la red, pues hace que en ocasiones las redes se saturen y disminuya su eficiencia al generar embotellamientos en los nodos u ocasionando perdidas de paquetes. Otro aspecto para tener en cuenta es que en un ambiente ya sea doméstico o empresarial la red tiene conectado todo tipo de dispositivos ya sean IoT o de otro tipo, como computadores, impresoras, teléfonos, los cuales también generan su propio tráfico, razón por la cual es necesario realizar una clasificación y priorización en la red para ejercer un control al realizar el envío de la información por el medio y contrarrestar las vulnerabilidades que puedan presentar [18].

Para empezar a hablar de control de tráfico en las redes IoT, primero es importante conocer cuál es el tipo de tráfico que se generan en esta tecnología. De forma general los dispositivos cuenta con dos fuentes de tráfico, el primero es el de comunicación tipo máquina (MTC, *Machine Type Communication*), este tráfico es generado por los dispositivos y está asociada a los servicios básicos de comunicaciones, como telefonía, servicios de difusión, videovigilancia, entre otros; el otro tipo es que la comunicación tipo humano (HTC, *Human Type Communication*), el cual es el tráfico generado a través de los dispositivos convencionales y están asociados a las comunicaciones entre dispositivos y sistemas o viceversa, como sensores, actuadores, bases de datos, entre otros. Realizando una comparación de estos dos tipos, se tiene que el MTC es más homogéneo que el HTC, esto quiere decir que aquellos dispositivos que tienen el mismo propósito generan tráfico con características similares [18].

Se debe tener claro que realizar una completa caracterización del tráfico es sumamente importante y beneficioso para llevar a cabo el control de este, puesto que, al identificar el comportamiento de los usuarios, se puede realizar una mejor distribución y administración de los recursos de la red y sus dispositivos; otro aspecto importante es lograr diferenciar el tráfico útil del anormal, permitiendo tener un mejor

control en la seguridad y adicional a ello garantizando que la información que es realmente importante para el usuario llegue más rápido. Dado que esta tecnología avanza a pasos agigantados, cada vez son más las formas de etiquetar los paquetes que surgen, donde cada uno de ellos es más preciso con el paso del tiempo y todos los modelos de control utilizan propiedades del tráfico que permiten modelar en términos de variables la llegada de paquetes, el tiempo de servicio en el sistema, entre otros [18].

1.4.1. Clasificación de Tráfico

Hoy en día existen múltiples formas de clasificar el tráfico que circula por la red; se pueden agrupar ya sea por el servicio al que están asociadas, según las características de transferencia o de calidad. En la Tabla 1-5 se muestra una serie de características de tráfico que pueden ser útiles para realizar la clasificación y las cuales están divididas en tres niveles; inicialmente se tiene el nivel de paquetes, donde el tráfico se genera cuando diferentes hosts se comunican entre si haciendo uso de protocolos de red; el tráfico a nivel de flujo son una colección de paquetes que comparten atributos; a nivel de conexión hace referencia al tráfico que se genera entre direcciones IP, es decir se describe la comunicaciones entre dos direcciones y en cuanto a nivel de host es el tráfico que se recopila de un host local [18].

Tabla 1-5 Características del Tráfico [18]

Tipo	Características	Descripción
NIVEL DE PAQUETES	1. Tamaño del Paquete	Indica la longitud del paquete en bytes
	2. Longitud del encabezado del paquete	Hace referencia a la longitud de la información de control de paquetes
	3. La cantidad de paquetes	Es el número de paquetes transmitido durante un cierto periodo de tiempo
	4. Hora entre llegadas del paquete	Es el intervalo de llegada del paquete, es decir, la hora de llegada del paquete
	5. Sello de tiempo	Es el momento en el que se envía o se recibe un paquete
	6. Tamaño de los primeros N bytes	La longitud de los primeros N bytes de un paquete
	7. Cadena específica	Son los bytes secuenciales específicos de la carga útil del paquete
	8. Valores estadísticos del tamaño del paquete	Es la desviación estándar máxima, mínima, media de la longitud del paquete

	9. Valores estadísticos del paquete en tiempo entre llegadas	Es la desviación estándar máxima, mínima, media de la hora entre llegadas del paquete
	10. Dirección de origen	Es la dirección IP de una interfaz de origen
	11. Dirección de Destino	Es la dirección IP de una interfaz de destino
	12. Puerto de Origen	Es el punto final de una interfaz fuente
	13. Puerto de Destino	Es el punto final de una interfaz de destino
	14. Protocolo	Es el protocolo usado por una aplicación
	15. TTL	Es el tiempo de vida
	16. Volumen de bytes	Corresponde al número total de bytes transmitidos
	17. Duración de Flujo	Es el intervalo entre las máscaras de tiempo del primer paquete y el último paquete
	18. Longitud de Flujo	Es el número total de paquetes en un flujo
	19. Tamaño de Flujo	Es la suma del tamaño de los paquetes contenidos en un flujo
	20. Tamaño de los primero N paquetes	Es la suma del tamaño de los paquetes contenidos en un flujo
NIVEL DE FLUJO	21. Valores Estadísticos del primero N tamaños de paquete	Es el máximo, mínimo, promedio, desviación estándar de la longitud de los primeros N paquetes
	22. Relación de Fuente y Bytes de destino	Es la proporción de los paquetes transmitidos entre un host de origen y un anfitrión de destino
	23. Min_iat	Es el paquete mínimo entre llegadas de un flujo
	24. Mean_iat	Es el paquete medio entre llegas de un flujo
	25. Var_iat	Es el paquete de varianza entre llegadas de un flujo
	26. Mean_data_ctrl	Son los bytes de control medio de un flujo
	27. Mean_data_wire	Es la media de bytes de paquetes de ethernet de un flujo
	28. Avg_wim_avg__c	Es el tamaño de la ventana promedio desde un origen hasta un destino
	29. Mean_data_wire_s	Representa la media de bytes de paquetes ethernet desde un destino hasta una fuente
	30. Mean_data_Ip_s	Es la media de bytes del paquete ip desde un destino a una fuente
	31. RDBUB	Es la relación ente los bytes descendentes y los bytes ascendentes
	32. APITD	Es el tiempo medio entre llegadas de paquetes en sentido descendente
	33. IEPSD	Hace referencia a la entrada de información del tamaño del paquete en sentido descendente
	34. NDSF	Es el número de subflujos descendente
NIVEL DE CONEXION	35. Duración de la Conexión	Es la duración del intervalo de tiempo durante el cual el túnel esta activo.

	36. Hora entre llegadas de la conexión	Es el tiempo transcurrido entre dos solicitudes consecutivas, estableciendo un túnel desde el mismo usuario y servidor.

Es importante tener en cuenta algunas características de los niveles mencionados, en primer lugar, los datos a nivel de paquetes están conformados por tres partes, el encabezado, la carga útil y la actividad de paquetes, es decir, este nivel hace referencia a la información que se puede obtener de un paquete. En cuanto al tráfico a nivel de flujo son aquellos paquetes que tienen las mismas cinco características que son la dirección IP destino y origen, el puerto de origen y destino y el protocolo; el nivel de conexión contiene los datos de entrada y salida, el número de conexiones realizadas y la duración de cada una de ellas. Todas estas particularidades son de mucha ayuda en la realización de la clasificación y priorización del tráfico que circula por la red, y estas son usados por los distintos métodos de clasificación que se explican a continuación [18].

Métodos de Clasificación

En cuanto a los métodos de clasificación, se consideran cinco categorías: a) la basada en estadísticas, la cual usa valores de tráfico estadísticos ya sea a nivel de flujo o de paquetes, b) la basada en correlación en la que se agregan paquetes a los flujos y se clasifican acorde a la relación existente entre los flujos de la red, lo que evita la redundancia de características, sin embargo tiene un alto consumo de recursos, c) la basada en el comportamiento, que da una perspectiva diferente, ya que realiza la clasificación de acuerdo a los comportamientos de los host, d) la basada en carga útil que es propuesta con el fin de mejorar la precisión, para esto se divide en dos subconjuntos que son la inspección profunda de paquetes (DPI, *Deep Packet Inspection*), la cual es una tecnología que detecta el tráfico y el contenido de los paquetes y dada su alta precisión es muy usada en aplicaciones de gestión de tráfico, seguridad y prevención de ataques y la inspección completa de paquetes (SPI, *Stateful Packet Inspection*), esta se propuso para poder clasificar los datos cifrados, puesto que no utiliza el contenido del paquete directamente sino que utiliza la información estadística de la carga útil, la desventaja de esto es que requiere de muchos recursos computacionales y finalmente c) la basada en puertos la cual realiza la identificación de las aplicaciones por el número de puertos utilizado, los cuales son asignados por la Autoridad de Números Asignados de Internet (IANA, *Internet Assigned Numbers*

Authority), pero con la cantidad de aplicaciones nuevas muchas de ellas eligen usar puertos dinámicos haciendo que este enfoque sea ineficiente y poco preciso [19].

1.4.2. Congestión

Una red IoT se compone de varios dispositivos, lo cuales generan su propio tráfico en la red de forma simultánea, provocando una pérdida de paquetes y con ello una congestión en el sistema. Actualmente se encuentran distintos métodos de control de congestión; sin embargo, en [20] se hace una preselección de tres de ellos lo cuales resaltan dada su diversidad, las pruebas realizadas, año de publicación y cantidad de citas; estos son: el método DALPAS, el cual es un algoritmo que hace un control de los recursos mediante el uso de caminos alternos; el método ECODA hace uso de una técnica de doble buffer para detectar la congestión en los nodos, con esto se permite aceptar o rechazar paquetes dependiendo de la estrategia; finalmente el método FUSION hace uso de control de flujo salto a salto para prevenir que los nodos transmitan paquetes que se van a descartar, y a su vez asegura que los nodos tengan acceso prioritario al canal al tener la MAC priorizada. En la Tabla 1-6 se puede ver que los tres métodos se diferencian tanto en la detección, notificación y mitigación de la congestión [20].

Tabla 1-6 Métodos de control de congestión [20]

Método	Detección de Congestión	Notificación de Congestión	Control de Congestión	Capas de Control	Ventajas	Desventajas
ECODA	Tamaño del buffer	Implícita	Control de Tráfico	Buffer Red	Muy buena administración del buffer Establecimiento de prioridades de paquetes	Únicamente se centra en manejo de buffer y no considera métodos adicionales
DALPAS	Tamaño del Buffer	Implícita	Control de Recursos	Red	Modifica la topología para convertirla en un árbol Tiene en cuenta la energía y disponibilidad del nodo para	Únicamente se centra en el manejo de la capa de red y no considera métodos adicionales

					decidir el enrutamiento	
FUSION	Tamaño del Buffer Carga del Canal	Implícita	Control de Tráfico	MAC Red Aplicación	Implementa el acceso prioritario al canal cuando hay congestión Involucra a la aplicación generadora de tráfico	El método utilizado puede llegar a llenar demasiado el buffer Cuando hay congestión en un nodo principal se detiene la trasmisión, afectando todo el funcionamiento del sistema

Un forma de realizar un control de tráfico en las redes y mitigar la congestión es establecer una prioridad a los paquetes que circulan por estas; dicha acción, se puede realizar teniendo en cuenta los distintos protocolos de enrutamiento, que fueron diseñados para ser tolerantes a fallos, disminuir el consumo de energía, entre muchos otros, que desafortunadamente según [20] no responden de forma favorable cuando se presenta congestión en la red; por otra parte las funciones presentadas en la Tabla 1-6 tiene sus propios mecanismos para realizar la priorización del tráfico circulante, ECODA, define un número entero a los paquetes para identificar los servicios, dicho número es constante en la prioridad estática, mientras que en la dinámica cambia con cada salto que da el paquetes de nodo a nodo, sin embargo su funcionamiento se limita solamente al buffer dejando de lado el estado de los enlaces y demás métodos [20].

El control de tráfico es una tareas fundamentales en las redes, y es por eso por lo que a la fecha se han realizado extensas investigaciones en una diversidad de contextos, dando como resultado técnicas como la optimización de rutas, donde se realizan configuraciones de enrutamiento teniendo en cuenta las condiciones de tráfico observadas previamente o con respecto a una gama de posibles escenarios de tráfico; sin embargo, los métodos vistos en esta sección pueden presentar fallas a la hora de obtener un rendimiento óptimo en condiciones reales; por tanto, el enfoque de machine learning propone una opción diferente para realizar la optimización de rutas y realizar

un control de tráfico, este nuevo enfoque consiste en utilizar sus distintas técnicas de para aprovechar la información que se tiene en el sistema para que este aprenda a configurarse de una mejor manera y así aplicarlo a futuro [21].

1.4.3. Protocolos de comunicaciones en IoT

En el apartado 1.1. Internet de las Cosas, se establece que el ambiente IoT está constituido por múltiples dispositivos conectados que funcionan de manera independiente, por esta razón no se puede especificar un único tipo de tráfico, ya que este dependerá de factores como tiempo de inactividad, si el dispositivo es unidifusión o multidifusión, y se configura para host locales o servidores de Internet, o si tiene vinculados un protocolo o varios, el número de puerto especificado, la tecnología utilizada, entre muchos otros. Cuando se habla de tráfico IoT, generalmente se hace referencia al envío esporádico de ráfagas cortas, donde la eficiencia de este dependerá del volumen del flujo, la duración de este; es de aclarar que algunos de los dispositivos IoT utilizados en un ambiente doméstico solo entran a un estado de inactividad cuando no hay conectividad, otros solo están en estado activo cuando haya un movimiento. Es por esto, por lo que además de los métodos de clasificación mencionados anteriormente, en [22] se planea una clasificación basada en el comportamiento del dispositivo ya sea que este esté iniciando, activo o inactivo. Por otra parte, en cuanto a tasas de transmisión se encuentra que los dispositivos IoT tiene tendencia a transferir un pequeño volumen de tráfico cada determinado instante de tiempo; normalmente se estable el promedio de descarga de los elementos IoT es menos de 500KB cada media hora en contraste con los no IoT que tiene un promedio de entre 10 KB y 10MB en el mismo intervalo de tiempo [22].

Dadas algunas restricciones que tienen los dispositivos IoT como las baterías, capacidad de almacenamiento y proceso, deben afrontar desafíos como el direccionamiento e identificación, la comunicación con bajo consumo de energía, protocolos de enrutamiento eficiente y con bajos requerimientos de memoria, alta velocidad y comunicaciones sin pérdidas y la movilidad. Generalmente estos dispositivos se conectan a Internet por medio de la pila TCP/IP o por medio de algunas redes no IP como RFID, NFC y Bluetooth. Por tanto, los protocolos de comunicación M2M se clasifican ya sea en función del rango o del consumo en capilares o celulares. Los dispositivos IoT suelen trabajar en bandas desde los 433MHz, 868 MHZ, 905 MHz

y 2,4 GHz; las bandas que están por debajo de los 433 MHz no son usadas frecuentemente en esta tecnología y por arriba de los 2,4 GHz se pueden hallar bandas libres disponibles [23].

Los sistemas máquina a máquina capilares son redes de área local que emplean tecnologías de corto alcance para brindar conectividad a un grupo de dispositivos; las topologías empleadas aquí generalmente son las de tipo estrella donde todos los dispositivos se conectan a un nodo central y entre ellos ya sea de forma directa o por medio de nodos intermedios. En la capa física de este sistema se especifican los parámetros mecánicos, eléctricos y las conexiones físicas; por su parte, en la capa de enlace se define la conexión entre dos dispositivos, mientras que las tecnologías usadas varían según la capa y se mencionan a continuación [23].

A. Bluetooth

Fue diseñado por la industria para establecer una conexión basada en radio frecuencia (RF); está definido por el estándar 802.15.1, es de corto alcance y trabaja con una frecuencia de 2.4 GHz con tasas hasta 1Mbps [23].

B. RFID

Esta es una tecnología diseñada y utilizada para la identificación de objetos a distancia sin la necesidad de que estos tengan contacto. Este sistema consiste en un microchip que opera junto con una antena de radio y lo que permite la identificación; esta tecnología trabaja en rangos de baja frecuencia en 125 KHz o 134 KHz [23].

C. NFC

Esta tecnología es de muy corto alcance, su funcionamiento se basa en la interacción de los dispositivos móviles a solo centímetros de distancia; opera en la banda de los 13,56 MHz y cuenta con dos modos de operación, el activo el cual es donde ambos dispositivos generan campos magnéticos y el modo pasivo, donde solo un dispositivo generará el campo magnético [23].

D. Wi-Fi

Está definido bajo el estándar 802.11 del Instituto de Ingenieros Eléctrico y Electrónicos, trabaja en las bandas de 2.4 GHz, 5GHz y 60 GHz. Actualmente cuenta con varias versiones de este estándar que son: el 802.11n, 802.11b, 802.11g, aunque estas versiones son compatibles entre sí, al combinarlas y admitir un entorno mixto las velocidades de datos se limitan [23].

Dentro de este estándar también se encuentra el 802.11ah o HaLow, la cual opera en frecuencia menores a 1GHz lo que permite una mayor gama de conectividad de dispositivos de baja potencia, además adopta otros protocolos Wi-Fi de tal forma que garantiza ventajas de tener conexiones más robustas, además que amplía la banda de trabajo de Wi-Fi a los 900 MHz [23].

E. Z-Wave

Está basada en RF y fue diseñada para controlar, monitorear y comprobar el estado de las aplicaciones de entornos residenciales y comerciarles estrictamente. Esta tecnología permite una transmisión segura y con bajas tasas de velocidad, es adecuada para pequeños paquetes de datos que van hasta los 100 Kbps; a diferencia de las otras tecnologías esta mantiene una distancia máxima de cien metros, opera en la banda de los 868 MHz [23].

F. WPAN

Definido por el estándar 802.15.4 en el nivel físico y el control de acceso de las redes de área personal de corto alcance. Entre sus características se encuentra proporcionar comunicaciones de bajo costo, tiene un alcance de 10 m con una tasa de 250 kbps y se puede utilizar en tres bandas 868 MHz, 915 MHz y 2.45 GHz [23].

1.5. *Machine Learning*

Como se mencionó anteriormente aprovechar la información para que el sistema aprenda a configurarse de forma óptima y realice un buen control de tráfico en condiciones futuras es una tarea fundamental a la hora de optimizar el funcionamiento de las redes IoT; este tipo de aprendizaje se realiza por medio del Aprendizaje Automático (ML, *Machine Learning*) cuyo objetivo principal es generar o diseñar algoritmos con la capacidad aprender e identificar patrones en los datos y sean capaces de tomar decisiones sin la necesidad de ser programables. Dentro de este mecanismo se presentan dos opciones para realizar la labor de control, el primero es el aprendizaje supervisado donde se observan las demandas de tráfico y se aplican los algoritmos para predecir las próximas demandas para finalmente optimizar el enrutamiento con respecto a las datos previos; el segundo método es el aprendizaje por refuerzo, el cual en lugar de aprender explícitamente las demandas futuras de tráfico y optimizar con respecto a estas, busca aprender de un mapeo del historial observado; a continuación se definen estas dos ramas del ML [24].

1.5.1. Clasificación De *Machine Learning*

En la Figura 1-10 se observa que ML dispone de tres subconjuntos o modelos de aprendizaje, cada uno cuenta con características y algoritmos diferentes, que son aplicables dependiendo del tipo de investigación o problema a tratar, estos modelos son: el aprendizaje supervisado, no supervisado y reforzado [24], [25].

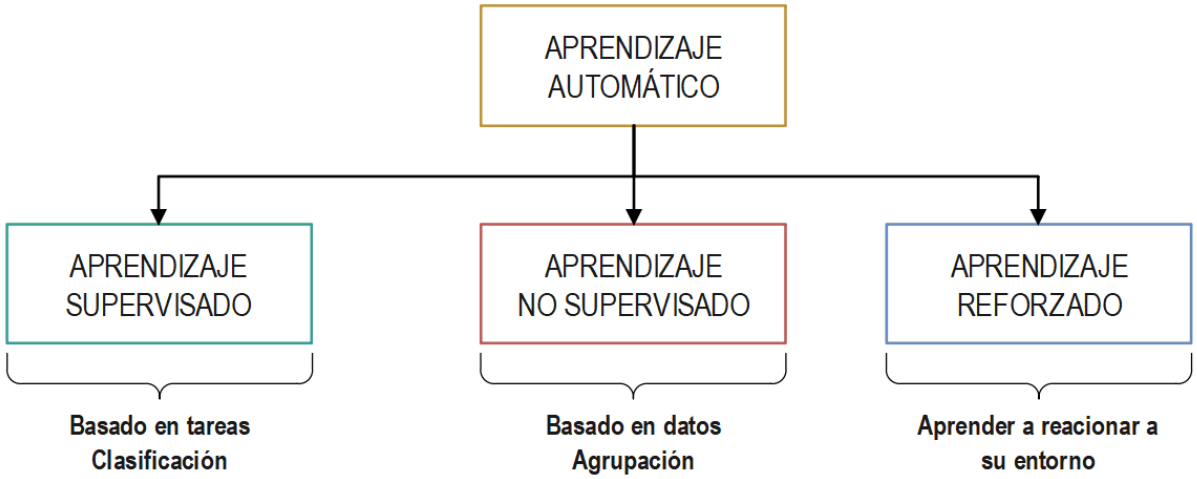

Figura 1-10 Tipos de Aprendizaje automático [Elaboración de los autores]

A. Aprendizaje Supervisado

El objetivo principal del aprendizaje supervisado es la predicción; en este orden de ideas, estos algoritmos de ML son entrenados con datos de entrada y salida, es decir aprenden a partir de una base de conocimiento previo mediante un grupo características y etiquetas para su funcionamiento, de esta forma el algoritmo es entrenado por una persona para ser capaz de identificar y hacer una predicción correcta con el comportamiento de nuevos datos [25].

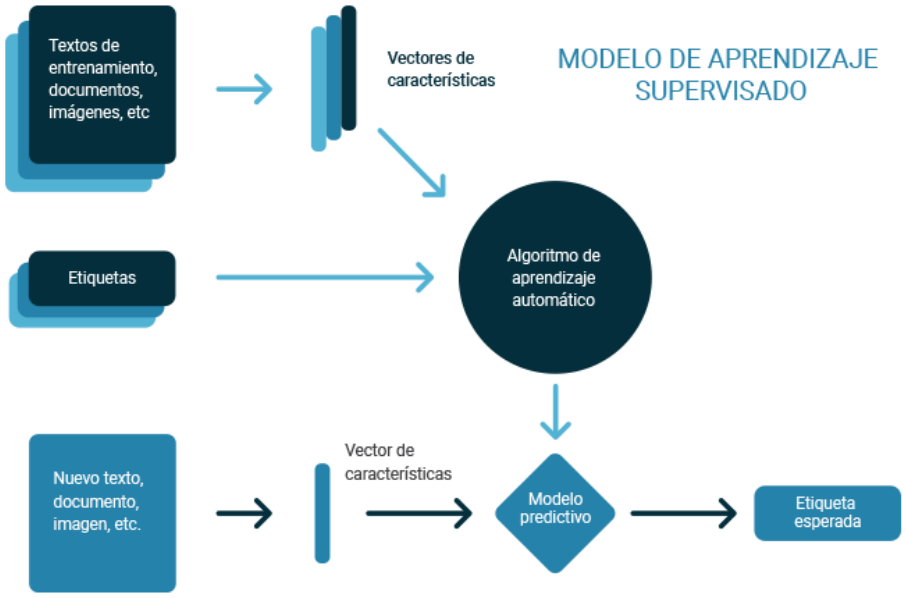

Figura 1-11 Diagrama de flujo y ejemplo del modelo supervisado [25], [30]

En la Figura 1-11 se muestra al lado izquierdo como sería el diagrama de flujo de este modelo de aprendizaje, indicando que al pasar la información, también se deben pasar ciertas etiquetas para que el sistema realice la predicción de acuerdo a los parámetros establecidos; al lado derecho de la imagen se observa un ejemplo de cómo una persona le enseña a la máquina los datos de entrada y cómo clasificarlos, una vez el algoritmo ya haya sido entrenado recibirá los datos de entrada y este podrá clasificarlos. Ahora bien, dentro del aprendizaje supervisado existen dos modelos:

Regresión

Este método devuelve un número, prediciendo una cantidad o valor específico lo que lo hace útil para pronósticos y predicciones. Este tipo de algoritmo se enfoca en

modelar la relación que hay entre una variable dependiente y una serie de variables independientes, es decir, el resultado final de este tipo de algoritmos será un valor numérico dentro de un conjunto infinito de posibles resultados. Algunos de las técnicas utilizadas para regresión son:

- **Regresión lineal:** este método se utiliza para predecir la variable dependiente en función de los valores de las variables independientes, en otras palaras, consiste en hallar una línea recta que encaje de la mejor manera en un conjunto de datos dados; suele usarse para casos donde se quiere predecir una cantidad continua como el tráfico de una tienda o el tiempo de permanencia de un usuario [24].
- **Regresión Polinomial:** Es similar el método anterior, sin embargo, en este caso los datos no son lineales; esencialmente es un algoritmo de ensayo y error donde se busca evaluar distintos grados de polinomios para obtener el que mejor se adapte a los datos. Suele usarse para predecir las medidas de una persona o el precio de una acción [24].
- **Máquinas de vectores de soporte:** El objetivo de este tipo de algoritmos es modelar una tendencia de datos de entrenamiento y de acuerdo con ello predecir cualquier dato futuro, se debe tener en cuenta que este modelo viene limitado por un rango por tanto todos los datos que estén fuera de este se consideran errores. Suele usarse para predecir la temperatura de un lugar, el cambio de precio de una acción, clasificación de textos y vehículos no tripulados entre otros [24].
- **Arboles de decisión regresión:** Este es uno de los métodos más utilizados y uno de los mejores, ya que potencia los modelos predictivos y facilitan la interpretación. Este proceso consiste en alinear todos los valores y probar distintos puntos de división utilizando distintos métodos y al final se evalúa y se elige el mejor resultado [24].
- **Bosques aleatorios regresión:** Este es un conjunto de árboles de decisión donde al final se obtiene un promedio del resultado de cada árbol que conforman en bosque. Suele usarse para predecir la enfermedad de un paciente, el precio de una acción y predecir la tasa de mortalidad entre otros [24].

Clasificación

Estos algoritmos predicen una etiqueta, compara los nuevos datos y los clasifica de acuerdo con sus características en grupo o categorías. Generalmente estos métodos usan las características aprendidas de los datos de entrenamiento para predecir las etiquetas de la clase. Los tipos de algoritmos de clasificación son los mismo de regresión, pero enfocados a la clasificación [24].

- **Regresión Logística:** este método de análisis predictivo es apropiado cuando se tienen una variable dependiente binaria, es de fácil interpretación y rápido de entrenamiento. Suele usarse para ordenar resultados por probabilidad o modelar respuestas de marketing [24].
- **K-Vecinos más cercanos:** este es uno de los algoritmos más simples y de fácil entendimiento, se basa en la similitud de las características. Suele usarse en la seguridad informática, corrección ortográfica, sistemas de reconocimiento, entre otros [24].
- **Máquina de vectores de soportes:** se basa en la construcción de un hiperplano óptimo en forma de superficie de decisión; es decir, partir de unos datos de entrenamiento se genera un hiperplano óptimo que clasifica los nuevos ejemplos en dos espacios dimensionales. Suele usarse para el reconocimiento de escritura a mano, entre otros [24].
- **Arboles de decisión clasificación:** es un diagrama de construcción lógica que sirve para representar y categorizar una serie de condiciones que ocurren de forma sucesiva para la resolución de un problema. Suele usarse para el diagnóstico médico y análisis de riesgo bancario [24].
- **Bosques aleatorios clasificación:** es una combinación de árboles de decisión individuales que operan como un conjunto, donde al final se toma la clasificación de cada árbol y el resultado final será la decisión con más votos. Suele usarse para problemas de bioinformática [24].

Finalmente, el aprendizaje supervisado proporciona una ruta directa que permite convertir datos en información real y procesable, además les facilita a las organizaciones o a quien lo use para procesar datos por medio de herramientas autónomas y así darles el mejor uso para tomar decisiones rápidas y precisas. El éxito de este tipo de algoritmos depende de que se cumplan cuatro fases que son: la fase

de entrenamiento, que es donde se realiza la recolección y preparación de datos, luego de tener los datos listos, se procede a elegir el modelo o algoritmo a utilizar para finalmente realizar el entrenamiento; la fase dos corresponde a la validación del modelo para verificar el funcionamiento y realizar los respectivos a justes antes de pasar a la tercera fase que es la de pruebas, donde con nuevo conjuntos de datos se corre el algoritmo seleccionado para que tome decisiones [26]–[29].

B. Aprendizaje no supervisado

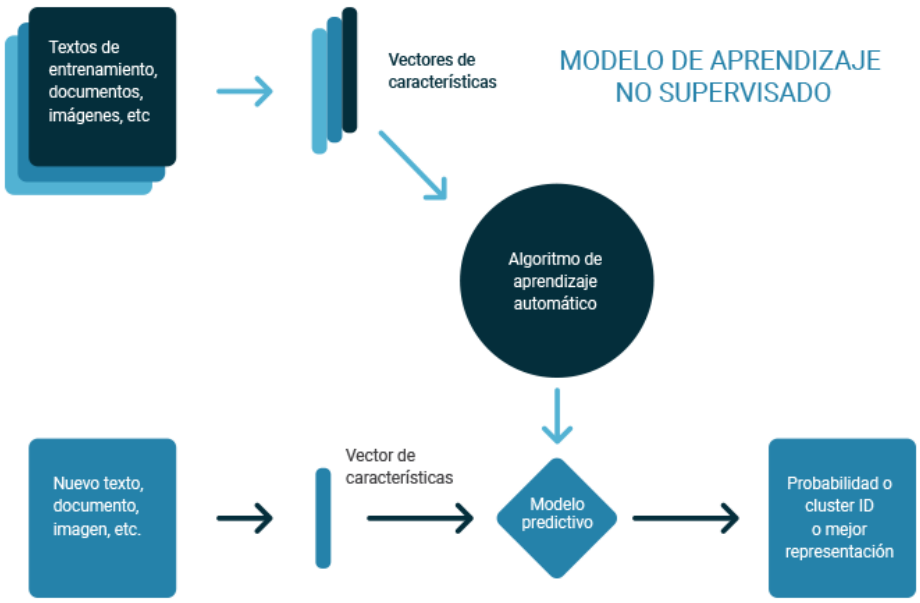

Figura 1-12 Secuencia de Flujo del Aprendizaje no Supervisado [24, 25]

Este tipo de aprendizaje es otra rama del ML que no requieren de un conocimiento previo ni de intervención humana; en la Figura 1-12 se muestra la secuencia lógica que tiene este tipo de algoritmos donde la predicción se realiza de acuerdo con los patrones que encuentre o la relación que exista entre ellos dentro del conjunto de datos; por tanto, dada su naturaleza son soluciones que no se pueden aplicar a problemas de clasificación o regresión, su enfoque se centra en realizar tareas de *Clustering*, donde lo que se busca es agrupar los datos acorde con las similitudes sin que estas agrupaciones tengan algún significado o utilidad; algunos de los métodos más utilizados aquí son:

- **Algoritmos de Clustering:** Su objetivo es agrupar un conjunto de objetos de tal forma que los objetos que queden dentro del conjunto sean lo más similares posibles [24].
- **Análisis de componentes principales:** Este es un proceso estocástico que se usa para realizar una transformación ortogonal y de esa forma convertir un conjunto de variables correlacionadas en no correlacionadas [24].
- **Descomposición en valores singulares:** Este es un sistema de factorización de una matriz compleja real [24].
- **Análisis de componentes principales:** Esta es una técnica estadística que se utiliza para revelar los factores ocultos que son la base de los conjuntos de variables [24].

C. Aprendizaje por Refuerzo

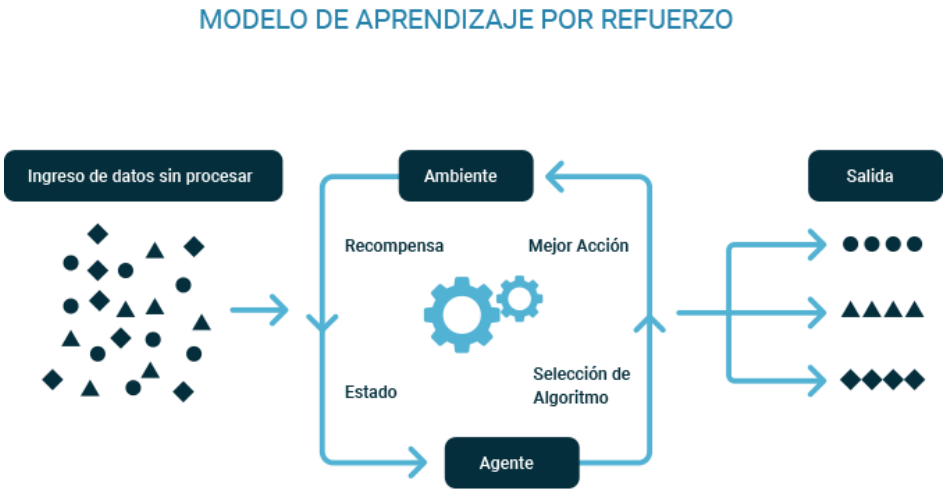

Figura 1-13 Diagrama de Flujo del Aprendizaje no Supervisado [24, 25]

Este modelo surge debido a que no todos los algoritmos de ML pueden encajar en el término de supervisado o no supervisado. En la Figura 1-13 se presenta el diagrama de flujo que sigue este modelo donde muestra que el enfoque de este tipo de algoritmo, cuyo procedimiento es: primero se tiene un agente el cual es quien interacciona con el entorno y será quien realice las acciones, seguido está la acción que afecta al estado del agente en el entorno, una vez ejecutada dicha acción se obtiene una recompensa,

la cual se limita a un señal que indica la conveniencia o no de las acciones que se realizaron y finalmente con todos los hechos se busca mejorar el comportamiento posterior [27, 28].

El mayor reto que enfrenta este tipo de aprendizaje es lograr el equilibrio entre la exploración y la explotación; es decir, el agente procura conseguir la mayor recompensa; sin embargo, para conseguir la mejor recompensa el agente debe repetir acciones con las que ya ha obtenido buenos resultados, a esto se le llama explotación; por otra parte, el agente también de explorar las acciones no elegidas con el fin de encontrar mejores recompensas. Otra característica importante del aprendizaje reforzado es que este tipo de aprendizaje el problema se considera como un iteración del agente con el entorno parcialmente conocido [31, 32].

Los elementos que hacen parte de estos algoritmos son: La política, que es la que permite determinar el comportamiento del agente en un momento o estado determinado, para entender un poco mejor, el quien ayuda a determinar la acción óptima para cada estado, como resultado devuelve o bien la acción a realizar o una tabla que contiene el estado y mejor acción; el segundo componente es la recompensa, en un número que el entorno entrega al agente en cada paso de la ejecución, al finalizar el objetivo del agente debe maximizar la recompensa total y así determinar cuáles de los estados son buenos y cuales negativos para el agente, estas recompensas se deben cuantificar en base a una función que tenga como entrada el estado del entorno y acciones seleccionadas. El tercer elemento que conforma este tipo de algoritmos es la función valor, que es igual a la recompensa máxima que el agente sea capaz de acumular desde el estado en el que se encuentra. Finalmente se tiene el modelo, el cual es un componente opcional que permite estudiar a que estados se llegan cuando se le eligen ciertas acciones al estar en un estado previo. Sin embargo, los algoritmos que más se resaltan son aquellos que son libres de modelos y se centran explícitamente en agentes de ensayo y error [32].

Dentro del aprendizaje por refuerzo libres de modelo se tiene el pasivo, el activo y el refuerzo activo aproximado.

Aprendizaje por Refuerzo Pasivo: En este caso el agente aprende de los observado pero no tiene ninguna decisión sobre la política, ya que esta se fija con anterioridad y

va a variar durante el proceso de entrenamiento; aquí, el agente tiene dos formas de aprenderse los valores de estados, por medio de una evaluación directa, la cual consiste en realizar experimentos y calcular el valor medio de la recompensa al terminar cada estado, o por medio de la diferencia temporal, la cual se consiste en estimar los valores de recompensa inmediata y futuras en una forma similar a la programación dinámica [32].

Aprendizaje por refuerzo activo y activo aproximado: A diferencia de los anteriores, estos se caracterizan por no tener un política fija, sino que esta se actualiza progresivamente y busca llevar el sistema a la una política óptima; algunos de los algoritmos que se derivan de este método son: [32]

- **SARSA:** se deriva del método basado en diferencia temporal, tiene una política inicial y la actualiza al final de cada episodio, es decir, la actualización se lleva a cabo en cada transición de un estado a otro. El funcionamiento de este método es el siguiente: Comienza con una política y función Q que se generan aleatoriamente, posterior a ello se ejecuta un episodio y en cada paso se actualizan los valores de los pares estado-acción visitados, al finalizar el episodio, la políticas actualizadas de modo que se eligen las acciones con mayor valor en cada paso para formar parte de la política [32].

- **Q-Learning**: Este es un algoritmo que se basa en la existencia de una tabla donde se almacena la recompensa esperada para determinar acción en determinado estado. En otras palabras, el objetivo de este modelo es que por medio de las simulaciones ir completando la tabla de políticas de tal forma que las decisiones que se tomen en el camino obtengan las mayores recompensas hasta llegar al objetivo final. A continuación, se lista los elementos y funcionen que se emplean para este modelo [32].

    - La política se llamará Q: es una tabla que indicará el valor de la política para un estado y una acción determinada [Q(estado, acción)].
    - Acciones: los distintas acciones que puede hacer el agente en cada estado.
    - Recompensas: es el valor o puntaje que se asigna después de cada acción tomada, este puede sumar o restar.
    - Comportamiento: el *greedy* del agente indica si este se deja llevar por grandes recompensas de forma inmediata o explora y valora los puntajes a largo plazo.

  Para completar la tabla de valores Q en este modelo se emplea la ecuación (1.1), la cual corresponde a la ecuación de Bellman, la cual se basa en ir actualizando la tabla de valores Q teniendo en cuenta el valor actual más un recompensa futura en caso de tomar determinada acción, esta ecuación cuenta con dos tasas, las cuales influyen en la recompensa, estos son: la tasa de aprendizaje, la cual regula la velocidad en la que se aprende y la tasa de descuento, que tiene en cuenta la recompensa a corto y largo plazo [32].

$$New\ Q(S,A) = Q(S,A) + \alpha\,[R(S,A) + \gamma MAXQ'(S',A') - Q(S,A)] \qquad (1.1)$$

Donde:

- (S,A) representa el estado actual y la acción
- (S', A') representa el estado previo y la acción
- R es la recompensa
- $\boldsymbol{\alpha}$ es la tasa de aprendizaje
- $\boldsymbol{\gamma}$ es la tasa de descuento
- Max Q(S', A') es la máxima recompensa futura

Dada la naturaleza del aprendizaje por refuerzo, este es el más indicado para aplicar mecanismos de control tráfico; puesto que con este, se tendrá una actualización constate del estado de la red y los escenarios serán más cercanos a la realidad.

1.6. Herramientas

En esa sección se describirán las herramientas que se usaran para el desarrollo del actual trabajo de grado, adicional a ello se presentaran las características de cada una de ellas así como los recursos necesario para proceder con la instalación y configuración.

1.6.1. ONOS

Figura 1-14 Logo ONOS [36]

Su logo se observa en la Figura 1-14, este es un sistema operativo de red abierta diseñado en lenguaje Java por Open Networking Lab y hospedado por Linux Foundation en septiembre del año 2014 con el objetivo de ayudar a desarrolladores a tener flexibilidad a la hora de crear nuevos servicios y solventar problemas a nivel de operador. El controlador ONOS permite tener el control de los componentes de la red y los dispositivos para poder comunicar a los hosts con otras redes por medio de múltiples módulos manejados por bundles OSGi [33] – [36].

Los creadores del controlador se concentraron en que este tuviera características como la modularidad de código lo que hace posible introducir nuevas funcionalidades, otros aspecto es la configurabilidad que le da la capacidad de cargar y descargar las diferentes características antes y durante su ejecución; adicional a esto, también tiene la capacidad de separación de conceptos lo que facilita la modularidad y finalmente ONOS no tienen librerías ligadas a protocolos específicos o implementaciones [33] – [36].

ONOS maneja una vista de todos los nodos, lo que hace posible soportar las fallas sin alterar el funcionamiento de la red; este modelo se caracteriza por ser líder en estrategia y procesamiento. La primera vez que se lanzó este controlador tuvo por nombre Avocet, posteriormente se llamó Blackbird; este controlar se maneja por medio de comandos y también cuenta con una interfaz web donde se pueden administrar todas sus aplicaciones y visualizar todas las herramientas disponibles [33] – [36].

La arquitectura de ONOS mostrada en la Figura 1-15 está compuesta por un conjunto de componentes que son los que definen la unidad de funcionamiento dentro del paradigma ONOS; este modelo está dividido en las capas de aplicaciones, la interfaz hacia el norte, la interfaz hacia el sur, el *core*, protocolos, elementos de red y proveedores [33] – [36].

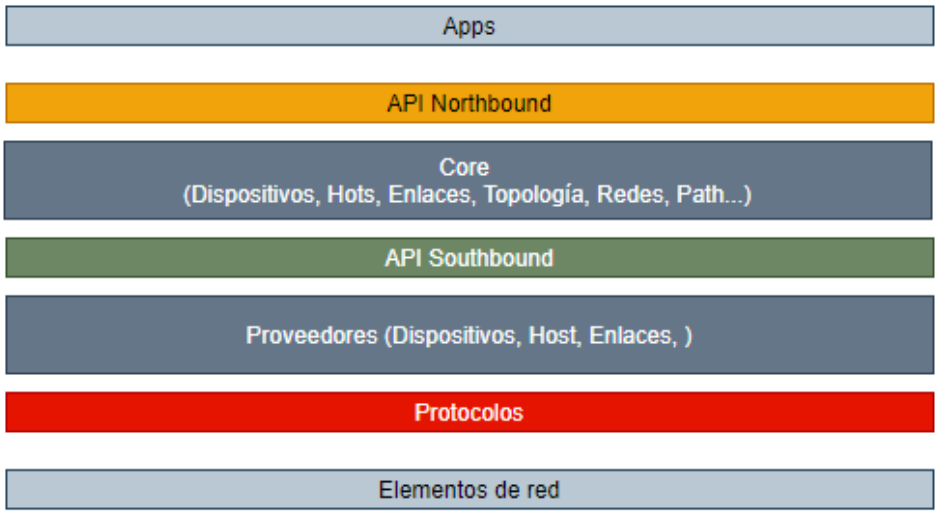

Figura 1-15 Arquitectura de ONOS [33, 36]

En cuanto al funcionamiento de la herramienta ONOS, esta es compatible con OpenFlow para manejar la interfaz en dirección sur, para el cálculo de ruta de los elementos se usa PCEP, para la configuración se utiliza NETCONF y para comunicarse con la interfaz de aplicación norte se hace uso de aplicaciones de transferencia de estado API REST [33] – [36].

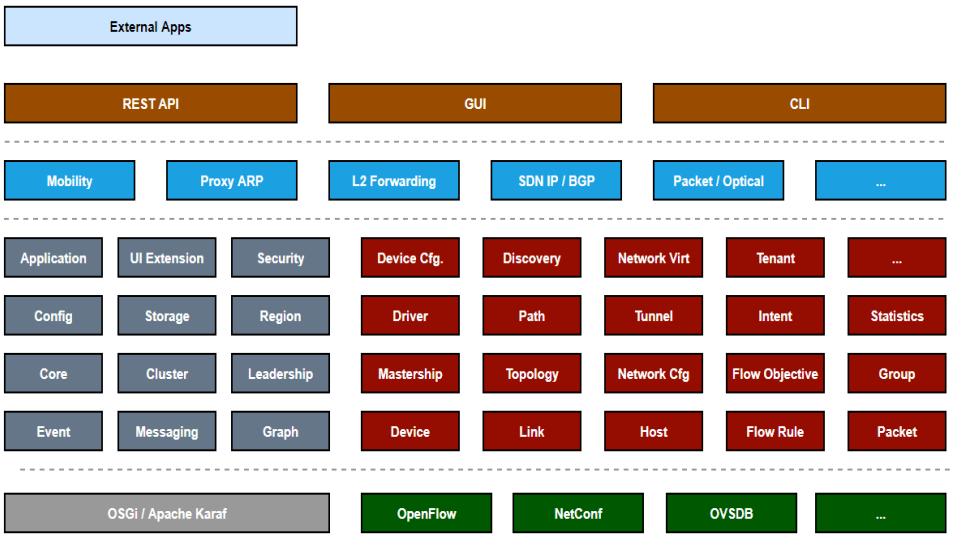

Figura 1-16 Subsistemas de ONOS [33, 36]

Los Subsistemas mostrados en la Figura 1-16 se encapsulan en algunas de las tres capas principales de ONOS, y se identifican por medio de una o más interfaces java que se emplean para descubrir la topología y calcular la ruta, luego el núcleo o Core

transporta esa información a la capa de aplicaciones para tomar las decisiones correspondientes, posterior a ello el protocolo OpenFlow comunica la ruta seleccionada a los conmutadores y dispositivos de la red, mientras que la interfaz hacia el sur se encarga de descubrir los dispositivos que van agregando al sistema, después la red transmite las rutas actualizadas a las capas superiores y de esa manera es que este controlador manipula la topología y calculas las rutas de la red de manera dinámica [33] – [36].

1.6.2. MININET

Es una herramienta de emulación de red que permite crear distintas topologías de red e interactuar con ellas haciendo de hosts; este tipo de redes es totalmente escalable ya que utiliza las funcionalidades de la virtualización para el uso de una gran cantidad de nodos lo que facilita crear, personalizar y compartir de manera rápida las SDN; este software es desarrollado en lenguaje Python e implementado sobre el kernel de LINUX y es compatible con múltiples controladores [37].

Sus características son:

- Permite al usuario trabajar de manera simultánea e independiente en la misma topología a múltiples desarrolladores.
- Soporta prueba de regresión que pueden ser repetidas y empaquetadas a nivel de sistema.
- Tiene una interfaz de aplicación desarrollada en lenguaje Python que es sencilla de manejar y es amigable con el usuario a la hora de crear redes y experimentar con ellas.
- Tiene un conjunto de topologías preestablecidos y soporta la creación de topologías personalizadas.
- Esta herramienta es realista, es decir, el comportamiento del esquema desarrollado debe coincidir altamente con el diseño real.

Algunas de las ventajas que presenta este entorno de emulación con respecto a los demás que existen es que este es más rápido al arrancar, es más escalable, ofrece mayor ancho de banda y es de fácil instalación. Sin embargo, la mayor limitación que presenta esta herramienta es la perdida de fidelidad ante cargas grandes, ya que los

dispositivos no transmiten a la misma velocidad; adicional a eso, actualmente el emulador solo trabaja con los enlaces cableados y todos los hosts comparten el mismo sistema de archivos [37].

Esta herramienta ofrece la posibilidad de crear distintas topologías por medio de comandos en su ventana CLI. Dentro de estas topologías se tiene la única que es la más simple y se crea usando el comando $ $\boldsymbol{sudo\ mn - topo\ single,n}$ donde n representa el número de host que se conectan a un único switch; el segundo tipo de topologías estandarizadas es la lineal donde un switch tiene asignado un único host; el comando usado para este diseño es $ $\boldsymbol{sudo\ mn - -topo\ linear,n}$ donde en este caso n representará al número de switches. La topología en árbol también es comúnmente utilizada en redes, el diseño de este tipo de topología está basado en n niveles de switch y m host conectados a cada uno de los conmutadores; el comando para este modelo es $ $\boldsymbol{sudo\ mn - -tipo\ tree, depht = n, fanout = m}$. Finalmente, mininet permite crear topologías que se ajusten a las necesidades y requerimientos de los usuarios, estas son las llamadas topologías personalizadas y para desarrollarla se debe hacer unos de scripts de python [37].

Otra forma de crear topologías de red en mininet es por medio de su interfaz gráfica llamada *MiniEdit*, la cual es una herramienta amigable con el usuario y capaz de crear y ejecutar distintas simulaciones de red, dado que permite configurar los elementos de la red de una forma más sencilla y si se requiere su script se puede exportar. Esta interfaz cuenta con un área de trabajo la cual se observa en la Figura B-0-6, aquí se encuentran los distintos elementos de red como los *Host* y *Stations* que realizan la funciones de equipos terminales, los *switch* cuya función es la de un conmutador y el cual debe estar conectado a un controlador; otros elementos que se encuetan en esta interfaz son los Legacy switch y Legacy router que hacen las veces de conmutador y enrutador con la particularidad de que estos no necesitan estar conectados a un controlador, el *controller*, como su nombre lo indica es el controlador de la red. Todos estos elementos se pueden configurar con las necesidades que requiera la red y se pueden agregan tantos como sea necesarios [37] – [39].

El alcance de simulación de mininet es bueno, sin embargo hacía falta integrar el mundo inalámbrico, es por eso por lo que los desarrolladores crearon una extensión que le permitirá a los usuarios agregar estaciones Wi-Fi virtualizadas así como puntos

de acceso basados en controladores inalámbricos; a esta extensión de la herramienta se le dio el nombre de mininet-wifi y dado que el desarrollo solo consistió en agregar clases y ampliar el código base, mininet-wifi aun admite todas las funcionalidades normales del mininet estándar, incluyendo las limitaciones que este tiene [37] – [39].

El desarrollo de este capítulo deja en evidencia la magnitud que tiene el Internet de las Cosas, así como los grandes beneficios que tiene el incorporar las SDN. Esta búsqueda documental permitió limitar el entorno de aplicación de este trabajo de grado, el cual es un entorno doméstico, dado que no solo es el más común de todos los ecosistemas de IoT en el mundo real sino también es el que más dificultades puede presentar en cuanto a la congestión debido a las capacidades y costos del servicio. Una vez estipulado el entorno de trabajo, se observa a detalle cada una de las propuestas de arquitectura IoT encontradas, y se establece desde ya que la que mejor se acopla a los objetivos de esta investigación es el modelo de tres niveles pues permite una adaptación menos compleja con las redes definidas por software y resalta el nivel de datos que es de interés para este trabajo. Adicionalmente se buscaron las herramientas que mejor se ajustaran al desarrollo del trabajo y se definió que para emular la red se manejará mininet-wifi y después de investigar los distintos controladores vigentes en el mercado, comparar las características de cada uno y la usabilidad en otras propuestas de investigación, se optó por usar el controlador ONOS, pues no solo es uno de los software más completos y actualizado hasta el momento, sino también es un controlador propuesto desde el planteamiento de la propuesta del actual trabajo de grado, con el fin de resaltar las cualidad y beneficios que tiene el usar este software en la aplicación de soluciones SDN.

2. METODOLOGÍAS

En el capítulo 2 se presentarán las metodologías que se utilizaron para el desarrollo del actual trabajo de grado; la metodología usada a lo largo del trabajo y la cual se definió en el anteproyecto, y la metodología para realizar la simulación del proyecto; finalizando el capítulo se plantearan algunos de los requisitos necesarios que debe tener el sistema para alcanzar los objetivos propuestos.

2.1. Metodología de Trabajo

Para el desarrollo de este trabajo de grado se implementó la metodología en cascada, de tal manera que se pudiera estructurar de forma lógica secuencial desde el inicio hasta el punto final, haciendo uso de varias capas donde cada una de ellas depende directamente de la culminación de la anterior. Algunos modelos de esta metodología contienen seis niveles y otros tienen cinco dependiendo del autor, para la implementación de este documento se usará una estructura de cinco niveles como se ve en la Figura 2-1 [40].

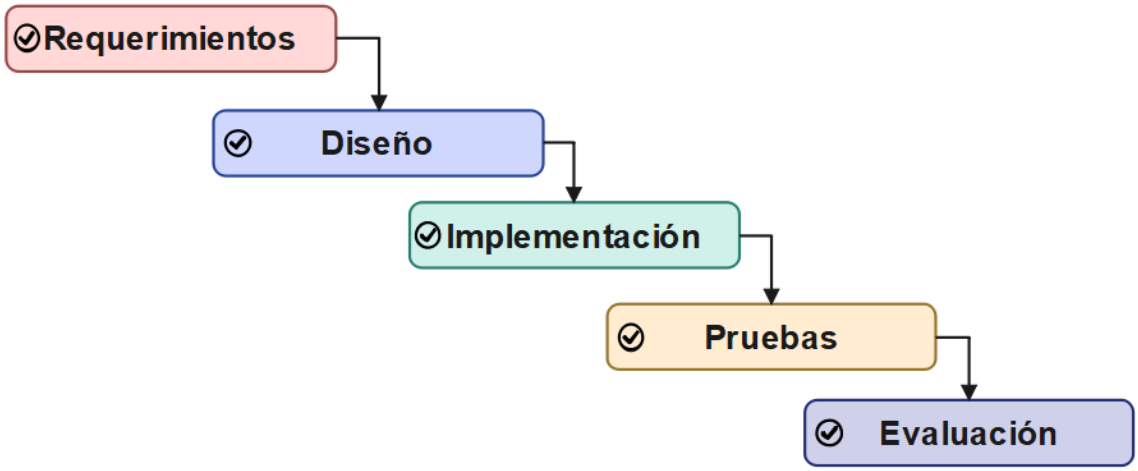

Figura 2-1 Modelo de la metodología en cascada [Elaboración Propia]

El modelo planteado y el cual fue una adaptación del presentado en [40] comienza por el análisis de requerimientos, donde se analizarán los conceptos teóricos y se realizará

una investigación y recopilación de la información necesaria para así determinar el alcance y establecer requisitos necesarios para alcanzar los objetivos.

Una vez se termina toda la investigación se entra a la etapa de diseño, donde se tomará todos los parámetros establecidos en la fase anterior para adaptar y diseñar una topología de red, un mecanismo de control de tráfico basado en algoritmos machine learning y la aplicación base para el correcto funcionamiento del controlador [40].

Al terminar la etapa de diseño se ingresa a la fase de implementación donde se realizará la instalación y adecuación de las herramientas establecidas en la fase de requerimientos, así como la incorporación del algoritmo de control de tráfico basado en ML y la topología de la red a la herramienta de simulación [40].

Después de terminar la implementación y adecuación de todo el sistema se ingresará a la fase de pruebas, aquí se definirán los escenarios y el plan de pruebas, donde se mirará el desempeño de cada elemento por separado y de todo el conjunto integrado para así determinar el funcionamiento óptimo del sistema desarrollado [40].

Finalmente, después de superadas todas las etapas anteriores, se ingresará a la última etapa llamada evaluación, aquí se hará un análisis del desempeño de la red, teniendo en cuenta los parámetros definidos anteriormente, se elaborarán las conclusiones y se plantearán los posibles cambios o mejoras a futuro que puedan realizarse al actual trabajo de grado [40].

Las ventajas que trae consigo esta metodología es que al tener que dar cumplimiento en orden a cada una de las etapas ayuda mitigar los posibles errores que se pudiesen presentar; se debe aclarar que ninguna metodología es mejor que otra, solo es cuestión de ver cual se adapta mejor a las necesidades del proyecto [40].

2.2. Metodología de Simulación

Para dar respuesta a la pregunta de investigación y dar cumplimientos a las metas planteadas en los objetivos del actual trabajo de grado, fue necesario implementar una metodologia de simulación. En ese orden de ideas y dado que es un grupo reducido de trabajo, se optó por utilizar una metodología ágil; de todas las propuestas que existen en la literatura se eligio *SCRUM*, puesto que es la que mejor se adapta al desarrollo de este trabajo; dado que permite hacer entregas regulares del producto final, al ser una metodología incremental facilita que el desarrollo se vaya realizando de a poco sin importar el orden, lo que indica que el trabajo se puede desarrollar por pequeñas partes y al final unificar todo y lo más importante es que permite regresar a una fase anterior y realizar cambios durante el proceso. A continuación, se describen los procesos o fases que conforman esta metodología que estan descritos en [38] [39] y por lo que se debe pasar para culminar con éxito el trabajo.

Fase 1, Planificación: tambien conocido como *Product Backlog*, es aquí donde se establecen las tareas prioritarias y donde se obtiene toda la información del proyecto que se requiere para desarrollar. En el caso de este trabajo, es aquí donde se identifica el problema a resolver, las condiciones, delimitaciones y ajustes del sistema, también se identifican los parámetros empelados para medir el desempeño de la red y observar el comportamiento del sistema en determinados escenarios [41] [42].

Fase 2, Ejecución: esta fase también conocida como *Sprint,* es la más importante dado que es aquí es donde se produce el desarrollo del producto entregable y tiene un timepo límite de entrega. Para ser más especificos, es aquí donde se inicia con el desarrollo del modelo de simulación y red, que permitirán realizar la recoleción de los datos necesarios para el análisis; también se implementará el desarrollo de las aplicaciones en los software de emulación con las características pertinentes tanto para la topología como para el control de tráfico [41] [42].

Fase 3, Control: también llamada como *Burn Down,* permite medir el progreso del proyecto. De manera puntual, aquí se harán las validaciones del modelo de desempeño basado en las métricas definidas con anterioridad; al igual se realizará la experimentación, el análisis y la presentación, es decir se harán las simulaciones, se

obtendrán los resultados y se presentarán por medio de gráficas o tablas respaldándolos con la teoría para tener un alto nivel de confiabilidad [41] [42].

Después de revisar el concepto de la metodología, se establece una adaptación de las fases mencionadas anteriormente y se elabora el diagrama de flujo que se observa en la Figura 2-2, el cual es el modelo a seguir para el diseño y la implementación tanto de la red, el algoritmo ML de control de tráfico y la aplicación del controlador para el mejoramiento del desempeño de la red [41] [42].

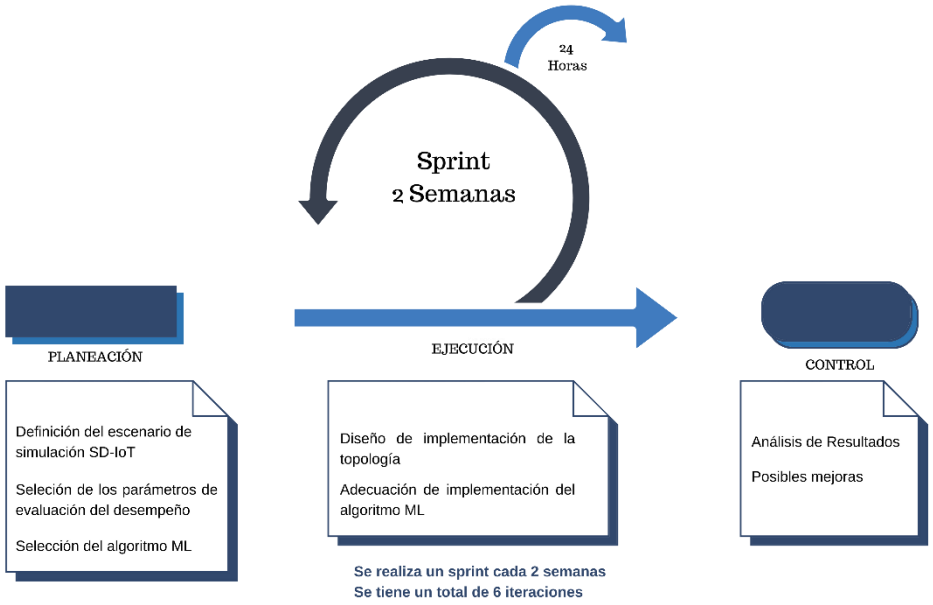

Figura 2-2 Diagrama de flujo de la metodología de simulación [Elaboración Propia]

2.3. Requerimientos

Teniendo en cuenta todo el componente teórico estudiado en el capítulo 1 y acorde con la fase 1 de la metodología de investigación, a continuación, se plantean los requerimientos necesarios que se deben tener en cuenta para el diseño de la red, el correcto funcionamiento de las herramientas, y algoritmo ML.

Después de considerar todos los posibles escenarios que tiene IoT en el mundo real, y analizar las falencias que se puedan presentar en cada uno de ellos, se encontró que el ecosistema donde llegan a presentan más inconvenientes a nivel de congestión o perdida de paquetes es el entorno doméstico, esto puede darse por la capacidad que se maneja en este tipo de redes y a que los dispositivos IoT deben compartir los recursos con otro tipo de dispositivos que no son IoT; teniendo en cuenta lo anterior se plantea una lista de requerimientos que deben cumplir a la hora de realizar el diseño dicha red.

- Se debe contar tanto con dispositivos IoT como no IoT
- La conexión de los dispositivos debe ser de forma inalámbrica
- Configuración de parámetros Wi-Fi
- Recopilación de las estadísticas de la información enviada a través de los dispositivos
- Conexión estable

Al revisar la información encontrada sobre los entornos de aplicación de las diferentes técnicas de machine learning, se encontró que las dos ramas de ML que se pueden adaptar para el control de tráfico en redes son el supervisado y el reforzado; una vez analizadas los distintos mecanismos y considerando lo expuesto en [21], se establece que la mejor técnica para realizar la tarea de control de tráfico son las basada en el aprendizaje reforzado, ahora, para lograr el buen funcionamiento de este técnica es necesario que se cumpla lo siguiente:

- Identificar el tipo de dispositivo
- Identificar todos los posibles estados
- Establecer las acciones a seguir
- Identificar los parámetros o estadísticas de los puertos de los puntos de acceso
- Establecer un rango de recompensas considerando los parámetros de desempeño de la red

En cuanto a la selección del controlador para la administración de la red, después de revisar las distintas opciones que se tiene en el mercado y las cuales se plantean en la sección 1.2.3. Elementos que Integran una SDN del capítulo anterior, se encontró que el que mejor se adapta al desarrollo del trabajo por la capacidad que tiene para agregar nuevas funcionalidades dentro del controlador, que versiones maneja del protocolo *openflow*, la compatibilidad con los distintos sistemas operativos y actualización es el controlador ONOS; para poder realizar la integración del algoritmo ML es necesario que ONOS cumpla con lo siguiente:

- Conexión con la red diseñada
- Incorporación de aplicaciones propias
- Compatibilidad con la red
- Inserción de nuevas rutas según el resultado del algoritmo

Una vez establecidos los requerimientos mínimos de la red, el algoritmo y el controlador, se da por terminada la fase 1 de la metodología y se puede entrar a la segunda fase que es el diseño.

2.4. Diseño

A partir de esta fase se comenzó con la implementación de la metodología de simulación, comenzando por la fase de planeación, de esta manera se estará realizando una retroalimentación constante del proceso para al final tener un buen sistema.

Inicialmente se hicieron los primeros diseños de la arquitectura, la red y mecanismos ML, y después de realizar las diferentes retroalimentaciones mostradas en el ANEXO I, se llegó finalmente a los modelos que se muestran continuación y los cuales se ajustan a los requerimientos mencionados previamente.

2.4.1. Red SDN-IoT

A partir de la documentación presentada en el apartado 0, donde se establece el modelo de arquitectura IoT, y teniendo en cuenta que el objetivo del proyecto es mejorar la experiencia del usuarios al integrar el Internet de las Cosas con el concepto de SDN; el esquema de la arquitectura SDN-IoT, se observa en la Figura 2-4, donde la capa más importante será la capa de datos, la cual se separa para agregar la entidad de control y a partir de esto, los dispositivos estarán conectados a la red mediante una *Gateway* permitiendo tener flexibilidad a la hora gestionar los datos para mantener la calidad de servicio entre otros; otra ventaja de implementar una puerta de enlace, es que se garantiza el manejo de las grandes cantidades de tráfico proveniente de todos los dispositivos conectados.

Considerando que para el desarrollo de este trabajo de investigación se trabajará con un sistema totalmente emulado, fue necesario realizar una adaptación de la arquitectura SDN-IoT para que se adecuara a las condiciones del desarrollo del trabajo actual; en la Figura 2-3 se puede observar el esquema resultante donde se tiene inicialmente el plano de datos donde se centraran los dispositivos OpenFlow emulados en la respectiva herramienta, en el segundo nivel se tiene el plano de control donde se encontrara ONOS que es el controlador seleccionado para ser el administrador de la red, y finalmente en la capa de aplicaciones.

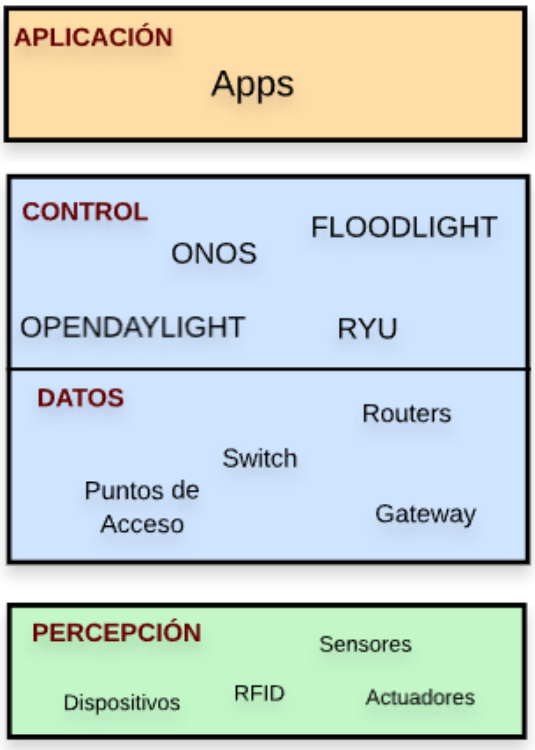

Figura 2-4 Arquitectura SDN-IoT [Elaboración propia]

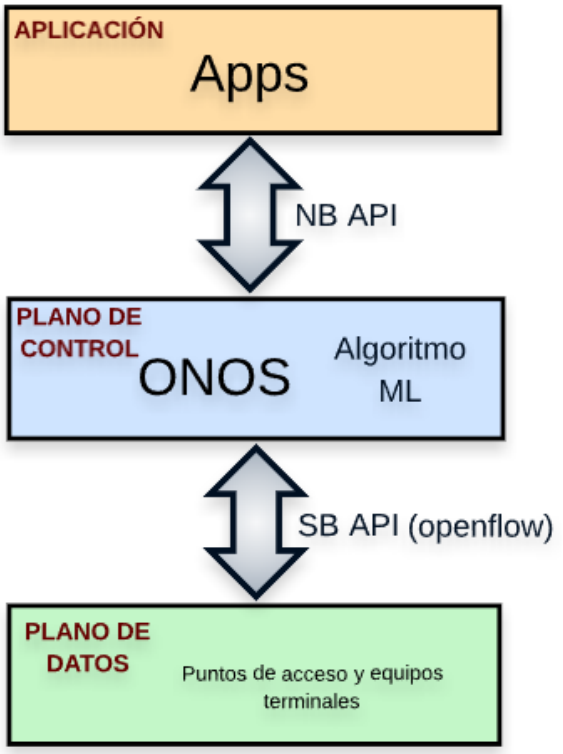

Figura 2-3 Adecuación de la arquitectura SDN-IoT [Elaboración Propia]

Considerando el escenario de motivación planteado y teniendo en cuenta los requerimientos establecidos previamente; inicialmente se consideró emular un escenario mediano, para ello se establecieron 24 equipos terminales que harán las veces de dispositivos IoT y no IoT, los cuales se conectan a diferentes puntos de acceso para acceder a la red, a su vez estos puntos de acceso están enlazados con el controlado quien será quien los administre. La cantidad de dispositivos fue elegida con el fin de generar una saturación en la red y de esta manera poder evaluar el mecanismo de control de tráfico.

En la Figura 2-5 se observa el esquema de la topología de red con la distribución de dispositivos por punto de acceso y la conexión establecida entre los APs. En este modelo, los equipos terminales harán las veces de dispositivos IoT y no IoT; por tanto, serán los que realicen las solicitudes de petición de información a la red, y quienes envíen la información a través de esta; todos estos requerimientos de información pasaran a través del controlador y será quien decida qué tipo de información tiene

mayor prioridad y por dónde se envía; de ese modo estos dispositivos no tienen conocimiento del estado de la red, ni de su topología.

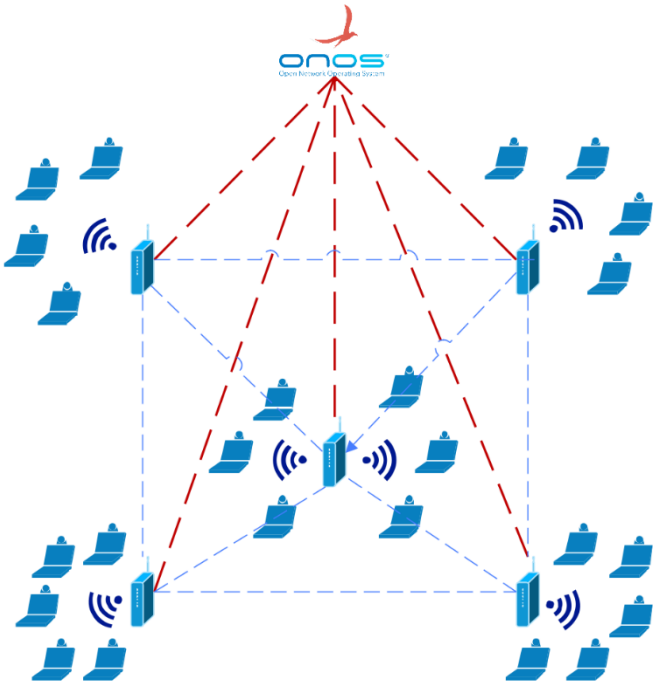

Figura 2-5 Diseño de la Topología de Red [Elaboración Propia]

2.4.2. Algoritmo ML

Como se estableció previamente en el apartado 1.5. *Machine Learning* el algoritmo que se implementará dentro del controlador es el Q-Learning; este modelo establece un conjunto de rutas menos congestionada desde el nodo de origen hasta nodo de destino. Dado que el controlador ONOS está desarrollado en *java*, el algoritmo estará implementado en este mismo lenguaje; un ejemplo de desarrollo en java del algoritmo seleccionado se encuentra en [43] – [45]; por tanto, se tomará como referencia el código encontrado para realizar una adaptación y elaborar un algoritmo que se ajuste a las necesidades del objetivo de este proyecto.

Para empezar con los ajustes necesarios, primero se establece un diagrama de flujo, el cual se muestra en la Figura 2-6; inicialmente, el algoritmo identifica el dispositivo de mayor prioridad y a que *access point* está conectado, una vez identificado procede

a establecer los saltos de las posibles rutas hasta llegar a nodo destino, y estos serán conocidos como los estados del algoritmo; posterior a ello, las acciones serán comprendidas como los posibles caminos que puede tomar los paquetes para llegar a su destino, que estas dos características establecidas se elabora e inicializa la tabla de valores-Q; el siguiente paso del mecanismo es seleccionar una acción para realizar y luego el sistema le asigna una recompensa, la cual es estipulada teniendo en cuenta la pérdida de paquetes, el retardo y cantidad de bytes que se transmiten por cada puerto del dispositivo; de esta manera el sistema sigue realizando hasta completar un episodio; finalmente, el algoritmo entrega un camino, el cual es el menos congestionado y esta es la información que maneja el controlador para realizar el envío de información al nodo respectivo.

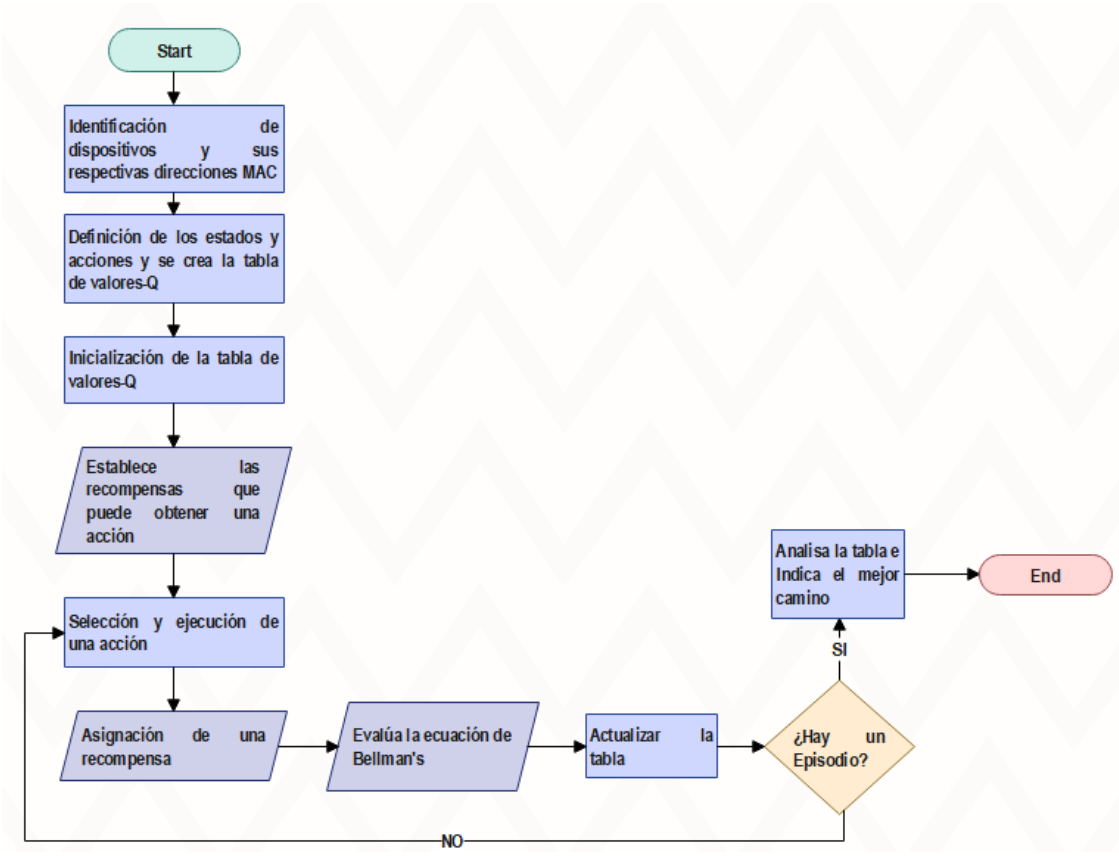

Figura 2-6 Diagrama de Flujo del algoritmo [Elaboración propia]

Una vez establecido el diagrama de flujo, a continuación, se realiza el pseudocódigo del algoritmo, el cual es el segundo paso por seguir antes de la implementación en el lenguaje java.

Inicio

Identificar dispositivo
Definir Alpha <= 0.1
Definir gamma <= 0.9
Definir acciones
Inicializar Q(s,a)

para (cada paso en el episodio actual)
mientras (el estado sea diferente al destino)
Elige una A de Q
Ejecutar la Acción
Obtener recompensa
*Q(s,a) <= Q(s,a) + alpha * (R(s,a) + gamma * Max(next state, all actions) - Q(s,a))*
Estado <= próximo estado
Fin mientras
Fin para
Se actualiza la tabla de valores Q
Se imprime el resultado de las posibles rutas

Fin

2.4.3. Parámetros de Desempeño

Para realizar la evaluación de la red SDN-IoT se implementarán dos métricas que permitan medir el rendimiento de esta en términos de tiempo, congestión y respuesta ante fallos. Ahora bien, el desempeño de la red se medirá en función del retardo de extremo a extremo y la pérdida de paquetes.

- **Retardo de extremo a extremo**

Para determinar este parámetro de beben considerar los retardos de procesamiento, de transmisión, de propagación y buffers, en la ecuación (2.1) se observa esta relación.

$$R_{total} = R_{procesamiento} + R_{trasmision} + R_{propagacion} + R_{buffer} \quad (2.1)$$

Donde el retardo de procesamiento indica el tiempo que requiere el sistema para procesar los datos y decide a donde enviar el paquete; el retardo de buffer es el tiempo que debe esperar un paquete hasta que se transmite, depende de la intensidad y tipo de tráfico; el retardo de transmisión se entiende como el tiempo que tarda el emisor en poner los datos en el medio de transmisión y está ligado a la velocidad de trasmisión; finalmente el retardo de propagación es el tiempo que toma el paquete para viajar desde el origen hasta el final.

- **Pérdida de Paquetes**

Este parámetro se define como los datos que se pierden en la red y de todas las causas, la que se tendrá en cuenta es la congestión, puesto que la información viaja a través de múltiples dispositivos y enlaces y en algún punto del proceso se puede tener un enlace con su capacidad total en uso, provocando que la información se afecte por retrasos tal punto que esta sea descartada después de un determinado tiempo. En las redes inalámbricas es normal que exista una pérdida de paquetes puesto que en ocasiones estas se ven afectadas por situaciones que no se pueden controlar como tener que atravesar obstáculos.

2.4.4. Casos de estudio

Para evaluar de una forma más apropiada el sistema y así dar una respuesta adecuada al objetivo planteado del actual trabajo de grado, se definen dos grandes escenarios, en el primero se tendrán todos los puntos de acceso conectados son sus respectivos enlaces activos, el segundo escenario se dan de baja a dos de los enlaces que conectan los puntos de acceso; adicional a esto, se establecen los siguientes casos de estudio que estarán ligados a los escenarios propuestos anteriormente.

- **Caso 1:** está ligado al escenario 1 y en este caso se observará el comportamiento de la red sin el uso de la aplicación de control, para ello se enviará tráfico mediano constante durante un tiempo determinado.

- **Caso 2:** se tienen en cuenta las mismas condiciones del caso anterior, pero en este caso se activa el mecanismo de control diseñado en el controlador.

- **Caso 3:** está ligado al escenario 2, en este caso se da de baja a dos de los enlaces que conectan los puntos de acceso y se observará el comportamiento con la misma carga de tráfico que en el escenario 1.

- **Caso 4**: se consideran las mismas condiciones del caso y se activará la aplicación en el controlador.

Una vez completadas las fases de diseño y planeación, se procede a iniciar las fases de implementación y ejecución de las respectivas metodologías; estas fases estarán expuestas en el siguiente capítulo.

3. IMPLEMENTACIÓN

3.1. Herramientas

Para empezar con la implementación y ejecución de la red y algoritmo de control de tráfico, primero es necesario instalar y configurar las herramientas sobre las cuales se va a desarrollar la simulación, una vez completa la instalación, se procede a realizar la implementación de la red y el algoritmo.

3.1.1. Emulador de redes SDN

Dado que el presente trabajo de grado se centra en el manejo de SDN y específicamente redes inalámbricas; por tanto, es necesario utilizar una herramienta que permita tener ambos; razón por la cual se decide utilizar *mininet-wifi*, la cual como se explicó en el apartado 1.6.2. MININET permite emular elementos *openflow* de las redes definida por software y adicional a ello extiende las funcionalidades normales de mininet, facilitando la emulación de elementos inalámbricos con tecnología Wi-Fi. La instalación de esta herramienta se puede apreciar con detalle en el ANEXO B

3.1.2. Controlador ONOS

Para realizar la instalación y configuración del controlador ONOS existen tres maneras de hacerlo, se puede hacer por medio de Dockers o contenedores, también se puede realizar por medio de la clonación del código fuente del núcleo del controlador y finalmente por medio de la descarga del archivo *tar.gz* de la página. Para el desarrollo de este trabajo de grado se optó por la última forma de instalación, la cual se encuentra explicada detalladamente en el ANEXO C se debe tener en cuenta que esta herramienta requiere de altos recursos hardware para su óptimo desempeño, en la Tabla 3-1 se muestra los recursos mínimos necesarios para su funcionamiento.

Tabla 3-1 Recursos de MV

Recurso	Mínimo	Asignado
Memoria	8 GB	30 GB
procesadores	4	24
Disco Duro	**30 GB**	80 GB

3.1.3. Generador de Tráfico

Debido a que se busca que el escenario simulado sea lo más cercano a la realidad, es necesario que las pruebas realizadas también sean lo más realistas posibles; para ello se busca una generador de tráfico que permita configurar los parámetros del tráfico que se van a enviar a través de la red, la herramienta seleccionada para cumplir este requisito es el generador de tráfico D-ITG, la instalación y comandos de configuración de esta herramienta se observa en la parte final del ANEXO B.

3.1.4. Configuración de herramientas adicionales

Es necesario instalar otras herramientas para el trabajo, debido a que se debe desarrollar aplicaciones nuevas en el controlador, y adicional a ello configurar la topología de red con la que se va a trabajar; por tanto, se va a trabajar en dos lenguajes de programación diferentes que son java para las aplicaciones y python para la topología red; por tanto, se requiere de un editor de códigos para trabajar de forma más dinámica y ágil; en ese orden de ideas se utilizara *atom* para la topología de red e *Intellij IDEA* para las aplicaciones; en el ANEXO D se encontrará la forma de instalación de estas dos herramientas en el sistema operativo Ubuntu.

3.2. Red SDN-IoT

Una vez instaladas y configuradas todas las herramientas, se procede a realizar la implementación de la SDN-IoT en el entorno de simulación Mininet-wifi y conectar esta red al controlador ONOS para así evaluar la red con y sin el algoritmo de control de tráfico de tal forma que se pueda medir el desempeño en función del retardo de extremo a extremo y la pérdida de paquetes.

El despliegue y emulación de la red en la herramienta mininet se hará por medio de un script en python; inicialmente se definen las características, haciendo uso de librerías y funciones, en la Figura 3-1 se muestra el código empleado para este fin.

```
def myNetwork():
    net = Mininet_wifi(topo=None,
                       build=False,
                       link=wmediumd,
                       wmediumd_mode=interference)
```

Figura 3-1 Configuración de parámetros iniciales

El siguiente paso, será configurar los parámetros del controlador para poder establecer un conexión entre este y la red; en la Figura 3-2 se observa las líneas de código utilizadas para este fin en el *script.*

```
info( '*** Adding controller\n' )
c0 = net.addController(name='onos',
                       controller=RemoteController,
                       ip='192.168.223.146',
                       protocol='tcp',
                       port=6653)
```

Figura 3-2 Configuración de parámetros del controlador

Un vez establecida la conexión del controlador con la red se procede a especificar los tipos de dispositivos que se van a utilizar y sus respectivas configuraciones. Como se mencionó en la fase de diseño se trabajarán con puntos de acceso y los equipos terminales serán inalámbricos; por tanto, estos serán emulados con estaciones. En la Figura 3-3 se muestran los comando empleados para añadir los puntos de acceso a la red; de igual manera en la Figura 3-4 se observa las líneas de código con los parámetros pertinentes para la configuración de los equipos terminales o estaciones.

```
info( '*** Add switches/APs\n')
ap1 = net.addAccessPoint('ap1', cls=UserAP, ssid='ap1-ssid', channel='1',
    mode='g', position='168.0,199.0,0', datapath='user')
ap2 = net.addAccessPoint('ap2', cls=UserAP, ssid='ap2-ssid', channel='1',
    mode='g', position='557.0,198.0,0', datapath='user')
```

Figura 3-3 Configuración de parámetros de los puntos de acceso

```
info( '*** Add hosts/stations\n')
sta3 = net.addStation('sta3', ip='10.1.0.103/24', position='507.0,317.0,0')
sta1 = net.addStation('sta1', ip='10.1.0.101/24', position='94.0,317.0,0')
sta2 = net.addStation('sta2', ip='10.1.0.102/24', position='223.0,319.0,0')
sta4 = net.addStation('sta4', ip='10.1.0.104/24', position='642.0,313.0,0')
```

Figura 3-4 Configuración de parámetros de las terminales

Una vez añadidos los dispositivos necesario para implementar la red, el paso siguiente es realizar las conexiones entre los puntos de acceso y la conexión de las estaciones a los respectivos puntos; este proceso se realiza tal como se muestra en la Figura 3-5, donde se establece la capacidad del enlace y el número de puertos por donde se realiza la conexión entre los AP.

```
info( '*** Add links\n')
ap1ap2 = {'bw':100}
net.addLink(ap1, ap2, 2, 2, cls=TCLink , **ap1ap2)
ap1ap3 = {'bw':100}
net.addLink(ap1, ap3, 3, 2, cls=TCLink , **ap1ap3)
ap2ap3 = {'bw':100,}
net.addLink(ap2, ap3, 3, 3, cls=TCLink , **ap2ap3)
```

Figura 3-5 Configuración de enlaces entre APs

En cuando a la conexión de las estaciones base a los puntos de acceso se estableció una conexión inalámbrica para definir qué equipo terminal se conectará a determinado punto de acceso, en la Figura 3-6 se observa el comando empleado, donde se establece el tipo de enlace, la dirección MAC y el ID de punto de acceso.

```
cmd = "iw dev {} connect {} {}"
sta1.cmd(cmd.format("sta1-wlan0", "ap1-ssid", "02:00:00:00:18:00"))
sta5.cmd(cmd.format("sta5-wlan0", "ap2-ssid", "02:00:00:00:19:00"))
sta10.cmd(cmd.format("sta10-wlan0", "ap3-ssid", "02:00:00:00:1a:00"))
sta15.cmd(cmd.format("sta15-wlan0", "ap4-ssid", "02:00:00:00:1b:00"))
sta20.cmd(cmd.format("sta20-wlan0", "ap5-ssid", "02:00:00:00:1c:00"))
```

Figura 3-6 Conexión de las sta a sus respectivos AP

Una vez definida la topología de la red, se debe iniciar y conectar todos los dispositivos al controlador; este proceso se muestra en la Figura 3-7.

```
net.get('ap3').start([onos])
net.get('ap5').start([onos])
net.get('ap4').start([onos])
net.get('ap1').start([onos])
net.get('ap2').start([onos])
```

Figura 3-7 Conexión del módulo APs al controlador

Las imágenes anteriores solo son una pequeña muestra de la configuración general de toda la red con la que se trabajará; un vez terminada toda la configuración e inserción de elementos necesario para conformar la red, para ver el código completo de la red ver el ANEXO H.

Después de tener todo listo se procede a realizar la compilación desde mininet usando el siguiente comando *sudo python redfinal.py –controller remote, ip=192.168.223.146;* si el código no tiene problemas, se compilará sin ningún inconveniente y se observará de manera gráfica como está distribuida la red, el resultado se observa en la Figura 3-8.

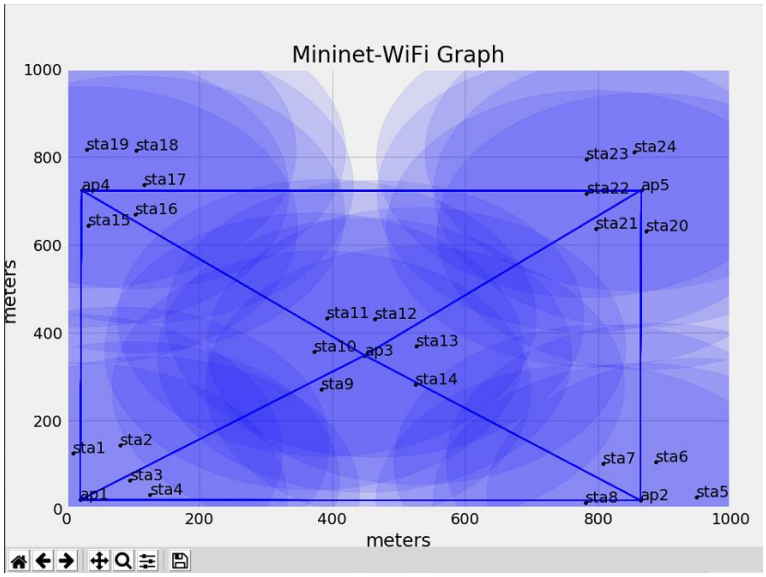

Figura 3-8 Gráfica de la Distribución de la red en mininet

Posterior a ello, se procede a comprobar la conexión entre la red y el controlador ONOS, para verificar esto se ingresa a la interfaz web del controlador y se tendrán que visualizar todos los AP como se observa y luego de hacer un *pingallfull* en el CLI de

mininet podrán verse los terminales conectados a cada punto de acceso como se ve en la Figura 3-9.

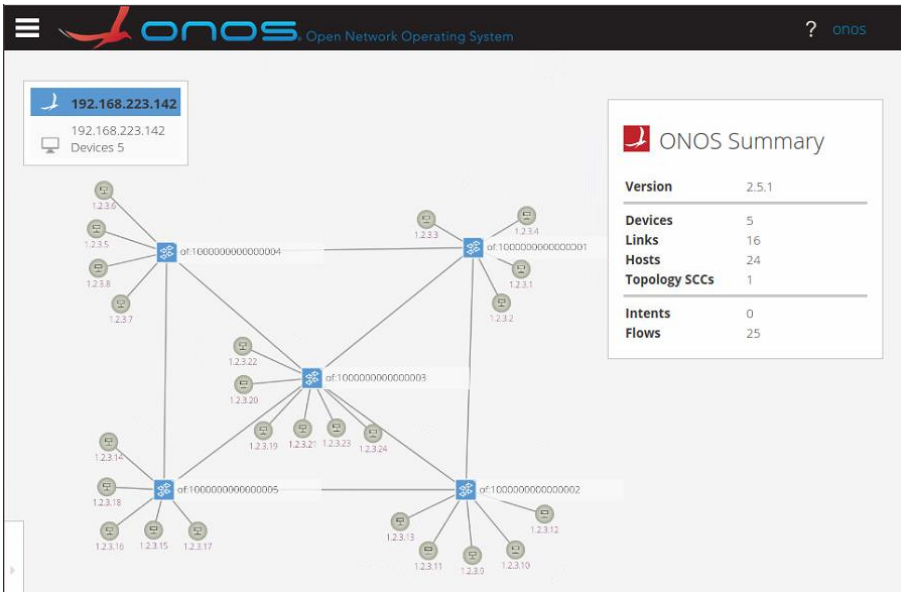

Figura 3-9 Topología de Red vista desde el Controlador

3.3. Tráfico

Para empezar a implementar el generador de tráfico, inicialmente se definen los roles de cada dispositivo, es decir cuáles de ellos hará las veces de receptores, cuales de emisores IoT y cuáles de ellos serán dispositivos no IoT; esta distribución se observa en la Tabla 3-2, de igual manera en la Tabla 3-3 se muestra la distribución de dispositivos por punto de acceso.

Tabla 3-2 Distribución de los roles de los Dispositivos

Receptor	No IoT	IoT
sta4	sta24	sta9
		sta14
sta8	sta18	sta17
		sta12
sta11	sta15	sta5
		sta21
sta16	sta10	sta22
		sta23
sta19	sta6	sta13

		sta7
sta20	sta2	sta3
		sta1

Tabla 3-3 Distribución de Dispositivos por punto de acceso

Ap1	Ap2	Ap3	Ap4	Ap5
sat1	sta5	sta9	sta15	sta20
sta2	sta6	sta10	sta16	sta21
sta3	sta7	sta11	sta17	sta22
sta4	sta8	sta12	sta18	sta23
		sta13	sta19	sta24
		sta14		

Una vez especificados los dispositivos, se procede a generar las terminales de cada uno para su respectiva configuración; este proceso se hace con el comando *xterm* seguido de las estaciones que se desean configurar; en la Figura 3-10 es posible observar el resultado de ejecutar el comando anterior.

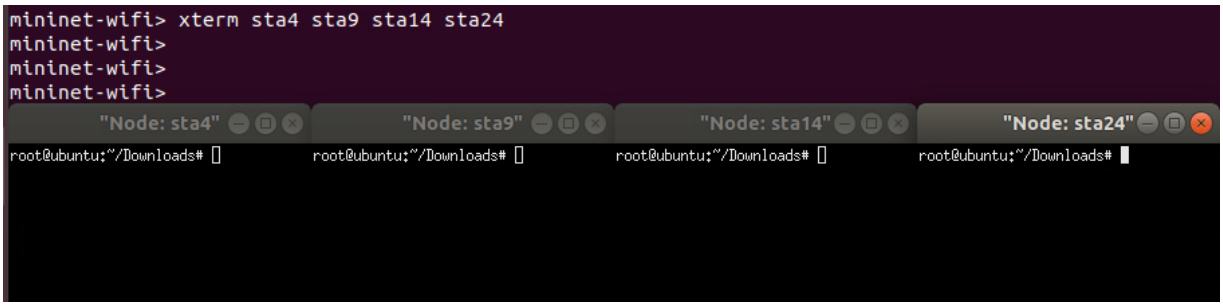

Figura 3-10 Ventana CLI de las estaciones

Teniendo en cuenta la distribución establecida en la Tabla 3-2, se procede a realizar la configuración de los receptores con el comando *ITGRecv receiver.log* y los emisores con el comando *ITGSend -T (TCP/UDP) -a x.x.x.x -c xxx -C xxxx -t xxxxx -l sender.log -x receiver.log*; en las Figura 3-11 y Figura 3-12 se observa la respectiva configuración de uno de los terminales que servirán de ejemplo para los demás.

Figura 3-11 Configuración del equipo como receptor

Figura 3-12 Configuración de equipos como emisores

Los parámetros establecidos para la configuración de los emisores son: el protocolo del mensaje, se enviarán mensajes tanto mensajes TCP como UDP, la dirección IP configurada para la red es la *10.1.0.0/24,* por tanto la que se asignó a cada dispositivo corresponde al mismo número del dispositivo iniciando en 100, es decir, la estación 1 tendrá la *10.1.0.101*, el dispositivo 2 se le asignará la *10.1.0.102* y así sucesivamente, la cantidad de bytes establecido dependerá de cada dispositivo y consideran lo planteado en [46], en la Tabla 3-4 se muestra el tipo de dispositivo IoT que representaría cada equipo terminal así como la cantidad bytes a transmitir, en cuando a la cantidad de paquetes por segundo que se enviaran se estableció que esta fuera constante para todos, la cual será de 1000 paquetes por segundo, el tiempo de transmisión será acorde con lo definido en la Tabla 3-5.

Tabla 3-4 Tipo de Dispositivo y cantidad de Bytes

Tipo de Dispositivo	Dirección IP	Bytes por paquetes
Cámara de Seguridad	10.1.0.109	600
	10.1.0.123	600
Monitor de Bebés	10.1.0.114	80
	10.1.0.103	80
Smart TV	10.1.0.112	1500
	10.1.0.122	1500
Iluminación	10.1.0.117	500

	10.1.0.101	500
Videojuego	10.1.0.105	1200
	10.1.0.113	1200
Electrodomésticos	10.1.0.121	550
	10.1.0.107	550

Tabla 3-5 Detalles del tráfico a enviar

Nombre	Tiempo de transmisión	Descripción
Run 1	10 min	Se envía tráfico con una constante de 1000 paquetes por dispositivo y una cantidad de datos que dependerá del tipo de dispositivos entre los dispositivos IoT y receptores, además de un envío contante de tráfico de los dispositivos no IoT.
Run 2	30 min	
Run 4	1 hora	
Run 4	3 hora	
Run 4	6 hora	

Al finalizar el envío de paquetes con la herramienta D-ITG, esta arroja un registro *"receiver.log"* con la información de algunos parámetros como el porcentaje de paquetes perdidos, el *jitter*, el *delay* promedio, un ejemplo de este resultado se observa en la Figura 3-13.

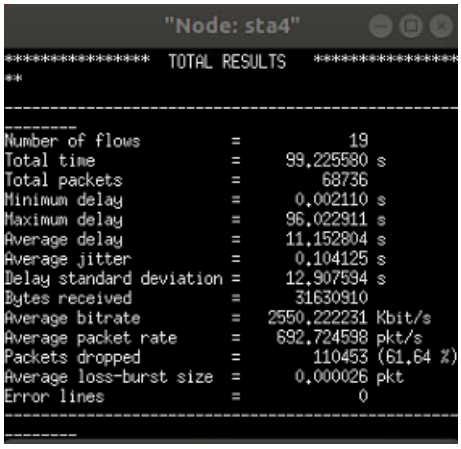

Figura 3-13 Registro de datos arrojado por el DITG

3.4. Aplicación ONOS

Para empezar a implementar el control de tráfico basado en ML es necesario implementar una nueva aplicación con la lógica del algoritmo y luego integrarla al núcleo de ONOS. Este controlador trae una plantilla para crear una nueva app, la cual trae por defecto el nombre de *foo-app,* la forma en la que se descarga e instala dicha plantilla junto con un pequeño ejemplo de verificación se encuentra detallada en el ANEXO D.

Una vez terminada la configuración inicial y las respectivas adecuaciones en el sistema, se comenzará con la modificacion de la plantilla de la aplicación para adaptarla al trabajo de grado. Este proceso se realiza en en el archivo *AppComponent.java* el cual es el esqueleto de la aplicación y se encuentra en el directorio foo-app/src/main/java/org/foo/app/; dentro de este archvo se tienen definidos dos métodos que permitirán la activación o desactivación de la aplicación, estos métodos se encuentran en la Figura 3-14 como @Activate y @Deactivate.

```
@Activate
protected void activate() {
    cfgService.registerProperties(getClass());
    log.info("Started");
}

@Deactivate
protected void deactivate() {
    cfgService.unregisterProperties(getClass(), false);
    log.info("Stopped");
}
```

Figura 3-14 Métodos de activación y desactivación de la App

Con el fin de optimizar el código de mejor manera, se trabajará por medio de clases y métodos; se tiene 3 clases la *AppComponent*, la cual se mencionó anteriormente y es la que permite activar y desactivar la aplicación en el núcleo de ONOS, la clase *AppCommand*, la cual es la que se encarga de llamar la información del controlador con la que se alimenta el algoritmo ML el cual está en una tercera clase llamada *Qlearning* y finalmente con el resultado que obtenga se instauran las reglas de flujo

necesarias en el controlador para que este haga el envío el paquetes conforme a los resultados.

A continuación de describirá cada clase implementada en el desarrollo de la aplicación; sin embargo, los códigos completos de cada clase estarán en el Anexo H Comenzando por la clase principal *AppCommand*, lo que se hace en primera instancia es crear las listas con un nombre, esto se hace con el comando *Public static ArrayList<Tipo_Dato> Nombre_lista = new ArrayList<Tipo_Dato>()*, dichas listas se usan más adelante para almacenar la información de la red y de esa manera hacer más sencilla la tarea de procesar la información. En la Figura 3-15, se observa la cantidad de listas creadas para este desarrollo y adicional a ello se establece un tiempo de sondeo con el comando *private statis Long Nombre_Variable = Tiempo_en_ms*, el cual corresponde al tiempo en el que se actualizan los datos de la red, es decir se evalúa la red cada 5s, este valor es debido a que es el tiempo mínimo que requiere el controlador para llevar a cabo una acción sin perder su estabilidad.

```
private static long t_s = 5;

public static boolean flag = true;
public static ArrayList<Boolean> flag_time = new ArrayList<Boolean>();
public static ArrayList<Integer> t_c = new ArrayList<Integer>();
public static ArrayList<Float> PktRx = new ArrayList<Float>();
public static ArrayList<Float> PktTx = new ArrayList<Float>();
public static ArrayList<Float> PktRxError = new ArrayList<Float>();
public static ArrayList<Float> PktTxError = new ArrayList<Float>();
public static ArrayList<Float> BytesRx = new ArrayList<Float>();
public static ArrayList<Float> BytesTx = new ArrayList<Float>();
public static ArrayList<Float> PktRxDrp = new ArrayList<Float>();
public static ArrayList<Float> PktTxDrp = new ArrayList<Float>();
```

Figura 3-15 Obtención de parámetros de ONOS

Una vez creadas las listas, se procede a crear una variable de tipo entero para almacenar el número de host que se tengan en la red, luego mediante un ciclo repetitivo *if* se crea una variable *aux* de tipo *float* la cual se iguala a 0, todos estos parámetros se usan para darle un tamaño a la listas creadas por medio de un ciclo *for*, como se observa en la Figura 3-16, en cual se le indica al sistema que por medio del *for* llene las listas desde la posición 1 hasta el número de host establecido en la variable *numHost* con incrementos de 1 y luego proceda a inicializarlas con el comando *Nombre_Lista.add(aux).*

```
if(flag) {

 float aux = 0;
  for(int i=1;i<=(25);i++) {

      BytesTx.add(aux);
      PktRx.add(aux);
      PktTx.add(aux);
      BytesRx.add(aux);
      PktTxError.add(aux);
      PktRxError.add(aux);
      PktRxDrp.add(aux);
      PktTxDrp.add(aux);

   }
}
flag = false;
```

Figura 3-16 Inicialización de las listas

Una vez se tengan creadas e inicializadas las tablas se procede a solicitarle la información al controlador para almacenarla en las tablas, esto se hace por medio del método *PortStats* el cual se ve en la Figura 3-17, en este método lo que se hace es asignarle los datos que se obtienen de ONOS con el comando *stat.Nombre_Variable* a las listas; *stat* hace referencia al arreglo donde están almacenados los datos en la herramienta, adicional a esto se realiza una serie de if anidados para asignarle un valor de 1 cuando el puerto no este transmitiendo o que no se estén perdiendo paquetes, lo que indica que es la mejor condición dentro de la red a la hora de transmitir; teniendo claro esto, con el código mostrado en la Figura 3-18 se recorre los distintos dispositivos que están en la red por medio de una ciclo for y llamando al método PortStats se obtiene los datos de los dispositivos.

```
private void printPortStats(DeviceId deviceId, Iterable<PortStatistics> portStats, int Ndevice, int sequence) {
    int cont2=0;
    cont2 = (Ndevice*5)-5+cont2;
    List<Port> ports = get(DeviceService.class).getPorts(deviceId);
     for (Port p: ports) {
        if(p.isEnabled()){
            PortStatistics traffic = get(DeviceService.class).getDeltaStatisticsForPort(deviceId, p.number());
            System.out.println("  \n trafico del puerto "+traffic);
        }
    }
    for (PortStatistics stat : sortByPort(portStats)) {
          float pktRx = stat.packetsReceived();
          float pktTx = stat.packetsSent();
          float bytesRx = stat.bytesReceived();
          float bytesTx = stat.bytesSent();
          float pktRxError= stat.packetsRxErrors();
          float pktTxError= stat.packetsTxErrors();
          float pktRxDrp = stat.packetsRxDropped();
          float pktTxDrp = stat.packetsTxDropped();
          float i=1;
          if (bytesRx==0){
            bytesRx=i;
          }
          if (bytesTx==0){
            bytesTx=i;
          }
          if (pktRx==0){
            pktRx=i;
          }
          if (pktTx==0){
            pktTx=i;
          }
          if (bytesRx==0){
            bytesRx=i;
          }
          if (pktTxError==0){
            pktTxError=i;
          }
          if (pktRxError==0){
            pktRxError=i;
          }
          if (pktRxDrp==0){
            pktRxDrp=i;
          }
          if (pktTxDrp==0){
            pktTxDrp=i;
          }
          float Tx = pktTxDrp*pktTxError*bytesTx;
          float Rx = pktRxDrp*pktRxError*bytesRx;
          BytesTx.set(cont2,Tx);
          BytesRx.set(cont2,Rx);
        cont2++;
```

Figura 3-17 Método para obtener los datos de los dispositivos del controlador

```
int i=1;
int cont = 1;

for (Device d : getSortedDevices(deviceService)) {
   if (cont<=Nhost_Real) {
      printPortStats(d.id(),
            deviceService.getPortDeltaStatistics(d.id()),cont,i);
             }
   cont++;
         }
```

Figura 3-18 Llamado del método PortStats para obtener datos del controlador

Ya con todos los datos almacenados y organizados en las lista, se procede a llamar la clase que contiene el algoritmo ML es la Qlearning, la forma de llamar la clase y usarla se observa en la Figura 3-19, donde se crea la variable *obj*, y sobre este llamar el método de la clase.

```
    Qlearning obj = new Qlearning();

                obj.run(BytesTx,BytesRx);
                obj.printResult();
                obj.showPolicy();
    try {
        Thread.sleep(t_s);
    } catch (InterruptedException ignored) {
}
```

Figura 3-19 Llamado de la clase Qlearning

El paso siguiente en esta clase, es implementar un método para retornar los datos de un punto del acceso, este método se observa en la Figura 3-20, donde se crea una lista para almacenar la información arrojada por los puertos de los respectivos AP.

```
private static List<PortStatistics> sortByPort(Iterable<PortStatistics> portStats) {
    List<PortStatistics> portStatsList = Lists.newArrayList(portStats);
    portStatsList.sort(Comparator.comparing(ps -> ps.portNumber().toLong()));
    return portStatsList;

}
```

Figura 3-20 Método para retornar los datos de un dispositivo

Ahora con el resultado obtenido en los métodos anteriores, es necesario crear una regla de flujo para indicarle al controlador cual es la ruta que deben seguir los paquetes que envía el dispositivo que tiene mayor prioridad entre los demás. Para ello, dentro de la misma clase principal *AppCommand* se llaman las respectivas clases de las librerías importadas previamente, estas clases se observan en la Figura 3-21, después de llamar estas clases, se establece la prioridad que tendrá la regla de flujo instaurada por la aplicación desarrollada. Ahora, para que esta regla de flujo sea prioridad para el controlador, es decir para que el software le dé más importancia a la regla de la aplicación desarrollada, es necesario que se le asigne un valor mucho más alto a las reglas que se están ejecutando de las demás aplicaciones que contiene el controlador, es por esto que teniendo en cuenta el valor más alto que tenían las otras aplicaciones el cual oscilaba entre 5, 10 y 4000, que se estableció una variable prioridad con un

valor de 65000 para asegurar que el sistema siempre tome primero las reglas de la aplicación diseñada.

```
private int flowTimeout = FLOW_TIMEOUT_DEFAULT;
private int flowPriority = FLOW_PRIORITY_DEFAULT;
private ApplicationId appId;

@Reference(cardinality = ReferenceCardinality.MANDATORY)
protected CoreService coreService;

@Reference(cardinality = ReferenceCardinality.MANDATORY)
protected DeviceService deviceService;

@Reference(cardinality = ReferenceCardinality.MANDATORY)
protected HostService hostService;

@Reference(cardinality = ReferenceCardinality.MANDATORY)
protected HostProbingService hostProbingService;

@Reference(cardinality = ReferenceCardinality.MANDATORY)
protected FlowObjectiveService flowObjectiveService;

@Reference(cardinality = ReferenceCardinality.MANDATORY)
protected FlowRuleService flowRuleService;

@Reference(cardinality = ReferenceCardinality.MANDATORY)
protected PacketService packetService;

@Reference(cardinality = ReferenceCardinality.MANDATORY)
protected PacketContext context;

@Reference(cardinality = ReferenceCardinality.MANDATORY)
protected TopologyService topologyService;

 int priority = 65000;
```

Figura 3-21 Importación de clases para instaurar las reglas de flujo

El siguiente paso para instaurar la regla de flujo, es establecer cuál es el origen y destino de la transmisión, para ello se obtienen datos como la dirección física de los respectivos dispositivos involucrados en el envío de información, este proceso se realiza con las líneas de código que se observan en la Figura 3-22, donde después de obtener los parámetros con los métodos *get* se usa un *paht* para obtener las rutas menos congestionadas entre la fuente y el destino.

```
appId = get(CoreService.class).getAppId("org.foo.app");

DeviceService deviceService = get(DeviceService.class);

int numHosts = get(HostService.class).getHostCount();
Host HosDts = get(HostService.class).getHost(hostId("02:00:00:00:13:00/None"));
Host HosSrt = get(HostService.class).getHost(hostId("02:00:00:00:00:00/None"));
Device deviceId = get(DeviceService.class).getDevice(deviceId("of:000000acfec00001"));
installRule(deviceId, HosDts, HosSrt);
int Nhost_Real = numHosts;
Topology topology = get(TopologyService.class).currentTopology();

Set<Path> paths = get(TopologyService.class).getPaths(topology,
    HosSrt.location().deviceId(),
    HosDts.location().deviceId());

System.out.println("\n paths" + paths);
```

Figura 3-22 Obtención de parámetros para instaurar la regla de flujo

Finalmente se crea un método para crear e instaurar la regla de flujo en el controlador, en la Figura 3-23 se observa dicho método.

```
private void installRule( Device deviceId,Host HosSrt, Host HosDts) {

    MacAddress macAddressSrc= HosSrt.mac();
    MacAddress macAddressDts= HosDts.mac();
    TrafficSelector.Builder selectorBuilder = DefaultTrafficSelector.builder();
    long puerto = 1;
    selectorBuilder.matchEthSrc(macAddressSrc)
    .matchEthDst(macAddressDts);
    TrafficTreatment treatment;
    treatment = DefaultTrafficTreatment.builder()
                    .setOutput(PortNumber.portNumber(puerto))
                    .build();

    ForwardingObjective forwardingObjective = DefaultForwardingObjective.builder()
    .withSelector(selectorBuilder.build())
    .withTreatment(treatment)
    .withPriority(flowPriority)
    .withFlag(ForwardingObjective.Flag.VERSATILE)
    .fromApp(appId)
    .makeTemporary(flowTimeout)
    .add();

}
```

Figura 3-23 Método para crear la regla de flujo

En cuanto a la clase *Qlearning,* aquí se encuentra la desarrollado el algoritmo ML, el cual se hizo siguiendo la lógica del diagrama de flujo presentado en el apartado 2.4.2. Inicialmente, se establecen las variables necesarias para aplicar la fórmula de *Bellman's*, dichas variables son las constantes Alpha y gamma, adicional a ello se declaran otras variables de tipo entero y se les asigna valores del 0 en adelante que

corresponderán a los diferentes puntos de acceso con los que cuenta la red, en la Figura 3-24 se observan las variables que se declararon, tenga en cuenta para esta ocasión dichas variables se declararon usando la forma *final* puesto que el valor asignado no va a cambiar mientras se ejecuta el algoritmo.

```
final DecimalFormat df = new DecimalFormat("#.##");
final double alpha = 0.1;
final double gamma = 0.9;
final int ap1 = 0;
final int ap2 = 1;
final int ap3 = 2;
final int ap4 = 3;
final int ap5 = 4;
```

Figura 3-24 Declaración de variables

Una vez identificados los dispositivos y asignadas las variables correspondientes, se procede a definir los estados y las posibles acciones del algoritmo; para ello se define un vector de tipo entero llamado *states*, el cual se observa en la Figura 3-25, donde se almacenarán los diferentes puntos de acceso encontrados en la red; adicional a ello también se declara una variable de tipo entero a la cual se le asigna un valor igual al número de APs que a su vez será el tamaño de las matrices de la tabla de valores Q.

```
final int statesCount = 5;
final int[] states = new int[]{ap1,ap2, ap3, ap4, ap5};
```

Figura 3-25 Definición de los estados de la tabla

Al tener definidos los estados, solo falta determinar las diferentes acciones, estas se establecen conforme a las conexiones de los puntos de acceso, en la Figura 3-26, donde se crearon 5 vectores de tipo entero indicando las acciones de cada AP, las cuales están definidas por las conexión directas con los demás puntos de acceso, por tanto, estos vectores almacenarán dichas conexiones, por ejemplo en la red trabajada el AP1 está conectado al AP2,3 por tanto su vector de acciones se define con la siguiente línea de comando *int [] actionsFromAp1 = new int [] {ap2, ap3, ap4}* y de esta manera se configuran las acciones de los demás puntos de acceso. Finalmente se crea un nueva matriz de tipo entero que tendrá como parámetros las acciones; y finalmente se crea un vector de tipo *string* que almacenará los nombres de los estados.

```
int[] actionsFromAp1 = new int[] { ap2, ap3, ap4 };
int[] actionsFromAp2 = new int[] { ap1, ap3, ap5 };
int[] actionsFromAp3 = new int[] { ap1, ap2, ap4, ap5 };
int[] actionsFromAp4 = new int[] { ap1, ap3, ap5 };
int[] actionsFromAp5 = new int[] { ap3, ap4, ap2 };

int[][] actions = new int[][] { actionsFromAp1, actionsFromAp2, actionsFromAp3,
        actionsFromAp4, actionsFromAp5};

String[] stateNames = new String[] { "AP1", "AP2", "AP3", "AP4", "AP5" };
```

Figura 3-26 Definición de las acciones

El siguiente paso en el diagrama de flujo, es establecer las posibles recompensas que puede obtener una acción ejecutada, para esto primero se debe tener en cuenta a que punto de acceso está conectado el destino; luego se establecen los enlaces directos que llevan al destino, y a estos se les asigna un valor máximo de recompensa; adicional a esto se debe tener en cuenta el número de bytes por puerto que transmitiendo un dispositivo, dado que esto ayudará a establecer cuál es menos congestionado y a este es al que se dará una mayor recompensa; en la Figura 3-27, se le asigna un valor específico a la tasa de cada enlace entre los APs.

```
public void init() {

    R[ap2][ap5] = 100;
    R[ap3][ap5] = 100;
    R[ap4][ap5] = 100;

    rate[ap1][ap2]=1;
    rate[ap1][ap3]=2;
    rate[ap1][ap4]=3;

    rate[ap2][ap1]=6;
    rate[ap2][ap3]=7;
    rate[ap2][ap5]=8;

    rate[ap3][ap1]=11;
    rate[ap3][ap2]=12;
    rate[ap3][ap4]=13;
    rate[ap3][ap5]=14;
```

Figura 3-27 Parámetros para definir la recompensa

Ya establecidos los parámetros iniciales, se procede con la ejecución del algoritmo, es decir que se seleccione una acción, que se le asigne una recompensa y se repita hasta que se complete un episodio.

Los siguientes métodos definirán esta acción; en la Figura 3-28 se establecen los ciclos que van a recorrer la matriz de valores Q, ejecutará la ecuación, asignará una recompensa y actualizará la matriz. Inicialmente se crea una variable rand cual es una variable de tipo Random que le permitirá al sistema seleccionar una acción de forma aleatoria, una vez establecido esto, se inicia un *for* que va a ir desde la posición 0 hasta la 1000 con incrementos de 1, donde para un acción aleatoria hace:

se declara una variable de tipo entero para indicar el próximo estado *actionsFromState* y se le asignará el valor de la acción del estado actual, ahora se crea una variable de tipo entero *index* donde se almacenará el tamaño del vector de la acción próxima y se convierte a entero, de esta manera el siguiente estado será igual al resultado de la acción ejecutada; posterior a ello, se llama el método *Q*, el cual se observa en la Figura 3-29, donde lo que se hace es pasarle dos parámetros que son los estados y las acciones y de esta manera se forma el término del valor *Q(state,action)* actual dentro de la ecuación, el siguiente método a llamar es el *maxQ* el cual ayudará a determinar el parámetros de máxima recompensa futura de la ecuación de Bellman's y el cual se puede ver en la Figura 3-30 y finalmente se establece el último término que haría falta para evaluar la ecuación, este método en el *R(state,action)* el cual se identifica como la recompensa; ahora bien, para obtener la información del transmisor se multiplica tanto los bit recibidos por los transmitidos y se pasa de *Bytes* a *KBytes* y ahora si ya se cuenta con todos los parámetros necesario para evaluar la ecuación.

```
    Random rand = new Random();
    for (int i = 0; i < 1000; i++) {
        int state = rand.nextInt(statesCount);
        int[] actionsFromState = actions[state];
        int index = rand.nextInt(actionsFromState.length);
        int action = actionsFromState[index];
        int nextState = action;
        double q = Q(state, action);
        double maxQ = maxQ(nextState);
        int r = R(state, action);
        int portTx = portTx(state, action);
        int portRx = portTx( action,state);
        float costo ;
        costo= (1000000)/(BytesTx.get(portTx)*BytesRx.get(portRx));
        double value = (q + alpha * costo*(r + gamma * maxQ - q));
        setQ(state, action, value);
        state = nextState;

    }
}
```

Figura 3-28 Método para la ejecución del algoritmo

```
double Q(int s, int a) {
    return Q[s][a];
}

void setQ(int s, int a, double value) {
    Q[s][a] = value;
}

int R(int s, int a) {
    return R[s][a];
}
```

Figura 3-29 Definición de los métodos Q(S,A) y R(S,A)

```
private int portTx(int state, int action) {
    return portTx[state][action];
}

double maxQ(int s) {
    int[] actionsFromState = actions[s];
    double maxValue = Double.MIN_VALUE;
    for (int i = 0; i < actionsFromState.length; i++) {
        int nextState = actionsFromState[i];
        double value = Q[s][nextState];

        if (value > maxValue)
            maxValue = value;
    }
    return maxValue;
}
```

Figura 3-30 Método para obtener la Máxima recompensa futura

Finalmente, se crea el método mostrado en la Figura 3-31 para evaluar la matriz de valores Q y seleccionar el valor de recompensa más alto obtenido por estado; se toma una posición del vector *actionsFromState*, luego se define una variable de tipo *double* a la cual se le asigna la función *Double.MIN_VALUE* con el fin de igualar cada valor de la tabla de valores Q con dicho valor para garantizar que la recompensa que se está tomando sea la de mayor valor; como última instancia se desarrollan los métodos mostrados en la Figura 3-32, los cuales permiten obtener la lista de posibles caminos que puede tomar los paquetes y mostrarla.

```
int policy(int state) {
    int[] actionsFromState = actions[state];
    double maxValue = Double.MIN_VALUE;
    int policyGotoState = state;
    for (int i = 0; i < actionsFromState.length; i++) {
        int nextState = actionsFromState[i];
        double value = Q[state][nextState];

        if (value > maxValue) {
            maxValue = value;
            policyGotoState = nextState;
        }
    }
    return policyGotoState;
}
```

Figura 3-31 Método para evaluar la matriz

```
void printResult() {
    System.out.println("Print result");
    for (int i = 0; i < Q.length; i++) {
        System.out.print("out from " + stateNames[i] + ":  ");
        for (int j = 0; j < Q[i].length; j++) {
            System.out.print(df.format(Q[i][j]) + " ");
        }
        System.out.println();
    }
}

void showPolicy() {
    System.out.println("\nshowPolicy");
    for (int i = 0; i < states.length; i++) {
        int from = states[i];
        int to =  policy(from);
        System.out.println("from "+stateNames[from]+" go to "+stateNames[to]);
    }
```

Figura 3-32 Métodos para obtener los resultados del algoritmo ML

3.5. Simulación de los escenarios y casos de estudio

Para los casos de estudio se establecieron distintos tiempos de simulación; lo primero es arrancar la red desde mininet, para ello se abre una ventana, se buscar el directorio donde este almacenado el archivo python y se ejecuta el siguiente comando.

sudo python nombre_del_archivo.py –controller remote, ip=x.x.x.x

Otro aspecto para tener en cuenta es la variación de dos parámetros, hay que considerar el ancho de banda de los enlaces entre AP, y el otro parámetro es de los dispositivos inalámbricos. La configuración de los enlaces entre AP se hace en el código de la red con el comando *ap1ap2 = {'bw=100'}* mostrado en la Figura 3-5, este parámetro se varía entre los valores de 20, 50 y 100 Mbps. En cuanto a la configuración de la tasa de transmisión de los dispositivos, se debe tener en cuenta que la herramienta no permite modificar la tasa de trasmisión por código, la única forma que se encontró para variar la tasa es cambiar el modo de configuración del estándar 802.11, el cual se realiza en el parámetro *mode* = como se ve en la Figura 3-4, los reléase seleccionados del estándar fueron en b, g y n en la banda de los 2.4 GHz con una capacidad de canal de 20 MHz, ya que es la configuración más usada para dispositivos de gama media-baja, en la Tabla 3-6 se observa las velocidades teóricas y reales de estos estándares y que se tendrán en cuenta a la hora de ejecutar las simulaciones.

Tabla 3-6 Tasas de transmisión del protocolo 802.11b,g,n en la banda de 2.4 GHz

Protocolo	Banda de Frecuencia	Velocidad Teórica	Velocidad Real
802.11b	2.4 GHz	11 Mbps	5.5 Mbps
802.11g	2.4 GHz	54 Mbps	22 Mbps
802.11n	2.4 GHz	300 Mbps	52 Mbps

3.5.1. Escenario de Simulación 1 sin el algoritmo ML (Caso 1)

Para el caso de estudio número 1, se tomó la red en su configuración inicial, la cual es la misma que se observa en la Figura 3-8 y se realizó la siguiente configuración: Inicialmente se abrieron los terminales externos tanto para los dispositivos emisores como receptores; posterior a ello se procedió a realizar la configuración teniendo en cuenta la cantidad de bytes estipulada para cada dispositivo en la Tabla 3-4 y los roles de cada uno descritos en la Tabla 3-2; y en la Tabla 3-5 se determinaron los distintos intervalos de tiempo para mirar el comportamiento de la red, para tener más veracidad con los resultados cada simulación se realizará 3 veces. La configuración para cada simulación es la siguiente:

- **Para los dispositivos receptores:** Una vez abierto el terminal se ingresa al directorio “Downloads/D-ITG-2.8.1-r1023/bin” y se ingresa el comando ./ITGRecv -l receiver.log

- **Para los dispositivos no IoT:** Estos dispositivos serán usado para ayudar a generar tráfico en la red, por tanto, se configurarán para que hagan un ping constante al receptor; para ellos se ejecuta el comando ping seguido de la dirección IP del receptor.

- **Para los dispositivos IoT:** la configuración de estos se realiza como se explica en la Figura 3-12; solo se debe tener en cuenta el tiempo de la transmisión el cual es el parámetro que se modifica.

Como se mencionó anteriormente aunque la mayoría de los equipos son IoT, para realizar la evaluación del algoritmo solo se tomó un dispositivo como prioritario y los demás se dejaron para generar caos en la red, en la Figura 3-33 se observa el sistema en funcionamiento; donde en la parte superior en la primera fila se encuentran los receptores, seguido en la segunda fila con los dispositivos no IoT que se dejaron para enviar un ping de forma constante todo el tiempo, en la tercera y cuarta fila están los dispositivos IoT que se configuraron para enviar tráfico UDP y TCP respectivamente por un tiempo de 6 horas; ahora al lado derecho de la imagen se encuentra la interfaz web del Controlador, en la cual se observa cómo se envían paquetes a través de los

enlaces; en la parte inferior de la venta se observa el dispositivo con prioridad, el cual se está enviando tráfico a su receptor, en la Figura 3-34 se observa el funcionamiento este dispositivo.

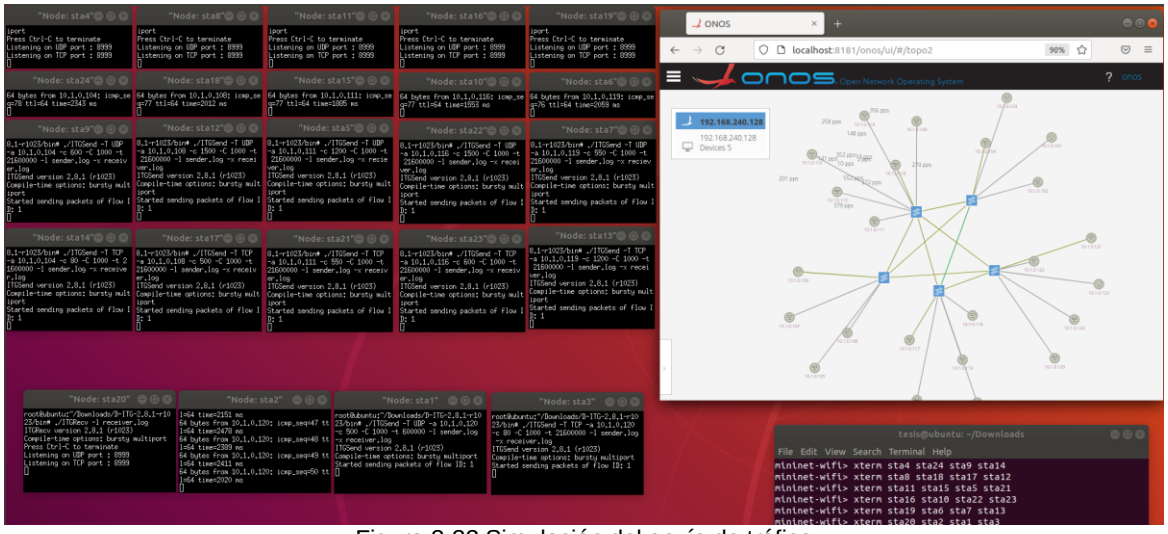

Figura 3-33 Simulación del envío de tráfico

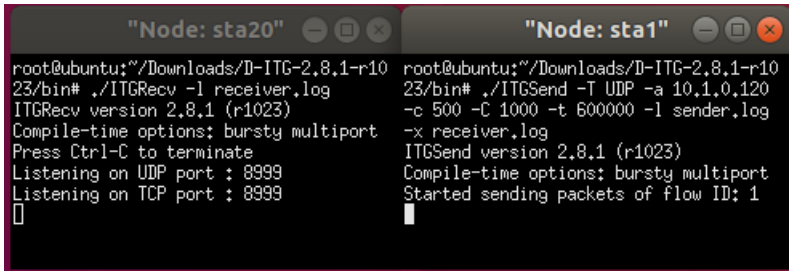

Figura 3-34 Simulación del Dispositivo con prioridad

3.5.2. Escenario de Simulación 2 sin el algoritmo ML (Caso 2)

Para el caso de estudio dos, se realizaron las mismas configuraciones del caso anterior y los mismos intervalos de tiempo para la transmisión, sin embargo, con el fin de evaluar la red, en este caso se dará de baja dos de los enlaces entre los puntos de acceso, para ello se emplea el comando *link [nodo1] [nodo2] down* en el CLI de mininet, tal como se ve en la Figura 3-35.

```
mininet-wifi> link ap1 ap4 down
mininet-wifi> link ap2 ap5 down
mininet-wifi>
```

Figura 3-35 Comando para dar de baja los enlaces en la terminal de mininet

Una vez se haya ejecutado el comando para dar de baja los enlaces, el resultado podrá verse en la interfaz web de ONOS tal como se observa en la Figura 3-36

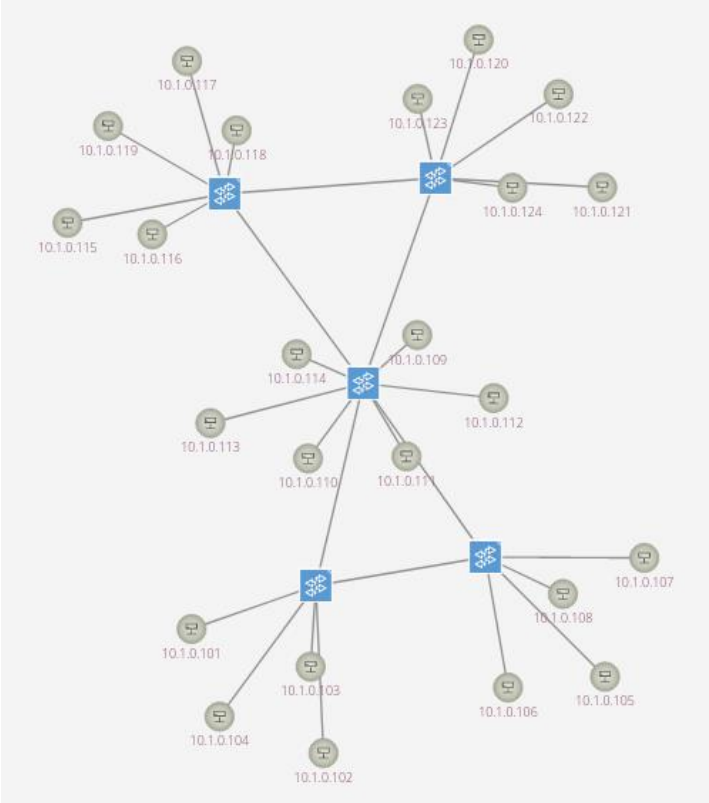

Figura 3-36 Verificación de la red después de dar de baja dos enlaces

La configuración de los parámetros se hace exactamente igual al del escenario anterior y al igual que como se explicó para el caso se varían las tasa de transmisión de los enlaces y los dispositivos; en las Figura 3-37, Figura 3-38 y Figura 3-39 se observa los respectivos resultados de la configuración del protocolo 802.11 en los dispositivos.

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:18:00 (on sta1-wlan0)
        SSID: ap1-ssid
        freq: 2412
        RX: 248671 bytes (4872 packets)
        TX: 65126 bytes (656 packets)
        signal: -75 dBm
        rx bitrate: 5.5 MBit/s
        tx bitrate: 5.5 MBit/s

        bss flags:
        dtim period:    2
        beacon int:     100
mininet-wifi>
```

Figura 3-37 Parámetros configuración del dispositivo 1 con el estándar 802.11b

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:18:00 (on sta1-wlan0)
        SSID: ap1-ssid
        freq: 2412
        RX: 64705 bytes (1146 packets)
        TX: 8190 bytes (94 packets)
        signal: -75 dBm
        rx bitrate: 36.0 MBit/s
        tx bitrate: 36.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
mininet-wifi>
```

Figura 3-38 Parámetros configuración del dispositivo 1 con el estándar 802.11g

```
mininet-wifi> sta1 iw dev sta1-wlan0 link
Connected to 02:00:00:00:18:00 (on sta1-wlan0)
        SSID: ap1-ssid
        freq: 2412
        RX: 260621 bytes (5038 packets)
        TX: 30606 bytes (336 packets)
        signal: -73 dBm
        rx bitrate: 54.0 MBit/s
        tx bitrate: 54.0 MBit/s

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
mininet-wifi>
```

Figura 3-39 Parámetros configuración del dispositivo 1 con el estándar 802.11n

3.5.3. Escenarios de Simulación 1 y 2 con el algoritmo ML (Caso 3 y caso 4)

En este caso se procede a cargar la aplicación con el algoritmo ML en el núcleo de ONOS; en las Figura 3-40 y Figura 3-41 se observan los comandos utilizados para compilar y cargar la aplicación en el controlador; una vez se cargue la aplicación se verifica en el CLI del controlador que se haya cargado la aplicación como se ve en la Figura 3-42 y posterior a ello se hace el envío de tráfico con las mismas características de los escenarios anteriores. Inicialmente se envía el tráfico como está configurado en la Figura 3-33 con todos los enlaces habilitados. Una vez se empieza a correr la red se abre el CLI de ONOS para verificar la ruta que seleccionó, en la Figura 3-43A se observa el resultado obtenido sobre el camino que debe seguir, donde se ve que toma los mayores valores de recompensa y determina la ruta que del AP1 va al AP2 y de este al AP5; después de unos minutos se volvió a probar el sistema y arrojo una ruta distinta a la anterior, también observa que las recompensas cambiaron, este resultado se observa en la Figura 3-43B, donde la ruta seleccionada va del AP1 al AP4 y de este al AP5.

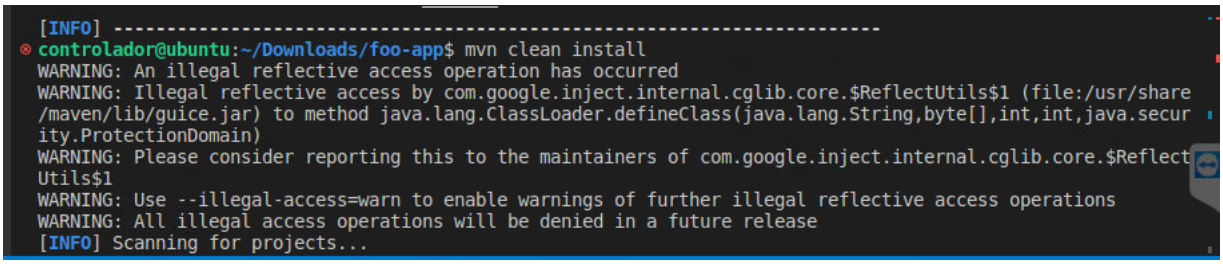

Figura 3-40 Comando para compilar la aplicación

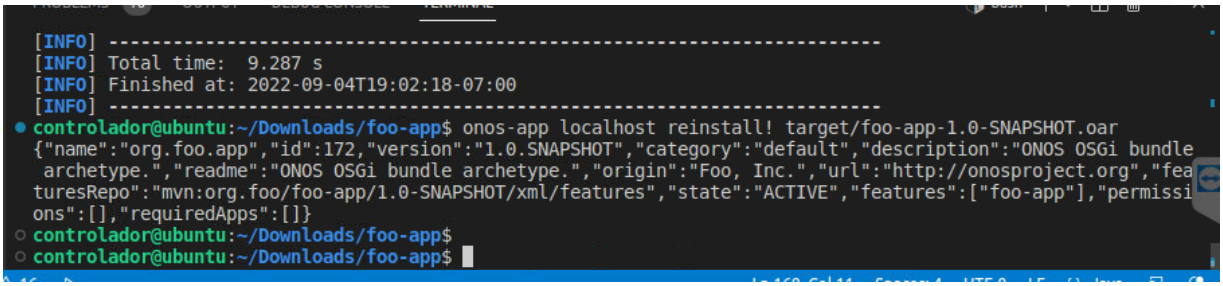

Figura 3-41 Comando para instalar la aplicación

```
controlador@root log> log:tail                                                  19:03:33
14:32:14.796 INFO [FeaturesServiceImpl] Refreshing bundles:
14:32:14.797 INFO [FeaturesServiceImpl]     org.foo.foo-app/1.0.0.SNAPSHOT (Bundle will be uninstalled)
14:32:14.798 INFO [FeaturesServiceImpl] Done.
14:32:14.799 INFO [ApplicationManager] Application org.foo.app has been uninstalled
14:32:14.833 INFO [FeaturesServiceImpl] Adding features: foo-app/[1.0.0.SNAPSHOT,1.0.0.SNAPSHOT]
14:32:15.403 INFO [FeaturesServiceImpl] Changes to perform:
14:32:15.403 INFO [FeaturesServiceImpl]   Region: root
14:32:15.403 INFO [FeaturesServiceImpl]     Bundles to install:
14:32:15.404 INFO [FeaturesServiceImpl]       mvn:org.foo/foo-app/1.0-SNAPSHOT
14:32:15.404 INFO [FeaturesServiceImpl] Installing bundles:
14:32:15.405 INFO [FeaturesServiceImpl]   mvn:org.foo/foo-app/1.0-SNAPSHOT
14:32:15.420 INFO [FeaturesServiceImpl] Starting bundles:
14:32:15.420 INFO [FeaturesServiceImpl]   org.foo.foo-app/1.0.0.SNAPSHOT
14:32:15.426 INFO [AppComponent] Started
14:32:15.429 INFO [CommandExtension] Registering commands for bundle org.foo.foo-app/1.0.0.SNAPSHOT
14:32:15.430 INFO [FeaturesServiceImpl] Done.
14:32:15.431 INFO [ApplicationManager] Application org.foo.app has been activated
14:32:15.678 INFO [AppComponent] Reconfigured
14:32:26.119 INFO [ServerUserAuthService] Session controlador@127.0.0.1:52298 authenticated
14:42:44.252 WARN [ServerSessionImpl] exceptionCaught(ServerSessionImpl[controlador@/127.0.0.1:52298])[state=Opened] InterruptedByTimeoutException: null
18:59:13.432 INFO [ServerUserAuthService] Session controlador@127.0.0.1:56118 authenticated
19:02:24.213 INFO [FeaturesServiceImpl] Removing features: foo-app/[1.0.0.SNAPSHOT,1.0.0.SNAPSHOT]
19:02:24.870 INFO [FeaturesServiceImpl] Changes to perform:
19:02:24.871 INFO [FeaturesServiceImpl]   Region: root
19:02:24.871 INFO [FeaturesServiceImpl]     Bundles to uninstall:
19:02:24.871 INFO [FeaturesServiceImpl]       org.foo.foo-app/1.0.0.SNAPSHOT
19:02:24.871 INFO [FeaturesServiceImpl] Stopping bundles:
19:02:24.871 INFO [FeaturesServiceImpl]   org.foo.foo-app/1.0.0.SNAPSHOT
19:02:24.873 INFO [AppComponent] Stopped
19:02:24.874 INFO [CommandExtension] Unregistering commands for bundle org.foo.foo-app/1.0.0.SNAPSHOT
19:02:24.876 INFO [FeaturesServiceImpl] Uninstalling bundles:
19:02:24.877 INFO [FeaturesServiceImpl]   org.foo.foo-app/1.0.0.SNAPSHOT
19:02:24.881 INFO [FeaturesServiceImpl] Refreshing bundles:
19:02:24.881 INFO [FeaturesServiceImpl]     org.foo.foo-app/1.0.0.SNAPSHOT (Bundle will be uninstalled)
19:02:24.883 INFO [FeaturesServiceImpl] Done.
19:02:24.884 INFO [ApplicationManager] Application org.foo.app has been uninstalled
19:02:24.920 INFO [FeaturesServiceImpl] Adding features: foo-app/[1.0.0.SNAPSHOT,1.0.0.SNAPSHOT]
19:02:25.486 INFO [FeaturesServiceImpl] Changes to perform:
19:02:25.486 INFO [FeaturesServiceImpl]   Region: root
19:02:25.487 INFO [FeaturesServiceImpl]     Bundles to install:
19:02:25.487 INFO [FeaturesServiceImpl]       mvn:org.foo/foo-app/1.0-SNAPSHOT
19:02:25.487 INFO [FeaturesServiceImpl] Installing bundles:
19:02:25.488 INFO [FeaturesServiceImpl]   mvn:org.foo/foo-app/1.0-SNAPSHOT
19:02:25.504 INFO [FeaturesServiceImpl] Starting bundles:
19:02:25.504 INFO [FeaturesServiceImpl]   org.foo.foo-app/1.0.0.SNAPSHOT
19:02:25.511 INFO [AppComponent] Started
19:02:25.513 INFO [CommandExtension] Registering commands for bundle org.foo.foo-app/1.0.0.SNAPSHOT
19:02:25.514 INFO [FeaturesServiceImpl] Done.
19:02:25.515 INFO [ApplicationManager] Application org.foo.app has been activated
19:02:25.764 INFO [AppComponent] Reconfigured
19:03:41.198 INFO [EventAdminConfigurationNotifier] Sending Event Admin notification (configuration successful) to org/ops4j/pax/logging/Configuration
```

Figura 3-42 Comando para verificar la instalación de la aplicación en ONOS

```
Print result
out from AP1:  0 140.26 10.51 95.43 0
out from AP2:  108.85 0 0 0 158.95
out from AP3:  0 0 0 0 35.99
out from AP4:  124.96 0 7.42 0 0.19
out from AP5:  0 67.84 0.01 0 0

showPolicy
from AP1 go to AP2
from AP2 go to AP5
from AP3 go to AP5
from AP4 go to AP1
from AP5 go to AP2
la mejor ruta es la opcion 2 de las que se pueden apreciar al ingrsear el siguiente comando
paths src  dts
controlador@root > paths of:000000acfec00001 of:000000acfec00005
of:000000acfec00001/4-of:000000acfec00004/2==>of:000000acfec00004/3-of:000000acfec00005/3; cost=2.0
of:000000acfec00001/2-of:000000acfec00002/2==>of:000000acfec00002/4-of:000000acfec00005/2; cost=2.0
of:000000acfec00001/3-of:000000acfec00003/2==>of:000000acfec00003/5-of:000000acfec00005/4; cost=2.0
```

A

```
Print result
out from AP1:  0 478.57 478.06 479.54 0
out from AP2:  431.59 0 478.06 0 531.22
out from AP3:  431.58 478.36 0 478.22 531.2
out from AP4:  431.59 0 478.06 0 531.2
out from AP5:  0 477.91 479.11 478.07 0

showPolicy
from AP1 go to AP4
from AP2 go to AP5
from AP3 go to AP5
from AP4 go to AP5
from AP5 go to AP3
la mejor ruta es la opcion 1 de las que se pueden apreciar al ingrsear el siguiente comando
paths src  dts
controlador@root > paths of:000000acfec00001 of:000000acfec00005
of:000000acfec00001/4-of:000000acfec00004/2==>of:000000acfec00004/3-of:000000acfec00005/3; cost=2.0
of:000000acfec00001/2-of:000000acfec00002/2==>of:000000acfec00002/4-of:000000acfec00005/2; cost=2.0
of:000000acfec00001/3-of:000000acfec00003/2==>of:000000acfec00003/5-of:000000acfec00005/4; cost=2.0
controlador@root >
```

B

Figura 3-43 Resultados del Escenario 1 con el algoritmo arrojado por el sistema en el CLI de ONOS

Ahora bien, finalmente se realiza una modificación con el algoritmo corriendo, se dió de baja los dos enlaces, para llegar al escenario 4 y así mirar el resultado del algoritmo cuando se cambia la red mientras está en funcionamiento. En la Figura 3-44A se observa el cambio que tiene el sistema cuando se inhabilita el enlace, en este caso toma la ruta que va del AP1 al AP3 y de este al AP5, en la segunda prueba realizada al escenario se obtiene lo mostrado en la Figura 3-44B, donde la cantidad de bytes aumenta de forma considerable debido a que se ha aumentado el tiempo de transmisión en la simulación; sin embargo, la ruta sigue siendo la misma del AP1 al AP3 y al AP5.

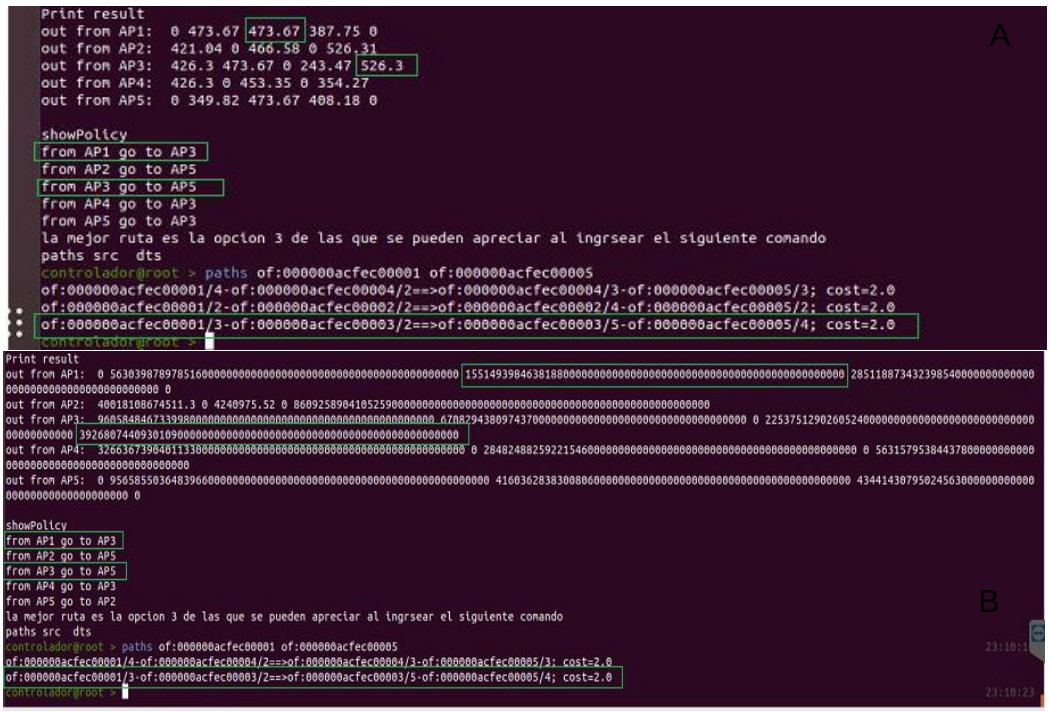

Figura 3-44 Resultados del Escenario 2 con el algoritmo arrojado por el sistema en el CLI de ONOS

Una vez terminada la fase de implementación y ejecución se da por cumplida y finalizada la metodología de simulación y la fase de pruebas de la metodóloga en cascada; por consiguiente, siguiendo con el orden de las fases, en el capítulo siguiente se presentará la fase final que es la de evaluación.

4. ANÁLISIS DE RESULTADOS

En el desarrollo de este capítulo se presentan los resultados obtenidos de las simulaciones realizadas anteriormente y se realizara el respectivo análisis, finalizando con la comparación de los escenarios propuestos en el capítulo anterior y de esa forma obtener la conclusión con base a lo arrojado por el sistema. La presentación de los resultados se hará en función del tiempo de simulación, esto puesto que se quiere observar cómo se comporta la red cuando todos sus dispositivos se activan durante un periodo de tiempo determinado y teniendo en cuentas las distintas tasas de transmisión utilizadas para las pruebas.

4.1. Resultados

Antes de presentar los resultados obtenidos en la Tabla 4-1 se muestra un resumen de los escenarios de simulación, así como los casos que corresponden a cada uno de ellos; adicional a ello también se tiene los subcasos de simulación, los cuales son las variaciones en las tasas de transmisión y al final de la tabla se muestran los tiempo de transmisión tenidos en cuenta.

Tabla 4-1 Resumen de los casos y subcasos de simulación

<table>
<tr><th colspan="9">Topología de red SD-IoT</th></tr>
<tr><th colspan="9">Casos de Simulación</th></tr>
<tr><th colspan="5">Escenario 1</th><th colspan="4">Escenario 2</th></tr>
<tr><th colspan="3">Caso 1</th><th colspan="2">Caso 2</th><th colspan="2">Caso 3</th><th colspan="2">Caso 4</th></tr>
<tr><td colspan="3">Sin el algoritmo Q-Learning</td><td colspan="2">Con el algoritmo Q-Learning</td><td colspan="2">Sin el algoritmo Q-Learning</td><td colspan="2">Con el algoritmo Q-Learning</td></tr>
<tr><th colspan="9">Subcasos de Simulación</th></tr>
<tr><th colspan="3">5,5 Mbps</th><th colspan="3">36 Mbps</th><th colspan="3">54 Mbps</th></tr>
<tr><td>10 min</td><td>30 min</td><td>1 h</td><td>10 min</td><td>30 min</td><td>1 h</td><td>10 min</td><td>30 min</td><td>1 h</td></tr>
</table>

3 h	6 h		3 h	6 h		3 h	6 h	

En las Tabla 4-2 y Tabla 4-3, se muestra los resultados de las simulaciones con las diferentes variaciones de las tasas de transmisión, sin embargo se encontró que la variación del ancho de banda de los enlaces afectaba entre un 0,6% y un 1.1% en la perdida de paquetes y en un 1% en el retardo promedio; por tanto, se trabajó con un ancho de banda entre los enlaces de los AP en 100 Mbps y se modificó las tasas de transmisión de los dispositivos inalámbricos.

Tabla 4-2 Resultado de la simulación variando tanto la tasa de Transmisión de las Sta como de los enlaces para el escenario 1

Protocolo	Capacidad entre enlaces de AP en Mbps	Delay Promedio en seg	Porcentaje de paquetes perdidos
802.11b	15	6,83	69,77
	50	6,84	68,99
	100	6,84	68,52
802.11g	15	3,52	67,8
	50	3,52	67,86
	100	3,53	67,12
802.11n	15	1,36	61,45
	50	1,35	61,46
	100	1,36	61,65

Tabla 4-3 Resultado de la simulación variando tanto la tasa de Transmisión de las Sta como de los enlaces para el escenario 2

Protocolo	Capacidad entre enlaces de AP en Mbps	Delay Promedio en seg	Porcentaje de paquetes perdidos
802.11b	15	6,9	71,25
	50	6,85	71,32
	100	6,88	71,42
802.11g	15	3,73	70,52
	50	3,73	70,33
	100	3,72	70,12
802.11n	15	1,62	61,45
	50	1,62	61,46
	100	1,63	61,65

- **Escenario 1 sin el algoritmo ML (Caso 1)**

En las Tabla 4-4 y Tabla 4-5 se observan los resultados obtenidos en el primer escenario de simulación para los diferentes tiempos de transmisión considerando las distintas tasas de transferencias.

Tabla 4-4 Pérdida de paquetes en el escenario 1 sin el algoritmo ML

5.5 Mbps	36 Mbps	54 Mbps	
Paquetes perdidos (%)	Paquetes perdidos (%)	Paquetes perdidos (%)	Tiempo en Segundos
68,52	67,21	61,05	600
69,31	67,24	61,18	1800
70,33	67,27	61,25	3600
71,19	67,36	61,32	10800
74,1	69,57	61,79	21600

Tabla 4-5 Promedio del escenario 1 sin el algoritmo ML

5.5 Mbps	36 Mbps	54 Mbps	
Retardo en (s)	Retardo en (s)	Retardo en (s)	Tiempo en Segundos
6,23	3,13	1,36	600
6,84	3,53	1,42	1800
7,17	3,79	1,48	3600
7,52	4,41	1,75	10800
7,94	5,06	1,94	21600

En las Figura 4-1 y Figura 4-2 se realiza la comparación de la pérdida de paquetes con respecto al tiempo de transmisión para cada una de las tasas y también el comportamiento del retardo promedio respecto al tiempo para el primer escenario de simulación sin el algoritmo ML.

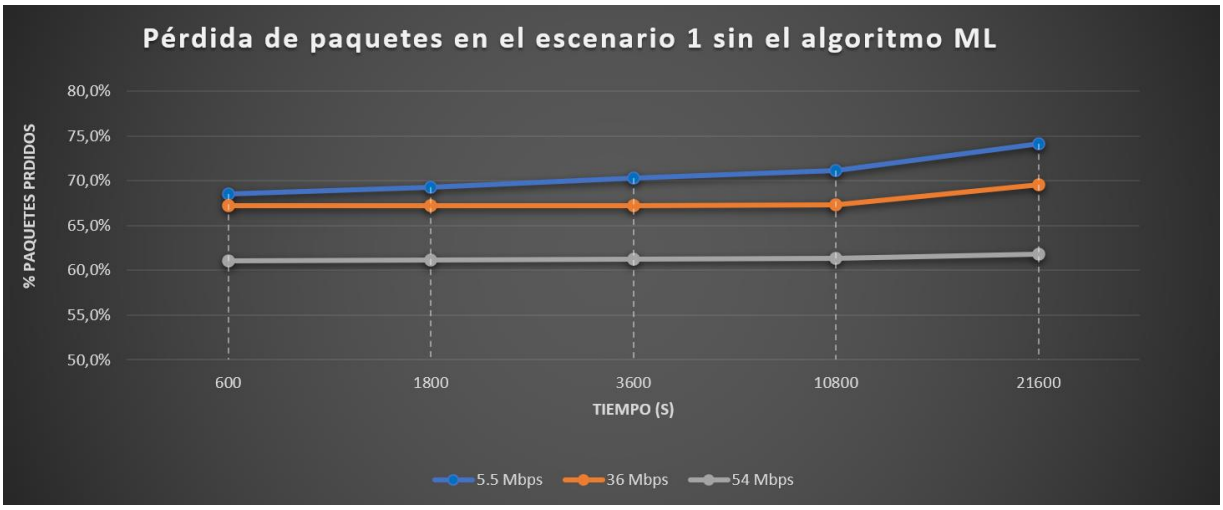

Figura 4-1 Pérdida de paquetes sin el algoritmo ML en el escenario 1 para velocidades de 5.5 Mbps, 36 Mbps y 54 Mbps

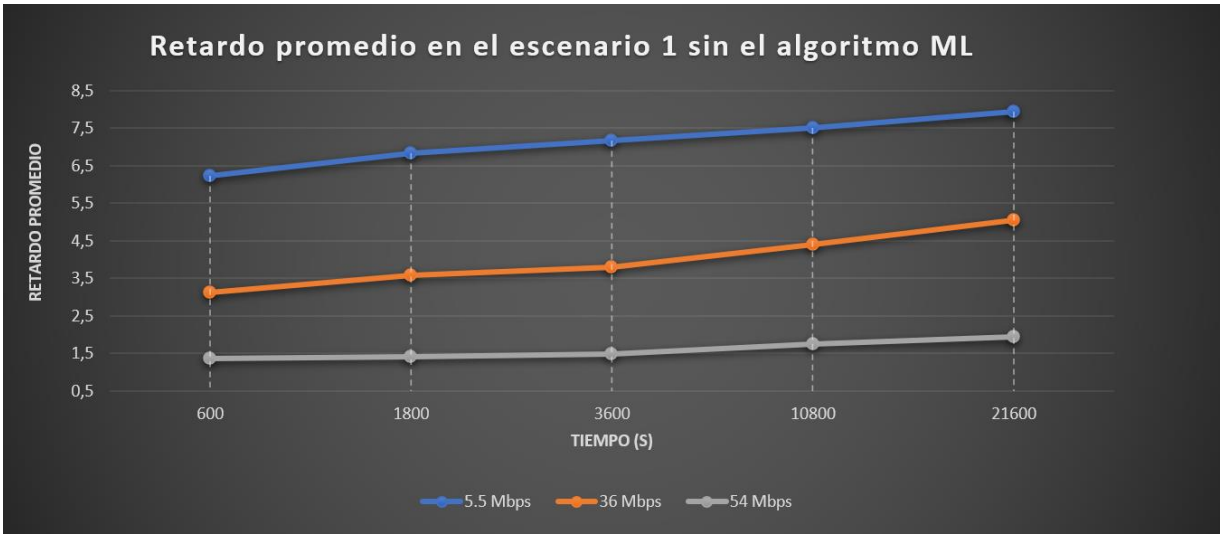

Figura 4-2 Retardo promedio sin el algoritmo ML en el escenario 1 para velocidades de 5.5 Mbps, 36 Mbps y 54 Mbps

En la Figura 4-1 se presentan los resultado de la pérdida de paquetes, se observa una variación entre el 2% y el 12% entre las distintas tasas de transmisión, donde la variación entre la tasa de 5.5 Mbps y la de 54 Mbps, se redujo aproximadamente un 20%, la Figura 4-2 muestra las variaciones que presenta el retardo promedio para las diferentes tasas de trasmisión, donde a diferencia del parámetro anterior la fluctuación va desde 56% hasta el 160%; notando que entre la tasa de 5.5 Mbps y la de 54 Mbps una reducción de más de 3.09 puntos, aproximadamente un 300%.

Otro aspecto que se observa en las figuras es que el tiempo de transmisión es directamente proporcional a la variación de los parámetros, es decir, a medida que se aumenta el tiempo de transmisión la perdida de paquetes tiende a aumentar en un 1% para la tasa de 5.5Mbps, un 3% para la de 36 Mbps y un 0.01% para la tasa de 54 Mbps y el retardo promedio varia 9% para la tasa de 5.5Mbps, un 0.07% para la de 36 Mbps y un 0.1% para la tasa de 54 Mbps.

- **Escenario 1 con el algoritmo ML (Caso 2)**

En este escenario se tienen en cuenta las mismas condiciones de configuración y tráfico del caso anterior, pero se implementa el algoritmo de simulación diseñado, el cual arrojo los resultados obtenidos en la Tabla 4-6 y Tabla 4-7.

Tabla 4-6 Pérdida de paquetes en el escenario 1 con el algoritmo ML

5.5 Mbps	**36 Mbps**	**54 Mbps**	
Paquetes perdidos (%)	Paquetes perdidos (%)	**Paquetes perdidos (%)**	**Tiempo en Segundos**
76,35	73,53	64,32	600
77,21	75,37	64,99	1800
77,77	76,52	65,27	3600
77,82	76,98	66,25	10800
79,24	78,12	66,99	21600

Tabla 4-7 Retardo promedio del escenario 1 con el algoritmo ML

5.5 Mbps	**36 Mbps**	**54 Mbps**	
Retardo en (s)	Retardo en (s)	**Retardo en (s)**	**Tiempo en Segundos**
2,45	1,29	0,41	600
2,5	1,51	0,43	1800
2,91	1,73	0,47	3600
3,12	1,81	0,5	10800
3,22	1,83	0,6	21600

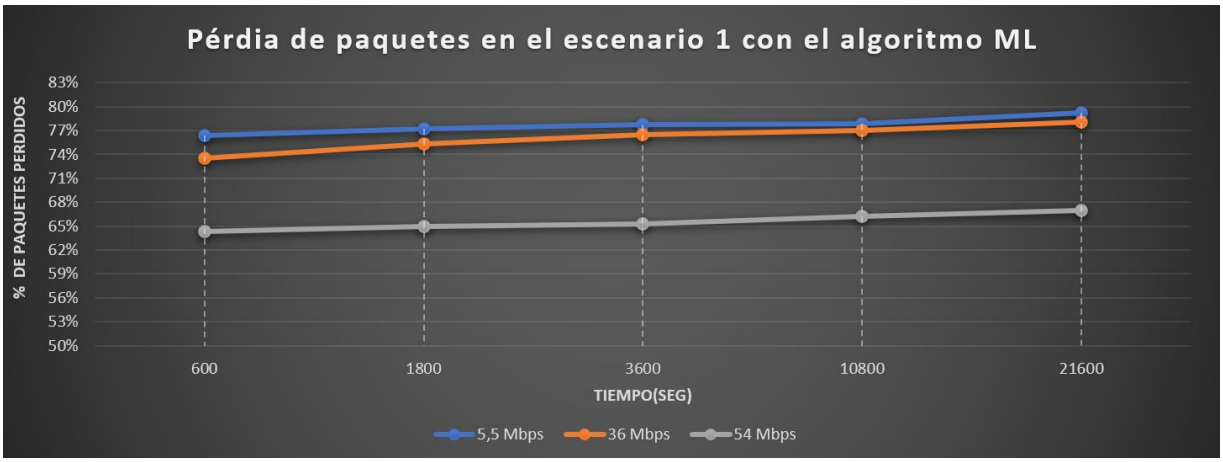

Figura 4-3 Pérdida de paquetes con el algoritmo ML en el escenario 1 para velocidades de 5.5 Mbps, 36Mbps y 54Mbps

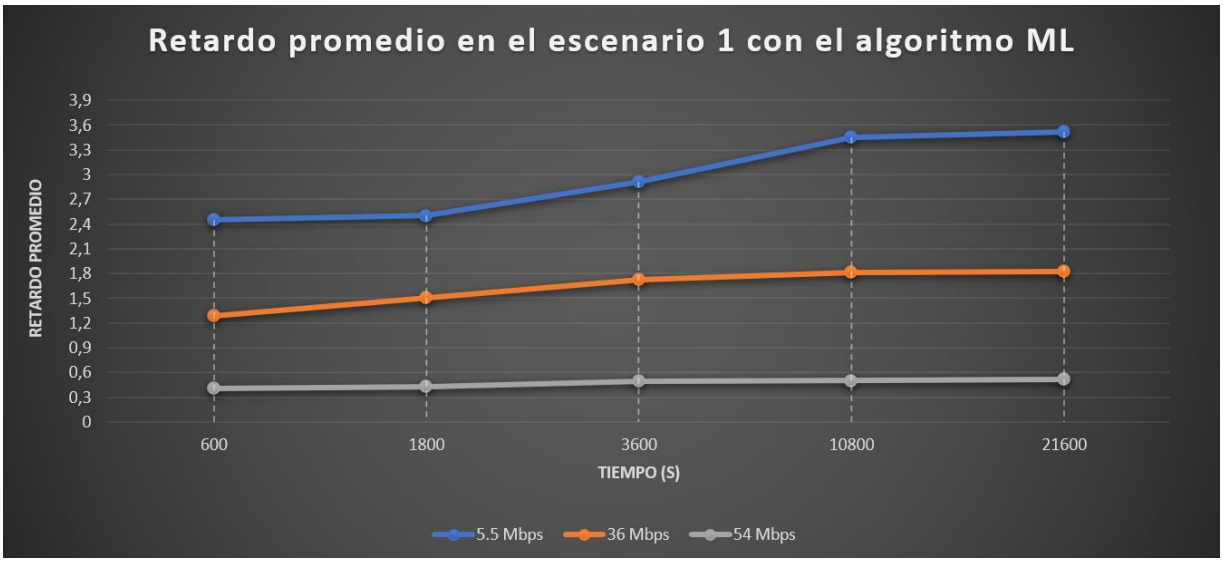

Figura 4-4 Retardo promedio con el algoritmo ML en el escenario 1 para velocidades de 5.5 Mbps, 36 Mbps y 54 Mbps

En las Figura 4-3 y Figura 4-4, se presentan los resultados con la implementación del algoritmo Q-Learning, donde la primera corresponde a la pérdida de paquetes y en la cual se observa que las variaciones entre tiempos de transmisión están alrededor de un 1% para cada una de las tasas de transmisión; sin embargo, la diferencia de la perdida de paquetes entre la tasa más baja de 5,5 Mbps y la más alta velocidad que es la de 54 Mbps es de un 18% aproximadamente; mientras que la variación presentada entre las tasas de 5.5 Mbps y 36 Mbps solo fue de un de un 1% a un 3%, dejando estas dos tasa casi en el mismo nivel de pérdida de paquetes. En la segunda figura, se observa la variación que presenta el retardo promedio entre los distintos

tiempos de trasmisión, la cual es de aproximadamente un 4% para las tasas de 36 Mbps y un 1% para la tasa de 54 Mbps, en cuanto a la tasa de 5.5 Mbps ésta presento una variación de hasta un 18% para los tiempo de transmisión; por otra parte, se observa un variación de aproximadamente 4.36 puntos o 436% de la tasa de 5,5 Mbps a la de 54 Mbps, dejando en evidencia que el mejor desempeño se obtuvo con la tasa más alta.

- **Escenario 2 sin el algoritmo ML (Caso 3)**

Para iniciar con la simulaciones del escenario 2, se debe recordar que para este caso se dió de baja dos de los enlaces que comunican a los puntos de acceso, en las Tabla 4-8 y Tabla 4-9, se muestran los datos obtenidos en esta simulación; adicional a ello, se debe tener en cuenta que al igual que en el escenario 1, también se consideraron tasas diferentes de transmisión para los dispositivos inalámbricos y se fijó el ancho de banda de los enlaces entre puntos de acceso.

Tabla 4-8 Pérdida de paquetes en el escenario 2 sin el algoritmo ML

5.5 Mbps	36 Mbps	54 Mbps	
Paquetes perdidos (%)	Paquetes perdidos (%)	Paquetes perdidos (%)	Tiempo en Segundos
71,04	70,1	61,18	600
71,21	70,12	61,45	1800
71,25	70,33	61,46	3600
71,32	70,52	61,65	10800
71,42	70,55	61,85	21600

Tabla 4-9 Retardo promedio del escenario 2 sin el algoritmo ML

5.5 Mbps	36 Mbps	54 Mbps	
Retardo en (s)	Retardo en (s)	Retardo en (s)	Tiempo en Segundos
6,85	3,25	1,63	600
6,91	3,72	1,67	1800
8,11	3,93	1,95	3600
8,21	5,06	2,05	10800
8,9	5,22	2,08	21600

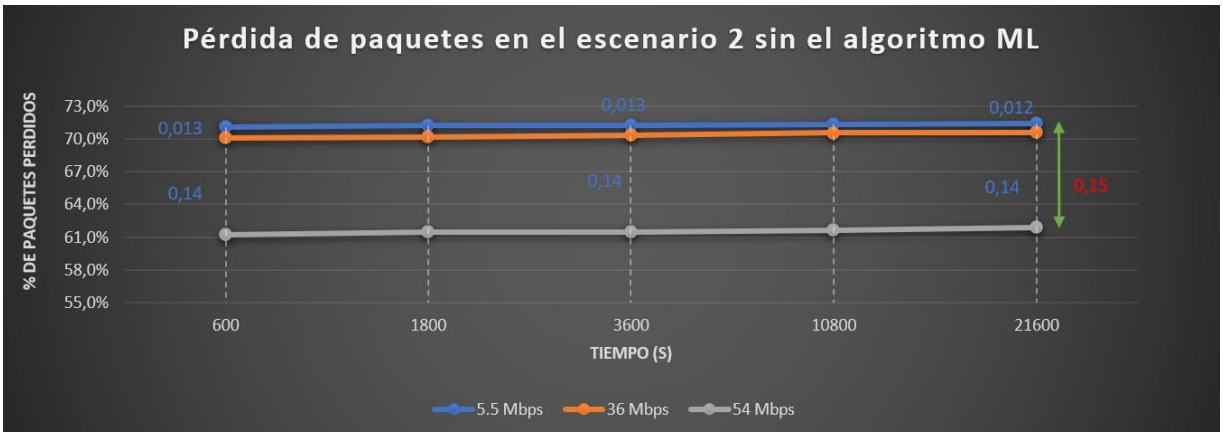

Figura 4-5 Pérdida de paquetes sin el algoritmo ML en el escenario 2 para velocidades de 5.5 Mbps, 36 Mbps y 54 Mbps

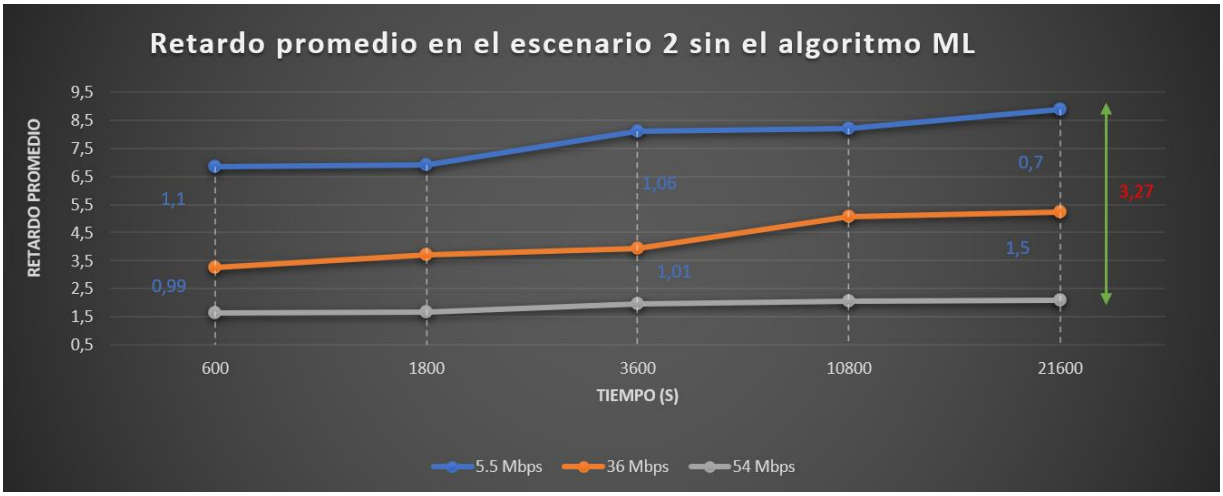

Figura 4-6 Retardo promedio sin el algoritmo ML en el escenario 2 para velocidades de 5.5 Mbps, 36 Mbps y 54 Mbps

El resultados del segundo escenario se muestra en la Figura 4-5 donde las pérdidas de paquetes presentadas en las diferentes tasas de transmisión varía aproximadamente un 1.3% entre las tasas de 5.5 Mbps y 36 Mbps, mientras que la variación entre la tasa de 36 Mbps y la de 54 Mbps esta alrededor de un 140%, al observar la variación de la tasa de 5.5 Mbps y la de 54 Mbps es de 150%; en cuanto al retardo promedio como se observa en la Figura 4-6 cambia entre aproximadamente un 100% para los periodos de transmisión más cortos y de un 70% para los tiempo de transmisión de 3 horas y 6 horas.

En cuanto a la relación del tiempo de trasmisión con respecto al incremento del retardo promedio, se tiene que está directamente relacionado, puesto que a medida que se aumenta el tiempo de transmisión este parámetro aumenta en un 23% para la tasa de 5.5Mbps y para la de 36 Mbps, mientras que para la tasa de 54 Mbps el retardo aumento aproximadamente un 2% a medida que aumenta el tiempo de transmisión; finalmente al realizar el cálculo de la reducción del retardo de la tasa de 5.5 Mbps a la 54 Mbps se obtuvo que este parámetro se reduce en 3.27 puntos lo que sería aproximadamente un 327% al usar la tasa de transmisión más alta.

- **Escenario 2 con el algoritmo ML (Caso 4)**

Considerando las mismas configuración del caso anterior, con los dos enlaces dados de baja y las mismas configuración de tráfico, se realiza la implementación del mecanismo de control, arrojando como resultado los datos en Tabla 4-10 y Tabla 4-11.

Tabla 4-10 Pérdida de paquetes en el escenario 2 con el algoritmo ML

5.5 Mbps	**36 Mbps**	**54 Mbps**	
Paquetes perdidos (%)	Paquetes perdidos (%)	**Paquetes perdidos (%)**	**Tiempo en Segundos**
79,32	74,53	68,19	600
81,05	75,37	70,82	1800
81,91	76,52	71,42	3600
82,01	77,24	73,6	10800
82,73	77,57	74,02	21600

Tabla 4-11 Retardo promedio del escenario 2 con el algoritmo ML

5.5 Mbps	**36 Mbps**	**54 Mbps**	
Retardo en (s)	Retardo en (s)	**Retardo en (s)**	**Tiempo en Segundos**
6,01	3,04	0,72	600
6,11	3,29	0,73	1800
6,45	3,52	0,83	3600
7,1	3,91	0,83	10800
7,15	4,02	0,92	21600

En la Figura 4-7 se muestra el resultado de implementar el algoritmo en el escenario 2 que consiste en inhabilitar dos de los enlaces entre puntos de acceso, en este caso la variación de pérdida de paquetes entre las dos tasas de transmisión más altas que son la de 36 Mbps y la 54 Mbps oscila entre el 9% y el 6%; la variación del segundo parámetro se observa en la Figura 4-8, donde la diferencia entre las tasas es muy notable y se puede ver claramente en la gráfica que se presenta una mejora por encima de un 200% para cada uno de los tiempos de transmisión.

En cuanto a cómo se comportan los parámetros en los distintos tiempo de trasmisión para cada tasa, se encontró que la pérdida de paquetes tanto para 5.5 Mbps como para 36 Mbps presenta una variación del 1% y la para la de mayor velocidad que es la de 54 Mbps hay una variación del alrededor de un 2%; en cuando al retardo promedio este arrojo una variación de 1% para la tasa de 36 Mbps y de 0.8% para la tasa de 54 Mbps, en cuanto a la velocidad más baja que es la de 5,5 Mbps, este sufrió una mayor variación con respecto a los mayores tiempos de trasmisión donde se tuvo un incremento de 3%.

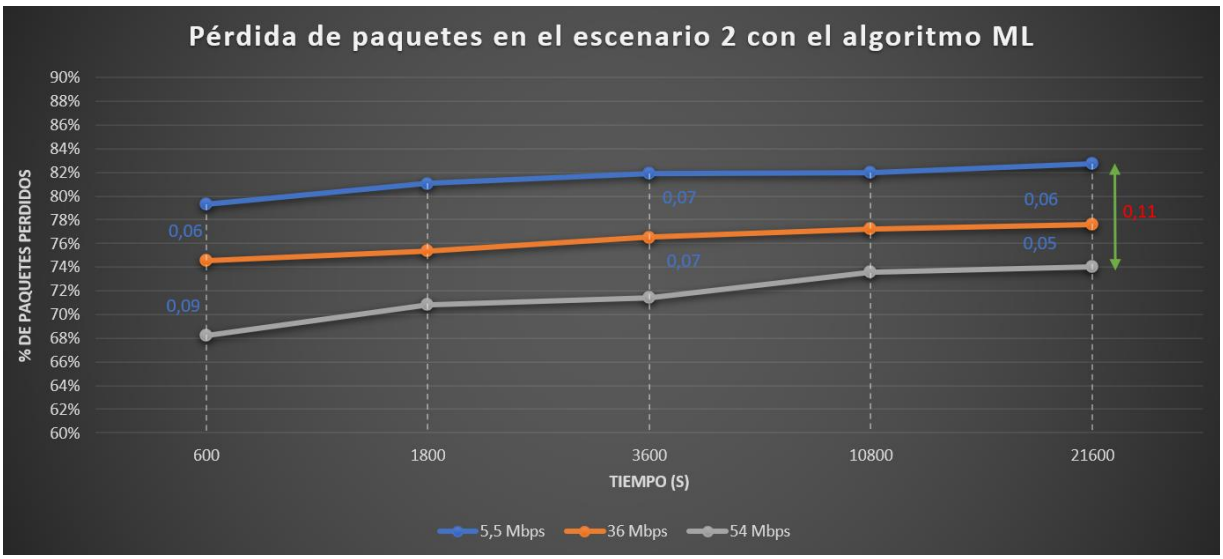

Figura 4-7 Pérdida de paquetes con el algoritmo ML en el escenario 2 para velocidades de 5.5 Mbps, 36Mbps y 54Mbps

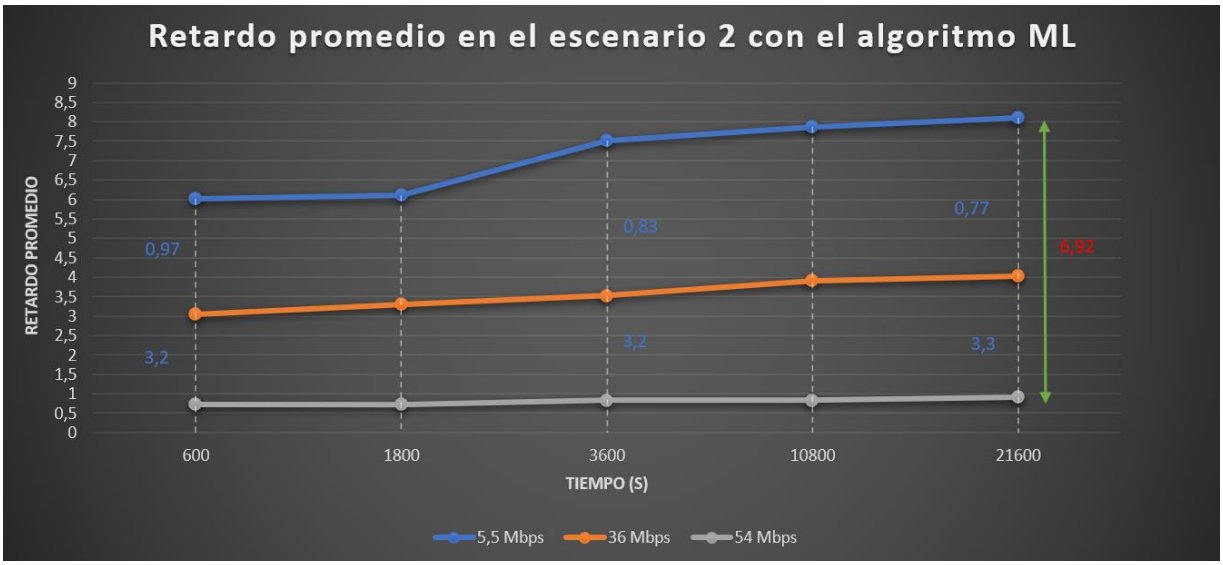

Figura 4-8 Retardo promedio con el algoritmo ML en el escenario 2 para velocidades de 5.5 Mbps, 36 Mbps y 54 Mbps

La tasa de transmisión que mejor se comportan en estos espacios y tipo de redes es la más alta, en este caso la tasa de 54 Mbps que corresponde al protocolo 802.11n en la banda de los 2.4 GHz, ya que fue la que mejores resultados arrojo para los dos parámetros de desempeño y para cada uno de los escenarios y sus respectivas variaciones.

Al implementar el algoritmo de control de tráfico en la red, se pudo observar un incremento del 8% al 12% para la tasa de 36Mbps y de un 5% al 7% para la tasa de 54 Mbps, esto para las condiciones planteadas en el escenario 1, esta variación se puede observar en las Figura 4-9 y Figura 4-10; en cuanto al retardo de extremo a extremo, en el escenario 1 se presenta una mejora al implementar el algoritmo de control de tráfico, ya que la variación presentada para la tasa de 36 Mbps fue de un 140% a un 176% y para la tasa de 54 Mbps la mejora de este parámetro estuvo por encima del 200%, relación que se puede ver en las Figura 4-13 y Figura 4-14.

Al aplicar el algoritmo en el escenario 2, se encontró un incremento en el parámetro de pérdida de paquetes entre un 5% y 9% para la tasa de 36 Mbps y de un 10% a un 16% para la tasa de 54 Mbps, relación que se puede observar en las Figura 4-11 y Figura 4-12., para el retardo promedio al aplicar el mecanismo de control se logró una reducción de 6% a un 29% para la tasa de 36 Mbps y de más de un 100% para la tasa de 54 Mbps, esta comparación se observa en las Figura 4-15 y Figura 4-16.

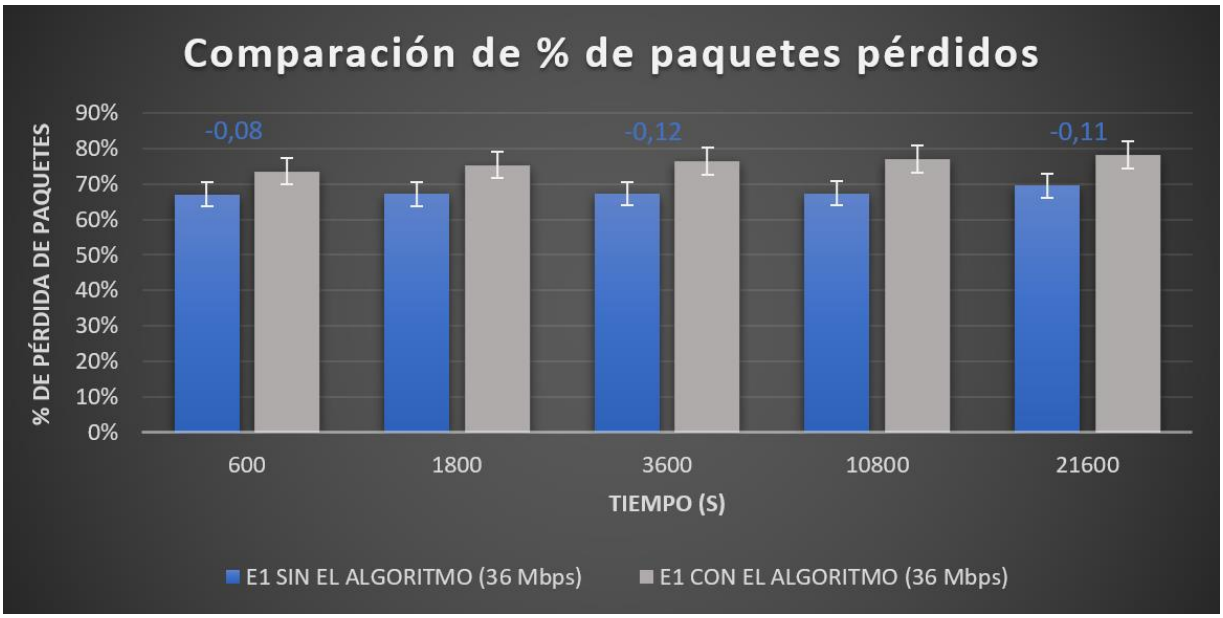

Figura 4-9 Comparación de resultados de pérdida de paquetes del escenario 1 con y sin el algoritmo ML para la tasa de 36 Mbps

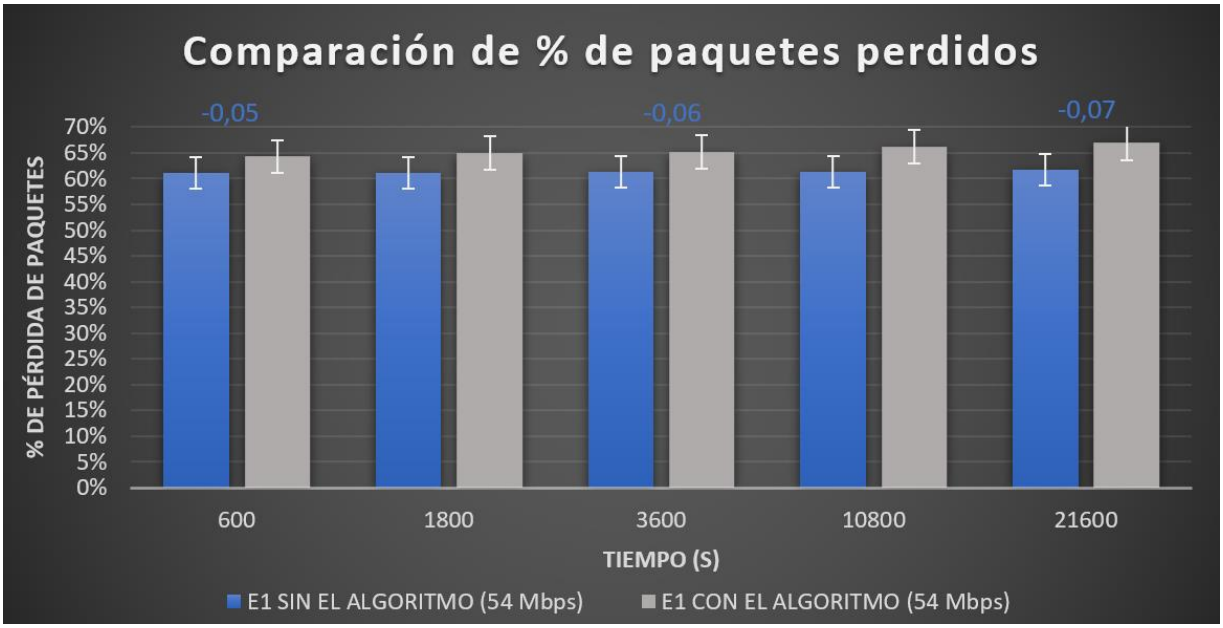

Figura 4-10 Comparación de resultados de pérdida de paquetes del escenario 1 con y sin el algoritmo ML para la tasa de 54 Mbps

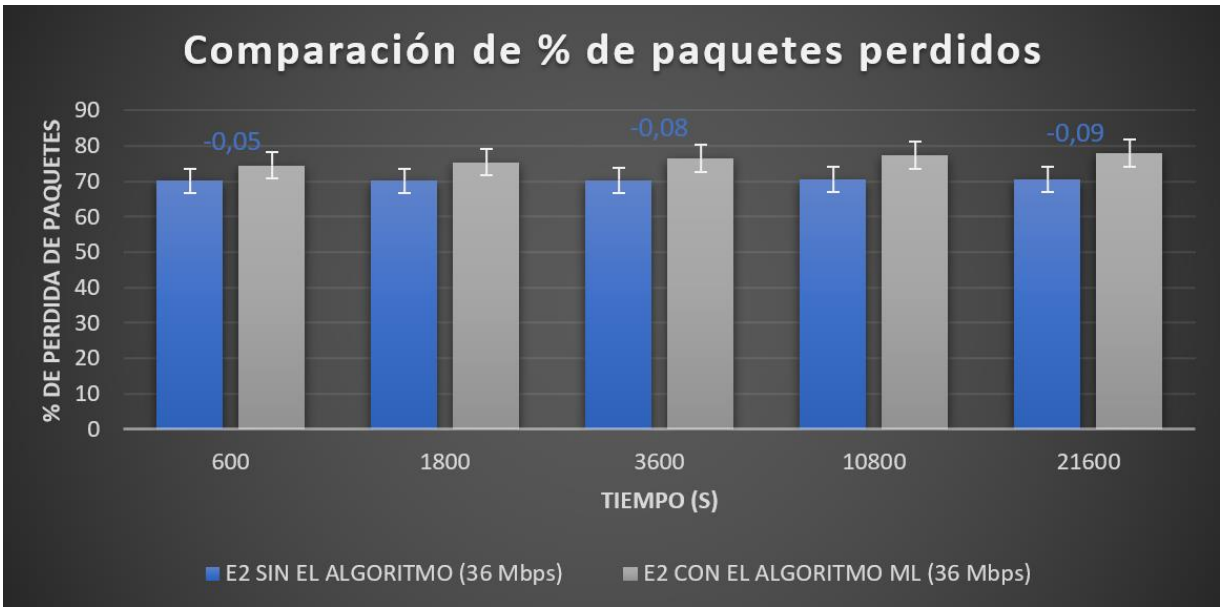

Figura 4-11 Comparación de resultados de pérdida de paquetes del escenario 2 con y sin el algoritmo ML para la tasa de 36 Mbps

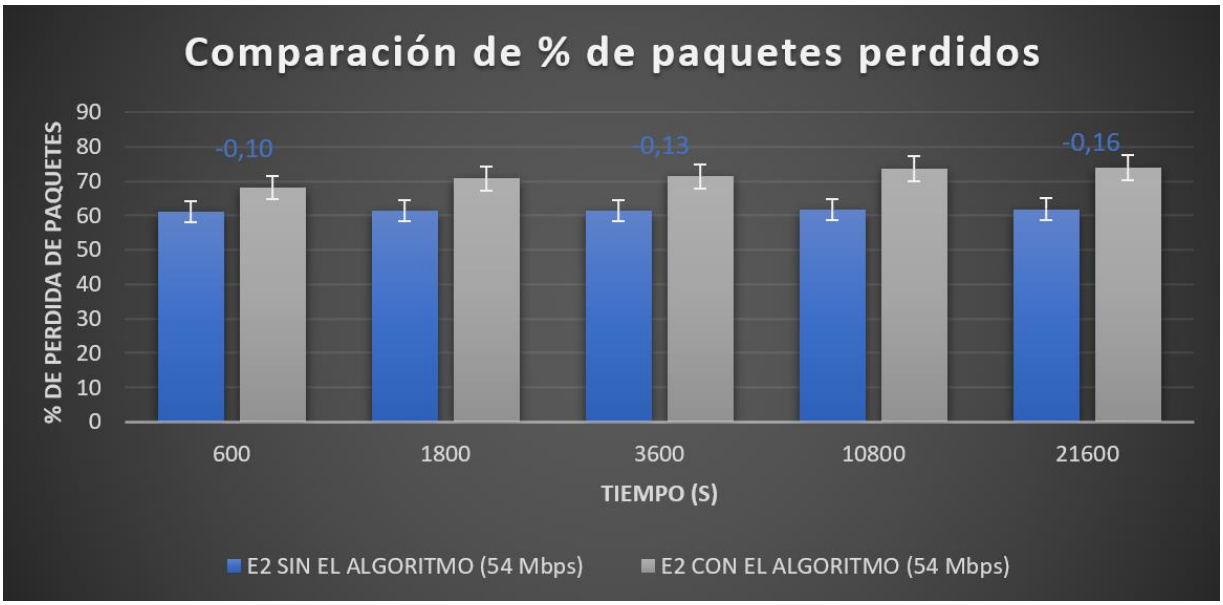

Figura 4-12 Comparación de resultados de pérdida de paquetes del escenario 2 con y sin el algoritmo ML para la tasa de 54 Mbps

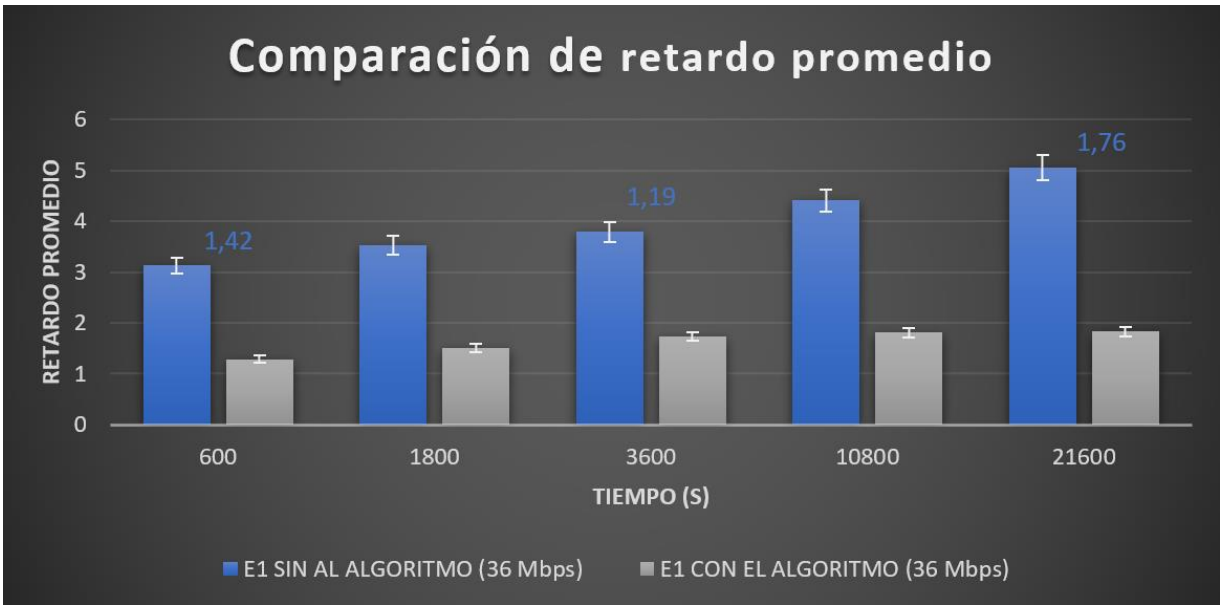

Figura 4-13 Comparación de resultados de retardo promedio del escenario 1 con y sin el algoritmo ML para la tasa de 36 Mbps

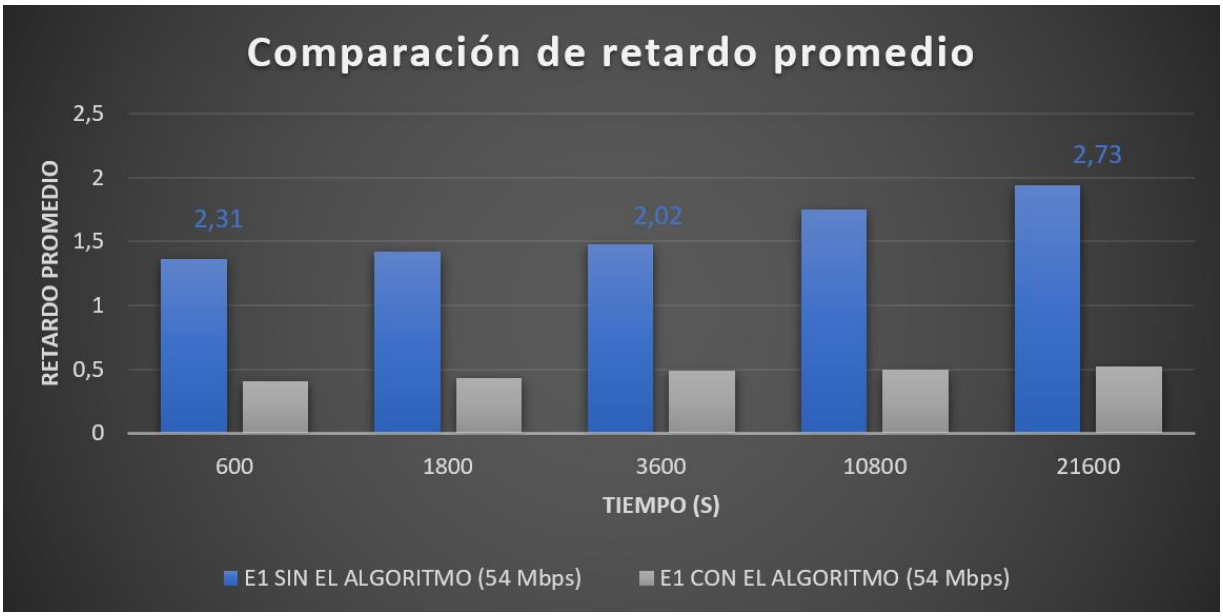

Figura 4-14 Comparación de resultados de retardo promedio del escenario 1 con y sin el algoritmo ML para la tasa de 54 Mbps

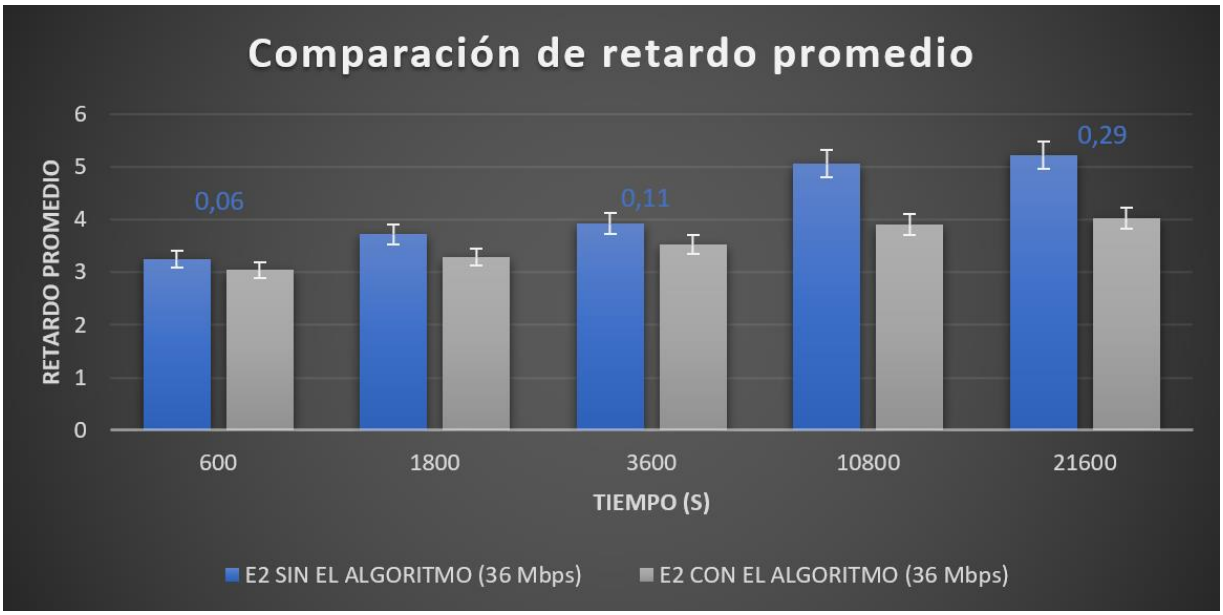

Figura 4-15 Comparación de resultados de retardo promedio del escenario 2 con y sin el algoritmo ML para la tasa de 36 Mbps

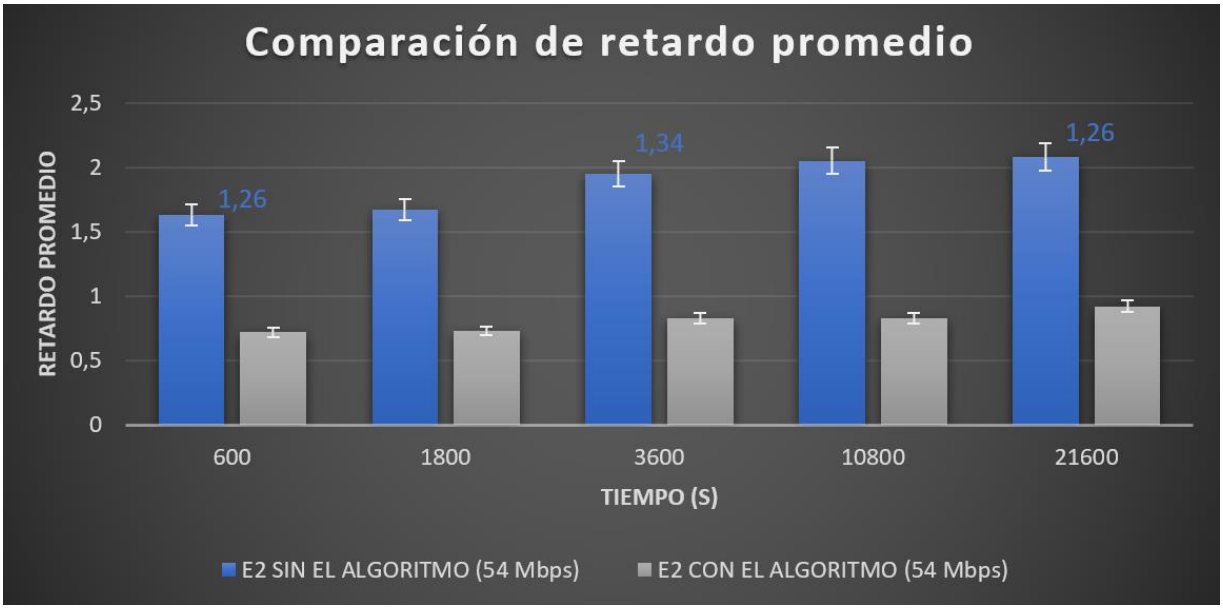

Figura 4-16 Comparación de resultados de retardo promedio del escenario 1 con y sin el algoritmo ML para la tasa de 54 Mbps

La pérdida de paquetes arrojada en cada una de las simulaciones fue mucho más alta de lo esperado; en el escenario 1 presentó un porcentaje del 60% para la tasa de 54 Mbps y de un 65% al 75% para las tasas de transmisión de 5.5 Mbps y 36 Mbps, esto debido a la cantidad de tráfico que se encontraba circulando por la red, puesto que lo

que buscaba era generar una sobrecarga o congestión en la red para ver el compartimiento del dispositivo seleccionado. En cuanto al retardo promedio obtenido en este escenario, como era de esperarse al tener sobrecargada de trafico la red se tiene que para la menor tasa de transmisión que es de 5.5 Mbps este parámetro está por encima de los 5.5 segundos para los 10 minutos de envío de información y por arriba de 7.5 para un periodo de transmisión de 6 horas, para la tasa de 36 Mbps esta por arriba de los 2.5 segundos para un menor tiempo de transmisión y por encima de 4.5 segundos para el mayor tiempo de transmisión, y el mejor resultado obtenido fue para la tasa de 54 Mbps, la cual obtuvo un retardo promedio por debajo de 1.5 segundos para los tiempos de transmisión más cortos y por arriba de 1.5 segundos para un tiempo de 3 horas y 6 horas.

Al implementar el mecanismo de control de tráfico se encontró para la tasa de 54 Mbps fueron de aproximadamente un 65% para los tiempos de transmisión más cortos que son de 10 min, 30 min y 1 hora, en cuando a los tiempos de 3 horas y 6 horas se obtuvo un promedio del 68% de pérdida de paquetes; en cuanto a las tasa de transmisión más bajas, las perdidas estuvieron por encima del 74%. Aunque lo que se buscaba era que al aplicar el mecanismo de control diseñado se lograra reducir la perdida de paquetes al controlar el flujo del dispositivo seleccionado para evitar la congestión o sobrecarga que presenta la red, lo resultados no fueron los esperados, debido al funcionamiento del sistema, donde cada nodo debe esperar a que el controlador le indique como debe tratar el flujo de información que le llega del dispositivo seleccionado; en otras palabras, cada nodo debe tener los paquetes en espera hasta que el controlador realice todo el proceso del algoritmo ML para así conocer cuáles son los saltos que deben dar los paquetes y esto hace que TTL se venza y por tanto se descarten aumentando el porcentaje de pérdida de paquetes. En cuanto al comportamiento del parámetro de retardo de extremo a extremo, este mejoró considerablemente al implementar el mecanismo de control de tráfico diseñado, logrando un retado de un 0.3 segundos para la tasa de 54 Mbps con un incremento de un 0.2% para los mayores tiempos de transmisión. Para la tasa de 36 Mbps se obtuvo que el retardo promedio estuvo alrededor de 1.2 segundos con un incremento de 4% a mayor tiempo de transmisión y para la tasa más baja de 5.5 Mbps el retardo estuvo en 2.4 segundos para los tiempo de 10 y 30 minutos con un incremento de un 18% para un tiempo de 6 horas; con esto se tiene que el mecanismo de ML está realizando su trabajo al seleccionar los caminos menos congestionados, sin embargo, a medida

que aumenta el tiempo de circulación este parámetro va aumentar, esto dado al comportamiento de todos los dispositivo que envían información por la red.

En el escenario 2 al dar de baja a dos de los enlaces que conectan a los puntos de acceso, lo que se buscaba era mirar si el comportamiento de este parámetro de pérdida empeoraba con la misma cantidad de tráfico; de esto se obtuvo que para promedio de un 61% para la tasa de 54 Mbps y de un 70% a un 72% para las tasas de 5.5 Mbps y 36 Mbps, dejando en evidencia lo que se esperaba que al disminuir los caminos de circulación la congestión o saturación aumentó y por ende la pérdida de paquetes también aumento. Al tener menos caminos de circulación para que el envío de información se obtuvo una mayor variación con respecto a los tiempos de transmisión, en el caso de la tasa de 5.5 Mbps se tiene que para el tiempo de 10 min y 30 min este parámetro estuvo alrededor de 6.5 segundos y para los tiempos de 1 hora en adelante la pérdida de paquetes aumento de un 5% aproximadamente; para la tasa de 36 Mbps se presentó una variación de un 8% del tiempo de 30 min al de 1 hora; finalmente, para la tasa de 54 Mbps el incremento con respecto a los tiempo de transmisión solo fue de un 1% a un 2%.

Cuando se activó la aplicación con el algoritmo de control diseñado se encontró que a diferencia del anterior en este caso para la tasa de 54 Mbps se presentaron variaciones de un 0.8% y un 3% entre los tiempos de transmisión, mientas que para las tasa de 5.5 Mbps y 36 Mbps solo se tuvo una variación de un 0.1% a un 1% para los tiempos de envío de información establecidos; sin embargo, al igual que en el caso anterior la implementación del algoritmo no mejoró la pérdida de paquetes por el contrario este parámetro paso de estar en un 61% a un 68%, esto dado que los nodos deben esperar que el controlador realice el análisis, aplique el mecanismo y luego le indique cual es el comportamiento que deben seguir los paquetes enviados por el dispositivo seleccionado. Para el parámetro del retardo de extremo a extremo para la tasa de 54 Mbps fue de 0.5 segundos y se mantuvo estable para todas las periodos de transmisión; sin embargo para la tasa de 36 Mbps y para la tasa de 5.5 Mbps se presentaron incrementos de un 1% a un 7% para los tiempos de transmisión, nuevamente se observa que con el mecanismo diseñado se reduce en más de un 100% para la tasa de 54 Mbps indicando que la lógica del algoritmo funciona de forma correcta.

A continuación se presenta en la Tabla 4-12 se muestra un comparativo y resumen de los resultados obtenidos, así como también las posibles causas de lo obtenido.

Tabla 4-12 Resumen de resultados obtenidos

	Pérdida de paquetes		Retardo promedio	
Escenario 1 sin el algoritmo	61% para 54 Mbps 67% para 36 Mbps 69% para 5.5 Mbps	En ambos escenarios se tuvo un incremento en el parámetro de perdida de paquetes al implementar el mecanismo de control diseñado, esto se debe a que los nodos deben esperar que el controlador haga todo el análisis de la red y tome la decisión sobre que camino deben seguir los paquetes del dispositivo seleccionado; todo el tiempo que toma realizar este proceso hace que el buffer se llene y los paquetes sobrepasen el TTL y se descarten.	Entre 1 y 2 seg para 54 Mbps Entre 3 y 5 seg para 36 Mbps Entre 6 y 7 seg para 5.5 Mbps	Al implementar el mecanismo de control en ambos escenario se redujo el retardo de transmisión, lo que indica que la lógica del algoritmo esta cumpliendo con su función que es seleccionar los caminos menos congestionado para realizar el envío de información.
Escenario 1 con el algoritmo	64% para 54 Mbps 76% para 36 Mbps 77% para 5.5 Mbps		Entre 0.1 y 0.5 seg para 54 Mbps Entre 1 y 2 seg para 36 Mbps Entre 2 y 4 seg para 5.5 Mbps	
Escenario 2 sin el algoritmo	61% para 54 Mbps 70% para 36 Mbps 71 % para 5.5 Mbps		Entre 1 y 2 seg para 54 Mbps Entre 3 y 5 seg para 36 Mbps Entre 6 y 8 seg para 5.5 Mbps	
Escenario 2 con el algoritmo	71% para 54 Mbps 77% para 36 Mbps 81% para 5.5 Mbps		Entre 0.5 y 1 seg para 54 Mbps Entre 3 y 4 seg para 36 Mbps Entre 6 y 7 seg para 5.5 Mbps	

Al finalizar la comparación de los parámetros de desempeño y teniendo en cuanto lo expuesto en la tabla anterior se considera lo siguiente:

- Los mejores resultados se presentaron para las tasas más altas.
- A menor tiempo de transmisión de los dispositivos la red se comporta de una mejor manera.
- El mecanismo diseñado logro reducir el retardo de extremo a extremo, no es una solución que se pueda aplicar a una red real, ya que dada la naturaleza del funcionamiento del controlador hace que las pérdida de paquetes aumente a tal punto que se pierde más de 75% de la información y por ende la información no llegará completa a su destino.

5. CONCLUSIONES Y TRABAJOS FUTUROS

5.1. CONCLUSIONES

- Realizar la incorporación de las redes SDN a las redes IoT reduce de forma significativa la complejidad en las arquitecturas implementadas hasta el momento y facilita la incorporación de nuevas funcionalidades.

- Los sistemas reforzados en la implementación de mecanismos de ML para el control de tráfico en rutas menos congestionadas es la mejor opción, debido a que éstos trabajan con la información actual de las redes y toman decisiones sobre la ejecución, haciendo que la migración estos escenarios simulados al mundo real sea más confiable.

- El algoritmo de control de tráfico implementado presentó una mejora de aproximadamente un 100% al implementarle el algoritmo de control al escenario 1 para la tasa de 36 Mbps y de un 200% para la de 54 Mbps en cuanto el retardo presentado y este mismo parámetro mejoró aproximadamente un 100% para la tasa de 54 Mbps al implementarle el algoritmo de control al escenario 2; dejando en evidencia que las mejores tasas para trabajar con estas redes son las más altas, en este caso la de 54 Mbps.

- La pérdida de paquetes sufrió un revés al implementarle el algoritmo, ya que al implementar el mecanismo de control al escenario 1 aumentó aproximadamente un 5% para la tasa de 54 Mbps y un 13% para la tasa de 36 Mbps, y al implementar el algoritmo en el escenario 2 aumentó aproximadamente de un 10% a un 16%; sin embargo, aunque el resultado obtenido no fue el esperado también se puede evidenciar que la tasa que obtuvo el mejor comportamiento fue la más alta, en este caso la de 54 Mbps y los mejores tiempos fueron los más cortos.

- La complejidad en la implementación del mecanismo así como los recursos del sistema hicieron que el periodo de análisis de la red por el controlador estuviera en alrededor de 1 segundo ocasionando el aumento en el retardo y la perdida de paquetes, ya que la cola se alcanza a saturar, puesto que debe esperar a que el mecanismo tome la información la pase por el algoritmo y obtenga una respuesta para indicarle al sistema cómo comportarse. Sin embargo, al lograr reducir el tiempo de análisis del algoritmo y mejorar los recursos hardware se podría mejorar estos parámetros.

- El principal inconveniente que se presentó en el desarrollo de la implementación fue el funcionamiento del controlador, dado que después de un tiempo de funcionamiento este dejaba de funcionar y la única solución que se encontró por parte de los otros desarrolladores fue volver a realizar la instalación e incorporación de las herramientas.

- Con la implementación de una red IoT definida por software se busca mostrar que la tecnología SDN ofrece soluciones ideales para gestionar la conectividad de los diferentes sistemas, permitiendo una reducción de costos a la hora de implementarlo en sistemas reales, dado que se reduciría el uso de equipos como los enrutadores lo que será una ventaja para las empresas.

5.2. TRABAJOS FUTUROS

Al finalizar el actual trabajo de grado surgieron nuevas propuestas de investigación que ayudan a mejorar las capacidades del algoritmo, el rendimiento de la red y manejo del controlador ONOS.

- Realizar la encapsulación del trabajo realizado en contenedores o *Docker* para facilitar el uso a terceros que lo requieran. Adicional a ello, hacer la solicitud a ONOS para añadir la aplicación desarrollada al conjunto de herramientas que hay en el código fuente de ONOS y mejorar la documentación de la wiki de la herramienta con los conocimientos adquiridos en el desarrollo de la aplicación y las soluciones encontradas.

- Analizar el desempeño de la red aplicando un mecanismo de control de tráfico basado en un modelo de inteligencia artificial distinto al trabajado como los algoritmos supervisados, y teniendo en cuenta parámetros diferentes a la pérdida de paquetes y el retardo extremo a extremo.

- Analizar el comportamiento de los parámetros y el desempeño de la red, al implementar el algoritmo de control de tráfico variando la prioridad de más de un dispositivos de la red dentro del controlador.

- Implementar este mecanismo en una red real de tipo *Smart Cities*, con la presencia de múltiples controladores de forma distribuido que permita experimentar escenarios complejos haciendo uso de las Eastbound y Westbound API teniendo en cuenta factores como el balanceo de carga, la seguridad así como mejorar el manejo de buffer.

Referencias

[1]. K. Lasse Lueth. "State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time". IoT Analytics. https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/ (Accedido el 15 de octubre de 2022).

[2]. "Descripción general de Internet de los objetos", Unión Internacional de Telecomunicaciones (UIT), Recomendación UIT-T Y.2060(06/2012), Accedido el 12 de Julio de 2021. [En línea]. Disponible en https://www.itu.int/rec/T-REC-Y.2060-201206-I/es

[3]. B. A. Hernández & D. P. Ortiz "Análisis general del enfoque IOT en redes", Tesis, Fundación Universitaria San Mateo, 2019. Accedido el 8 de marzo de 2022. [En línea]. Disponible: http://caoba.sanmateo.edu.co/jspui/bitstream/123456789/124/1/PROYECTO%20DE%20GRADO%20FINAL-pdf.pdf

[4]. J. González García, "IoT: Dispositivos, tecnologías de trasporte y aplicaciones", Tesis, Universidad abierta de Catalunya, Junio 2017. Accedido el 5 de enero de 2020. [En línea]. Disponible: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/64286/3/agonzalezgarcia0TFM0617memoria.pdf

[5]. N. E. Tarapuez, "Diseño y simulación de una red definida por software", bachelor Tesis, Universidad Central de Ecuador, 2020. Accedido el 8 de marzo de 2022. [En línea]. Disponible: http://www.dspace.uce.edu.ec/handle/25000/6505

[6]. R. A. Ríos Paredes, "Conceptualización de SDN y NFV", MASKAY, vol. 6, n.º 1, p. 29, noviembre de 2016. Accedido el 8 de marzo de 2022. [En línea]. Disponible: https://doi.org/10.24133/maskay.v6i1.163

[7]. Rodríguez, C. A. Talay, C.N. González, & L.A. Marrone, "Explorando las redes definidas por software (SDN)", In XXII Workshop de Investigadores en Ciencias de la Computación (WICC 2020, El Calafate, Santa Cruz).

[8]. J. Guano Viscarra, "Prototipo de una SDN utilizando herramientas open-source", Tesis, Universidad de Quito, 2017. Accedido el 8 de marzo de 2022. [En línea]. Disponible: https://bibdigital.epn.edu.ec/handle/15000/17327

[9]. P. Yagues Fernández, "Programación de redes SDN mediante el controlador POX", Tesis, Universidad de Cartagena, 2015. Accedido el 8 de marzo de 2022. [En línea]. Disponible: https://repositorio.upct.es/handle/10317/5254.

[10]. M. Barreto, "Evaluación de tecnología SDN", Tesis de Doctorado, Universidad Nacional de la Plata, 2017. Accedido el 8 de marzo de 2022. [En línea]. Disponible: http://sedici.unlp.edu.ar/handle/10915/67011

[11]. P. Sayans Cobos, "Despliegue de laboratorios virtualizados de SDN utilizando Open vSwitch", Tesis, Universidad Politécnica de Madrid, 2018. Accedido el 8 de marzo de 2022. [En línea]. Disponible: https://oa.upm.es/51526/

[12]. S. Rodríguez Santamaría, "Mecanismos de control de las comunicaciones en la Internet del futuro a través de OPENFLOW", Tesis, Universidad de Cantabria, 2012. Accedido el 8 de marzo de 2022. [En línea]. Disponible: https://repositorio.unican.es/xmlui/handle/10902/1165

[13]. J. Quimbayo Rodríguez, "Propuesta metodológica para la selección de controladores de redes SDN a nivel empresarial", Tesis, Universidad Santo tomas, 2020. Accedido el 8 de marzo de 2022. [En línea]. Disponible: https://repository.usta.edu.co/handle/11634/30425

[14]. GONZALES, O. FLAUZAC, & F. NOLOT, " EVOLUCIÓN Y CONTRIBUCIÓN PARA EL INTERNET DE LAS COSAS POR LAS EMERGENTES REDES DEFINIDAS POR SOFTWARE", IN MEMORIAS DE CONGRESOS UTP, 2018, (PP. 28-33). ACCEDIDO 8 DE MARZO 2022. [EN LÍNEA]. DISPONIBLE: HTTPS://REVISTAS.UTP.AC.PA/INDEX.PHP/MEMOUTP/ARTICLE/VIEW/1842

[15]. C. S. CHILIQUINGA RODRÍGUEZ, "INTERNET DE LAS COSAS BASADO EN REDES DEFINIDAS POR SOFTWARE", UNIVERSIDAD TÉCNICA DE AMBATO, TESIS, 2019. ACCEDIDO 8 DE MARZO DE 2022. [EN LÍNEA]. DISPONIBLE: http://repositorio.uta.edu.ec/handle/123456789/30708

[16]. M. OJO, D. ADAMI Y S. GIORDANO, "A SDN-IOT ARCHITECTURE WITH NFV IMPLEMENTATION", EN 2016 IEEE GLOBECOM WORKSHOPS (GC WKSHPS), WASHINGTON, DC, USA, 4–8 DE DICIEMBRE DE 2016. IEEE, 2016. ACCEDIDO EL 9 DE MARZO DE 2022. [EN LÍNEA]. DISPONIBLE: HTTPS://DOI.ORG/10.1109/GLOCOMW.2016.7848825

[17]. L. P. CARRASCO ESPINOSA, "ESTUDIO SOBRE LA APLICACIÓN DE LAS REDES DEFINIDAS POR SOFTWARE A LA INTERNET DE LAS COSAS. DISEÑO DE UN ESCENARIO DE PRUEBAS", TESIS MAESTRÍA, UNIVERSIDAD POLITÉCNICA DE MADRID, 2019. ACCEDIDO 8 DE MARZO 2022

[18]. S. Á. A. GÓMEZ, "ANÁLISIS Y CARACTERIZACIÓN DEL TRÁFICO HTC+MTC EN UNA SMART CITY. MODELADO DE FUENTES Y CALIDAD DE SERVICIO", TESIS DOCTORAL, UNIVERSIDAD POLITÉCNICA DE VALÈNCIA, 2018. ACCEDIDO EL 9 DE MARZO DE 2022. [EN LÍNEA]. DISPONIBLE: HTTP://HDL.HANDLE.NET/10251/107357

[19]. J. ZHAO, X. JING, Z. YAN Y W. PEDRYCZ, "NETWORK TRAFFIC CLASSIFICATION FOR DATA FUSION: A SURVEY", INFORMATION FUSION, VOL. 72, PP. 22–47, AGOSTO DE 2021. ACCEDIDO EL 9 DE MARZO DE 2022. [EN LÍNEA]. DISPONIBLE: HTTPS://DOI.ORG/10.1016/J.INFFUS.2021.02.009

[20]. C. A. MEDINA MONDRAGÓN, "CONTROL DE CONGESTIÓN EN REDES INALÁMBRICAS DE SENSORES", TESIS, PONTIFICIA UNIVERSIDAD JAVERIANA BOGOTÁ, 2017. ACCEDIDO EL 11 DE OCTUBRE DE 2021. [EN LÍNEA]. DISPONIBLE: HTTPS://REPOSITORY.JAVERIANA.EDU.CO/BITSTREAM/HANDLE/10554/34056/MEDINA MONDRAGONCAMILOALEJANDRO2017.PDF?SEQUENCE=3&ISALLOWED=Y

[21]. VALADARSKY, M. SCHAPIRA, D. SHAHAF, A. TAMA, "LEARNING TO RUTE", THE 16TH ACM WORKSHOP ON HOT TOPICS IN NETWORKS, 2017, P 185-191. [EN LÍNEA]. DISPONIBLE: HTTPS://CONFERENCES.SIGCOMM.ORG/HOTNETS/2017/PAPERS/HOTNETS17-FINAL28.PDF

[22]. SIVANATHAN, "IOT BEHAVIORAL MONITORING VIA NETWORK TRAFFIC ANALYSIS", DOCTORAL THESIS, SCHOOL OF ELECTRICAL ENGINEERING AND TELECOMMUNICATIONS THE UNIVERSITY OF NEW SOUTH WALES, SEPTEMBER 2019. ACCEDIDO EL 12 DE DICIEMBRE DE 2021. [EN LÍNEA]. DISPONIBLE: HTTPS://ARXIV.ORG/ABS/2001.10632

[23]. J. GONZALEZ GARCÍA, "IOT: DISPOSITIVOS, TECNOLOGÍAS DE TRASPORTE Y APLICACIONES", TESIS, UNIVERSIDAD ABIERTA DE CATALUNYA, JUNIO 2017. ACCEDIDO EL 5 DE ENERO DE 2020. [EN LÍNEA]. DISPONIBLE: HTTP://OPENACCESS.UOC.EDU/WEBAPPS/O2/BITSTREAM/10609/64286/3/AGONZALEZ GARCIA0TFM0617MEMORIA.PDF

[24]. BONACCORSO, “MACHINE LEARNING ALGORITHMS: POPULAR ALGORITHMS FOR DATA SCIENCE AND MACHINE LEARNING,” 2ND EDITION. PACKT PUBLISHING - EBOOKS ACCOUNT, 2018.

[25]. LUNA GONZALES, "TIPOS DE APRENDIZAJE AUTOMÁTICO", SOLDAI, FEBRERO 2018. ACCEDIDO EL 5 DE ENERO 2022. [EN LÍNEA]. DISPONIBLE: HTTPS://MEDIUM.COM/SOLDAI/TIPOS-DE-APRENDIZAJE-AUTOM%C3%A1TICO-6413E3C615E2

[26]. S. Perez Gómez, "Inteligencia Artificial Aplicada: Machine Learning", Huawei, 2022. Accedido Marzo 2022. [En línea]. Disponible: https://forum.huawei.com/enterprise/es/inteligencia-artificial-aplicada-machine-learning-parte-1/thread/836617-100757

[27]. Martínez Heras, "Las 7 fases del proceso de Machine Learning", septiembre de 2020, IArtificial.net. [En línea]. Disponible: https://www.iartificial.net/fases-del-proceso-de-machine-learning/

[28]. "Fases que componente el aprendizaje". [En línea]. Dipsonile: https://data.unimooc.com/materiales-cursos/machine-learning/Machine-Learning-9.pdf

[29]. G. Barrenetxea, "¿Cuáles son las etapas del Machine Learning?", solvenup, accedido el 9 de marzo de 2022. [En línea]. Disponible: https://solvenup.com/cuales-son-las-etapas-del-machine-learning/

[30]. "Etapas en un proyecto de machine learning", Sistemas Virtual, junio de 2020. Accedido el 9 de marzo de 2022. [En línea]. Disponible: https://sistemasvirtual.com/etapas-en-un-proyecto-de-machine-learning/

[31]. L. Lozano, "Aprendizaje por Refuerzo Elementos básicos y algoritmos", Universidad de Zaragosa. Accedido Marzo 2022. [En línea]. Disponible: https://core.ac.uk/download/pdf/289996907.pdf

[32]. P. Barrios, "Comparación de técnicas de aprendizaje por refuerzo jugando y videojuego de tennis", Universidad Politécnica de Madrid, 2019. Accedido Marzo de 2022. [En línea]. Disponible: https://oa.upm.es/55888/1/TFM_PABLO_SAN_JOSE_BARRIOS.pdf

[33]. Pegueroles Neyra, "Despliegue de un controlador SDN basado en ONOS", Universidad Politécnica de Catalunya, 2016. Accedido el 9 de marzo de 2022. [En línea]. Disponible: http://hdl.handle.net/2117/96674

[34]. V. KIVAHUK, "ESTUDIO DE LA PLATAFORMA ONOS PARA LA GESTIÓN DE ASOCIACIONES DE SEGURIDAD IPSEC EN ENTORNOS SDN", UNIVERSIDAD DE MURCIA, SEPTIEMBRE DE 2018. ACCEDIDO EL 9 DE MARZO DE 2022. [EN LÍNEA]. DISPONIBLE: HTTPS://DIGITUM.UM.ES/DIGITUM/BITSTREAM/10201/61119/1/TFG_INFOR.PDF

[35]. MIÑONES RODRÍGUEZ, "MONITORIZACIÓN DE REDES DEFINIDAS POR SOFTWARE (SDN) UTILIZANDO ONOS", TESIS, UNIVERSIDAD DE CORUÑA, 2020. ACCEDIDO EL 9 DE MARZO DE 2022. [EN LINE]. DISPONIBLE: HTTPS://RUC.UDC.ES/DSPACE/HANDLE/2183/26173

[36]. OPEN NETWORK OPERATING SYSTEM, ACCEDIDO JUNIO 2021. [EN LÍNEA]. DISPONIBLE: HTTPS://OPENNETWORKING.ORG/ONOS/

[37]. R. A. GARCÍA, D. F. HERNÁNDEZ & J. C. AGUDELO, "IMPLEMENTACIÓN DE UN GENERADOR DE TOPOLOGÍAS ALEATORIAS EN EL EMULADOR DE RED MININET", TESIS, UNIVERSIDAD CATÓLICA DE PEREIRA, 2015, ACCEDIDO 15 DE MARZO DE 2022. [EN LÍNEA]. DISPONIBLE: HTTPS://REPOSITORIO.UCP.EDU.CO/BITSTREAM/10785/3678/1/CDMIST122.PDF

[38]. R. D. R. FONTES Y C. E. ROTHENBERG, "MININET-WIFI", EN SIGCOMM '16: ACM SIGCOMM 2016 CONFERENCE, FLORIANOPOLIS BRAZIL. NEW YORK, NY, USA: ACM, 2016. ACCEDIDO EL 15 DE MARZO DE 2022. [EN LÍNEA]. DISPONIBLE: HTTPS://DOI.ORG/10.1145/2934872.2959070

[39]. R. R. FONTES, S. AFZAL, S. H. B. BRITO, M. A. S. SANTOS Y C. E. ROTHENBERG, "MININET-WIFI: EMULATING SOFTWARE-DEFINED WIRELESS NETWORKS", EN 2015 11TH INTERNATIONAL CONFERENCE ON NETWORK AND SERVICE MANAGEMENT (CNSM), BARCELONA, SPAIN, 9–13 DE NOVEMBER DE 2015. IEEE, 2015. ACCEDIDO EL 15 DE MARZO DE 2022. [EN LÍNEA]. DISPONIBLE: HTTPS://DOI.ORG/10.1109/CNSM.2015.7367387

[40]. R. S. PRESSMAN, INGENIERÍA DEL SOFTWARE - UN ENFOQUE PRACTICO 7: EDICION. MCGRAW-HILL COMPANIES, 2010.

[41]. K. H. Pries y J. M. Quigley, Scrum Project Management. CRC Press, 2010. Accedido el 15 de marzo de 2022. [En línea]. Disponible: https://doi.org/10.1201/9781439825174

[42]. M. Trigas gallego, "TFC. Metodología Scrum. GESTION DE PROYECTOS INFORMÁTICOS.", **2015. Disponible:** https://docplayer.es/917979-Tfc-metodologia-scrum-gestion-de-proyectos-informaticos-autor-manuel-trigas-gallego-consultora-ana-cristina-domingo-troncho.html

[43]. technobium. "q-learning-java/QLearning.java at master · technobium/q-learning-java". GitHub. https://github.com/technobium/q-learning-java/blob/master/src/main/java/com/technobium/rl/QLearning.java (Accedido el 11 de Julio de 2022).

[44]. K. Nykjaer. "Q-learning example with Java". Software Programming. https://kunuk.wordpress.com/2010/09/24/q-learning/ (accedido el 11 de julio de 2022).

[45]. K. Teknomo. "Q-Learning By Examples: Formula". Revoledu – Just another WordPress site. https://people.revoledu.com/kardi/tutorial/ReinforcementLearning/Q-Learning.htm (accedido el 11 de julio de 2022).

[46]. C. Jiménez, "Caracterización del tráfico de dispositivos de IoT usados en residencias", Tesis Doctoral, Universidad de Costa rica, 2018. Accedido el 8 de marzo de 2022. [En línea]. Disponible: https://pci.ucr.ac.cr/sites/default/files/trabajos_de_graduacion/A01127.pdf

ANEXO A. INSTALACIÓN DE VMWARE Y S.O

Para efectos de este trabajo se trabajará con la herramienta VMware Workstation, la cual es un hipervisor de escritorio que facilita la instalación de máquinas virtuales. El sistema operativo por instalar en la maquina será Ubuntu 18.04

Características del Equipo de Trabajo

- **Edición: Operativo Windows 10 Pro**
- **Procesador: Intel(R) Xeon(R) Silver 4108 CPU @ 1.8GHz**
- **Memoria RAM: 256 GB**
- **Tipo de Sistema: Sistema operativo de 64 bits, procesador basado en x64**

VMware Workstation

Para la instalación del VMware, inicialmente se descarga el archivo *.exe* desde la página oficial https://www.vmware.com/co/products/workstation-pro.html. Una vez descargado se procede a ejecutarlo y a seguir los pasos, los cuales se observan desde la Figura A-0-1 hasta Figura A-0-10.

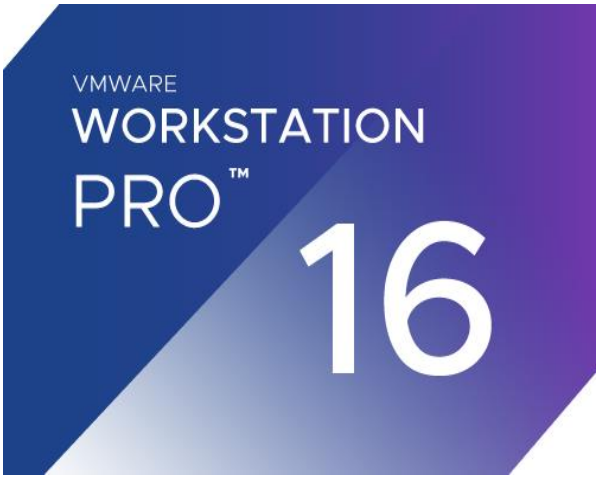

Figura A-0-2 Aquí inicia del Proceso de Instalación de VMware

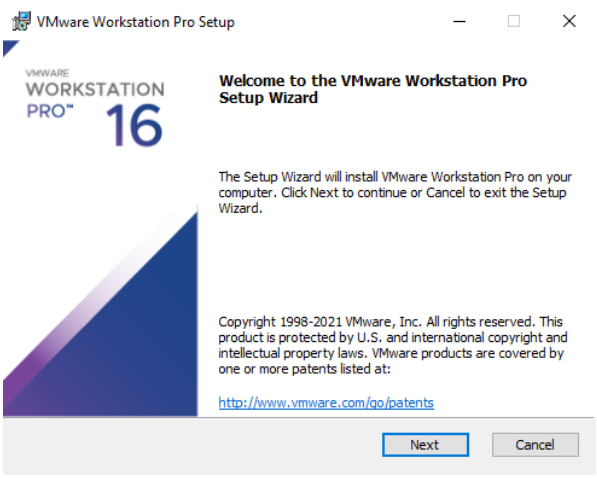

Figura A-0-1 Ventana de bienvenida

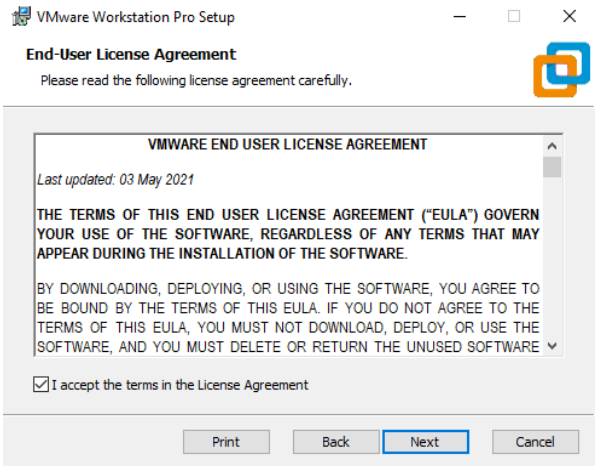

Figura A-0-3 Aceptación de términos y condiciones

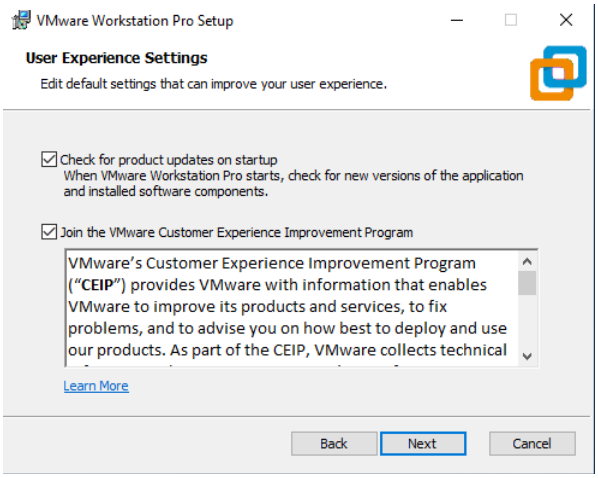

Figura A-0-4 Condiciones para mejorar la experiencia con VMware

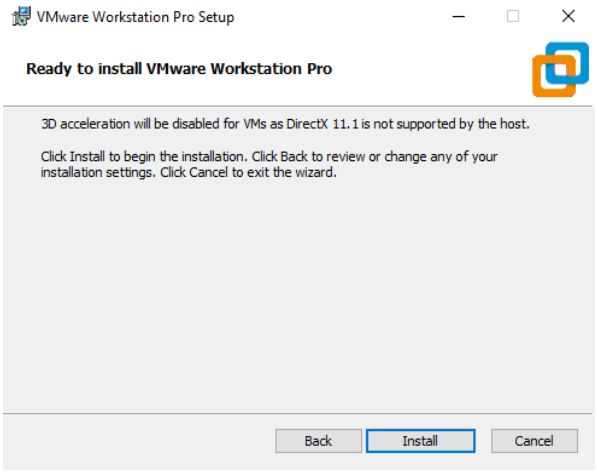

Figura A-0-8 Finalización configuración

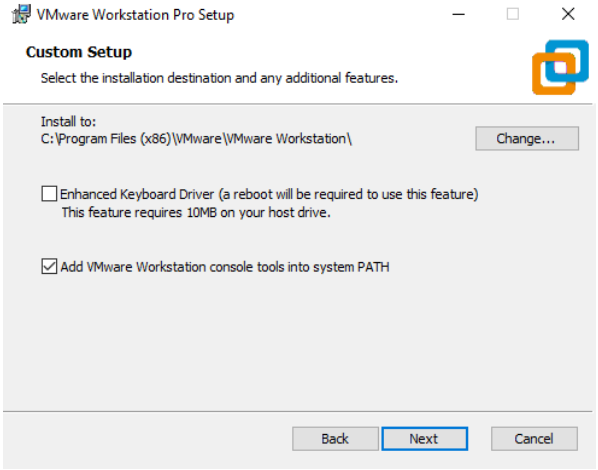

Figura A-0-6 Selección de ubicación de instalación

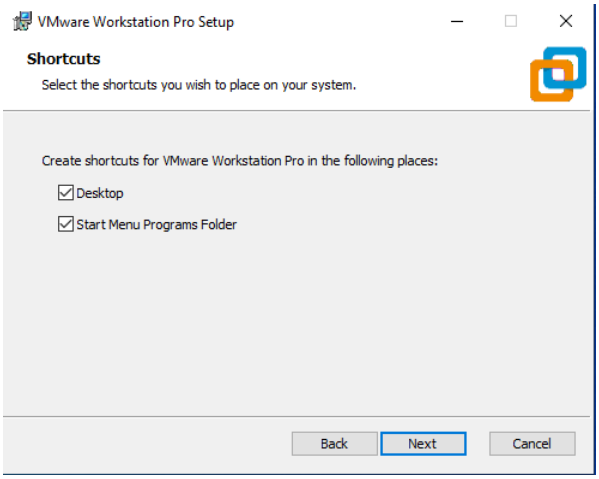

Figura A-0-5 Creación del icono de VMware en el escritorio

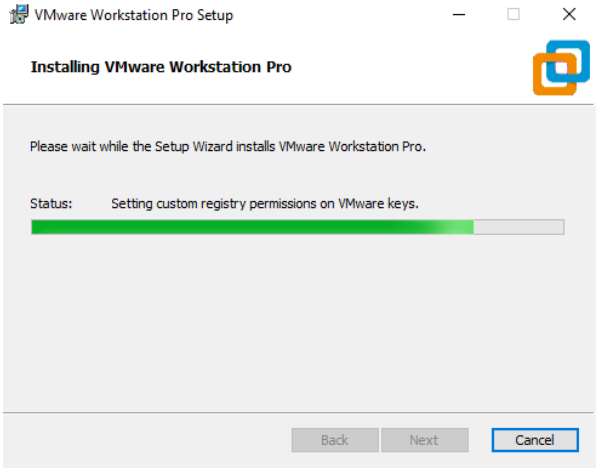

Figura A-0-7 Comienzo de la instalación

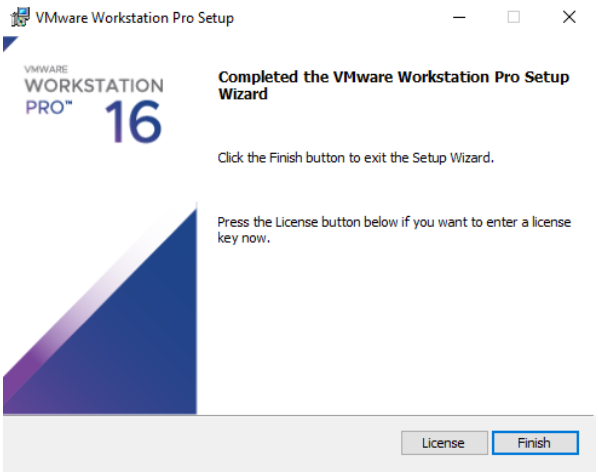

Figura A-0-10 Ventana final de la instalación

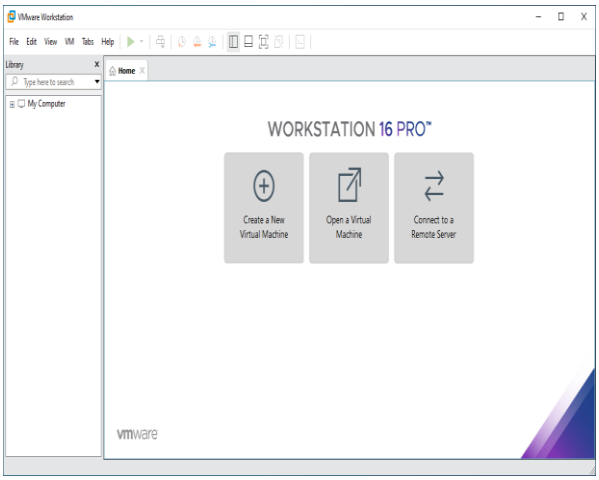

Figura A-0-9 Ventana de inicio de VMware

Creación de la Máquina Virtual e Instalación del S.O

Para realizar la creación de la máquina y la instalación del sistema operativo Ubuntu, lo pasos a seguir son muy mecánicos, puesto que VMware tiene la facilidad de que las configuraciones las hace internamente. Antes de iniciar el proceso asegúrese de descargar la imagen ISO del Sistema Operativo; una vez se tenga el archivo descargado se procede con la instalación, a continuación, desde la Figura A-0-11 hasta la Figura A-0-27 se muestra el proceso realizado para esta parte.

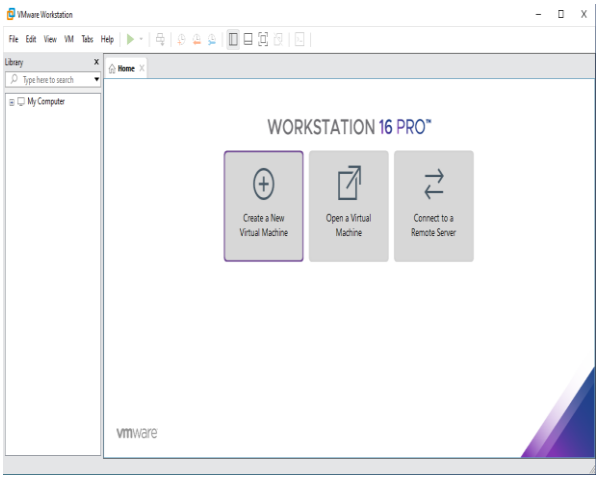

Figura A-0-11 Inicio del proceso de creación de una nueva máquina virtual

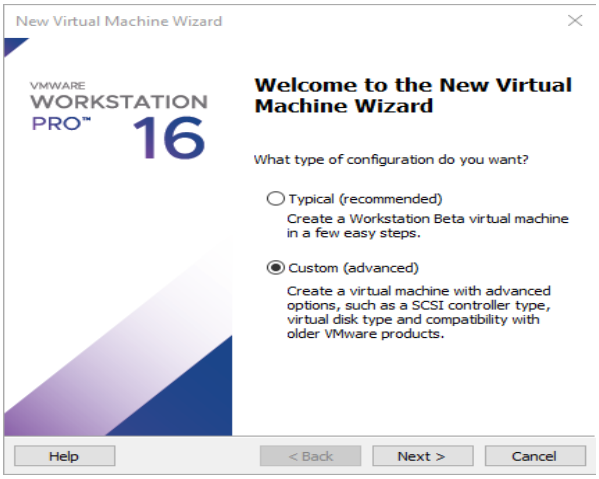

Figura A-0-12 Selección del tipo de instalación

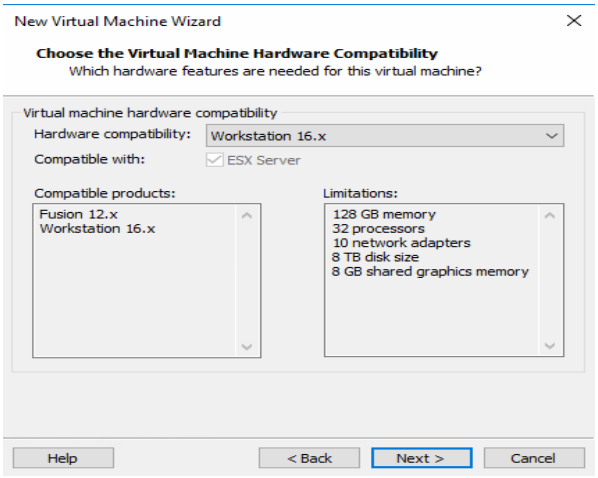

Figura A-0-14 características necesarias para la máquina

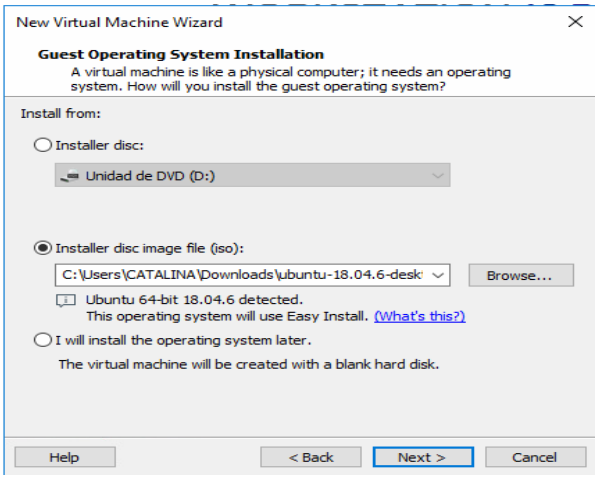

Figura A-0-13 Ventana para cargar la imagen del S.O

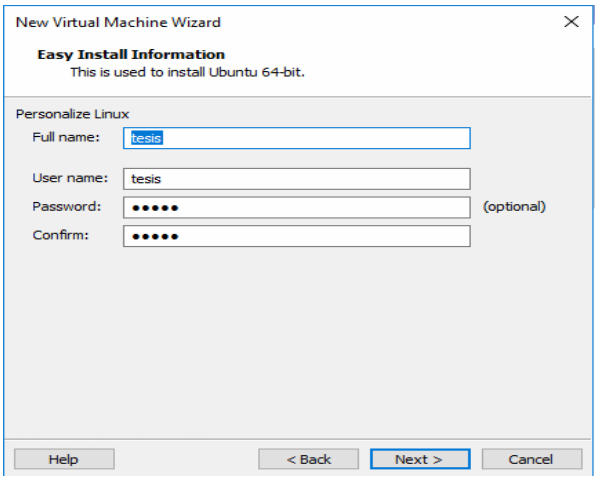

Figura A-0-17 Configuración de credenciales

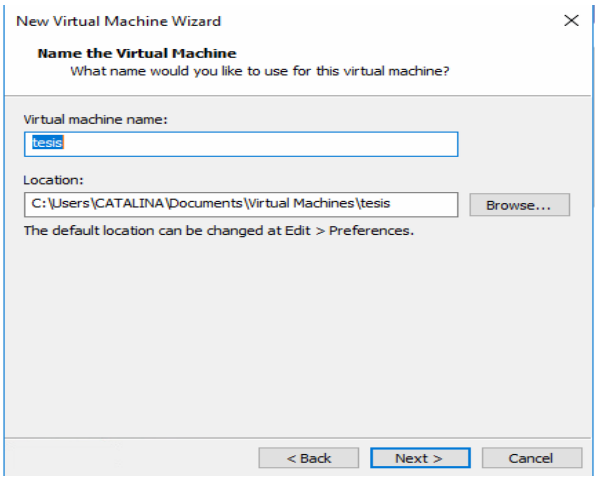

Figura A-0-16 Configuración del nombre de la máquina

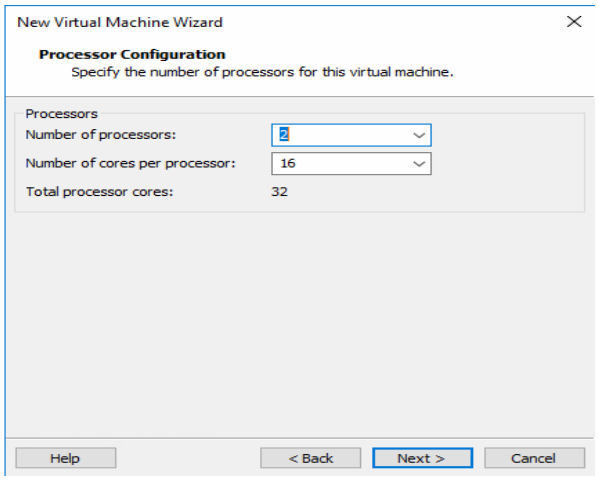

Figura A-0-18 Asignación del recurso de procesador

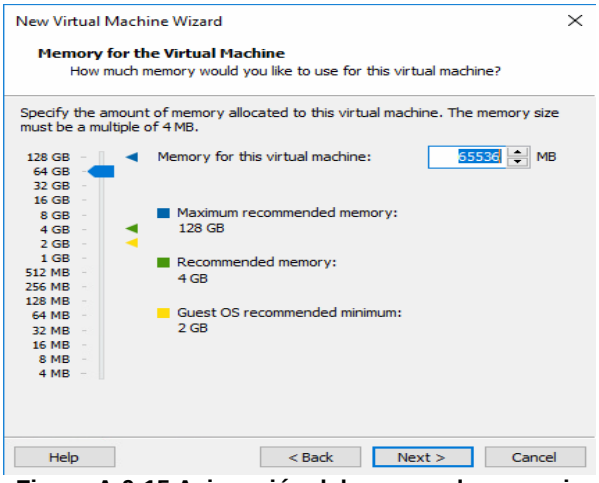

Figura A-0-15 Asignación del recurso de memoria RAM

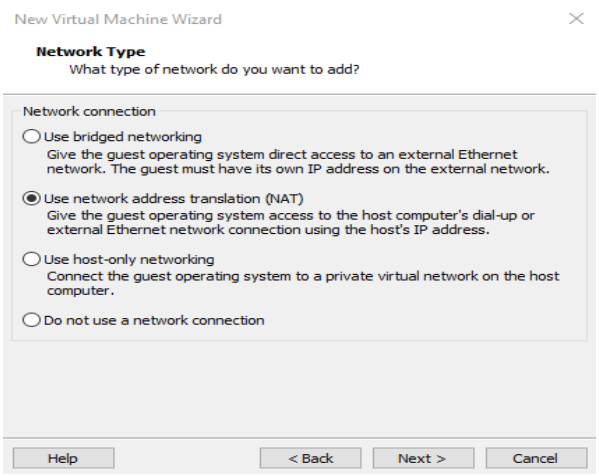

Figura A-0-21 Configuración de la interfaz de red

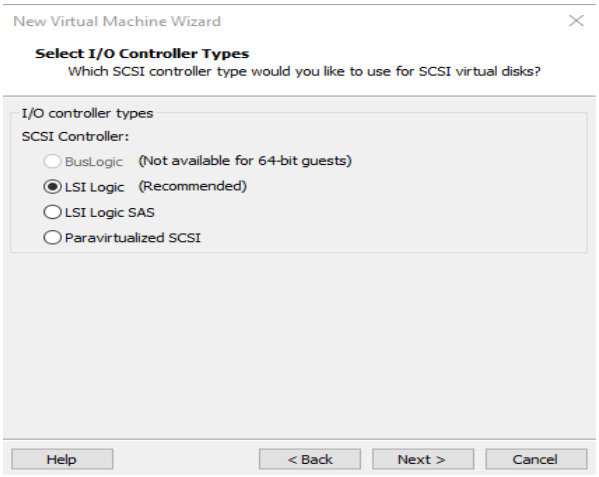

Figura A-0-19 Selección del tipo de controlador para el disco

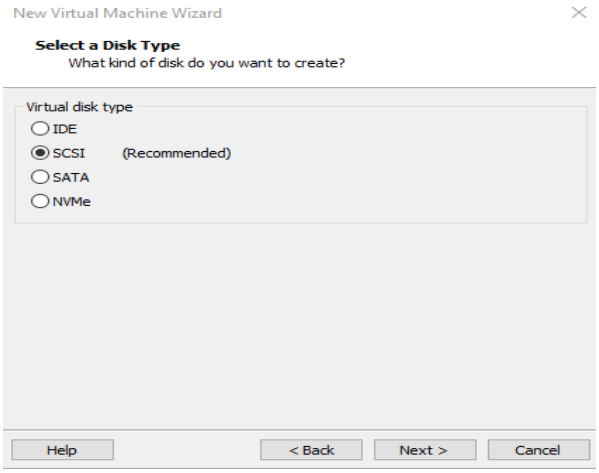

Figura A-0-20 Selección del tipo de disco

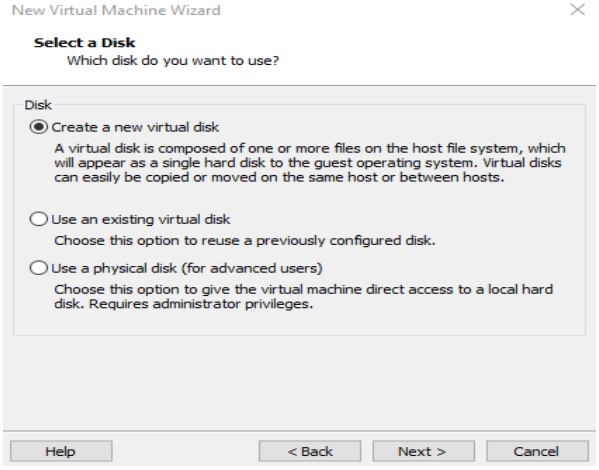

Figura A-0-24 Creación del disco virtual

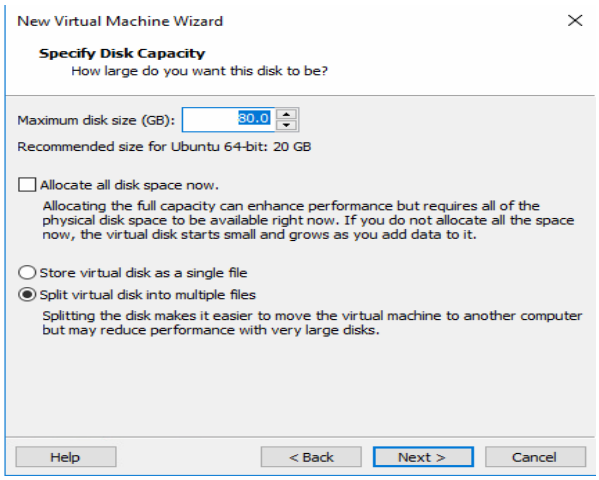

Figura A-0-23 Asignación del recurso de Disco duro

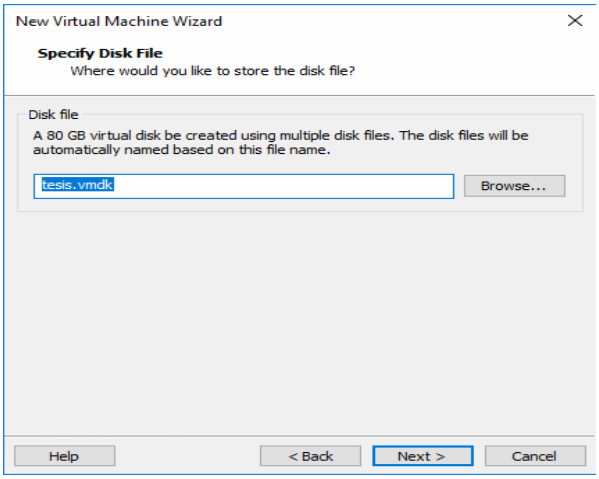

Figura A-0-22 Selección del lugar de almacenamiento

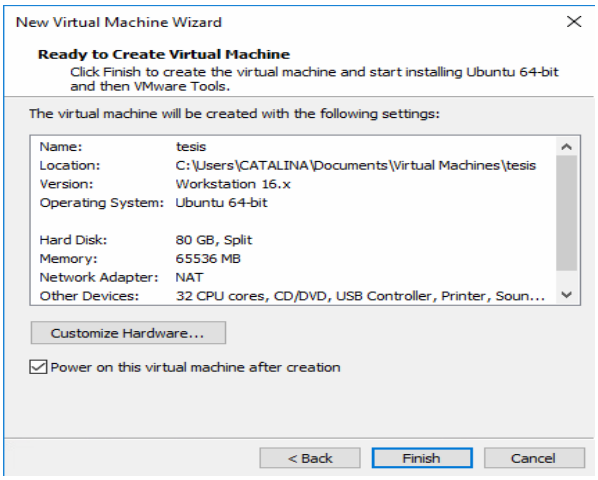

Figura A-0-26 Ventana de resumen de los parámetros de configuración

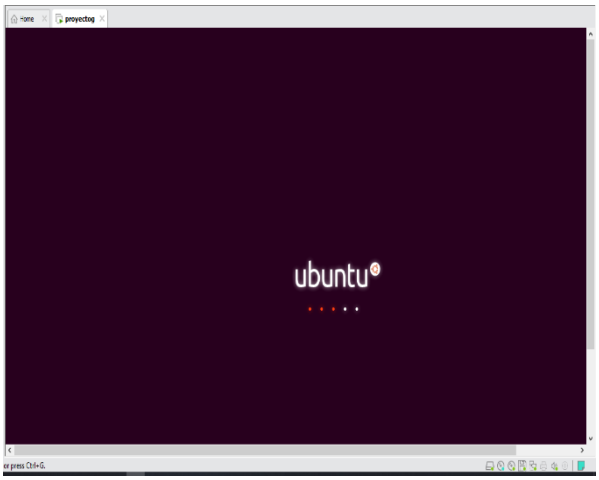

Figura A-0-25 Ventana de inicio de instalación de la máquina

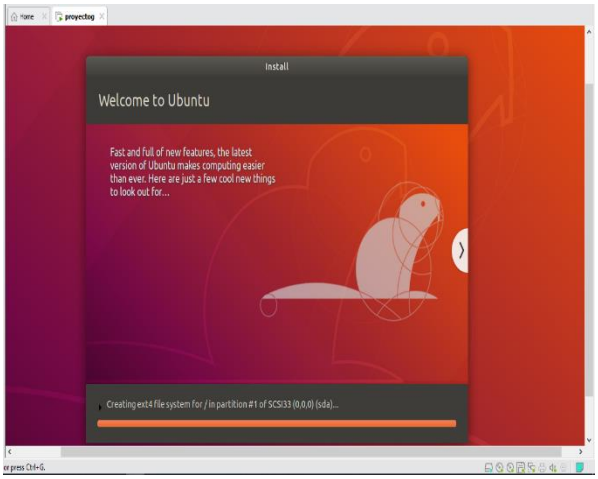

Figura A-0-28 Ventana del proceso de instalación

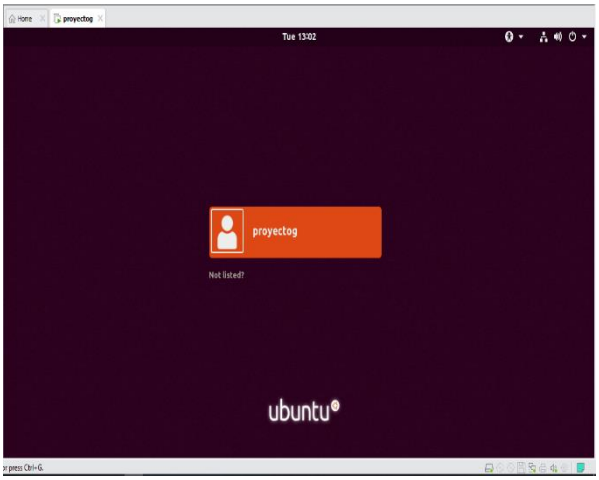

Figura A-0-27 Finalización del proceso de creación de una nueva máquina virtual

ANEXO B. INSTALACIÓN DE MININET Y COMPONENTES ADICIONALES.

Mininet

Antes de realizar la instalación de mininet es necesario instalar otras herramientas, además de realizar la actualización de todos los paquetes; a continuación, se encuentran los comandos a seguir para la instalación de *java*, *Maven*, el *software git* y *mininet-wifi*

Instalación de Java

:~$sudo su apt-get update
:~$sudo su apt-get upgrade
:~$sudo apt install default-jre
:~$sudo apt install default-jdk
:~$sudo java -version
:~$sudo update-alternatives --config java
:~$export JAVA_HOME=/usr/lib/jvm/java-aa-openjdk-amd64
:~$echo $JAVA_HOME

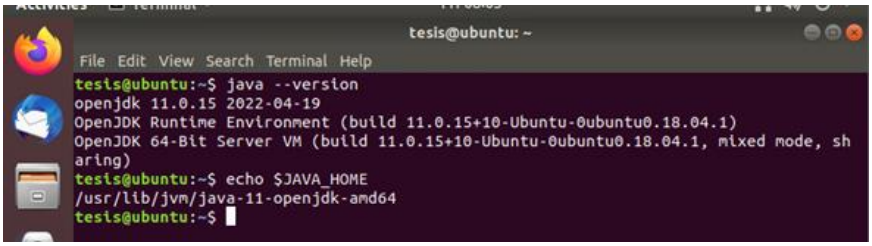

Figura B-0-1 Comando de Verificación de JAVA

Instalación del Maven

:~$sudo apt update
:~$sudo apt install maven
:~$mvn -version

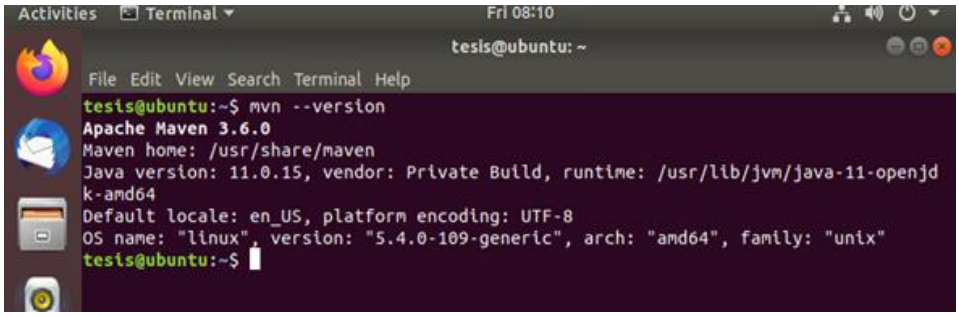

Figura B-0-2 Comando de verificación de Maven

Git-Core, Net-tools, Curl

:~$sudo apt-get install git-core
:~$sudo apt-get install net-tools
:~$sudo apt-get install curl

Python

:~$sudo apt install python3-pip
:~$sudo apt install python
:~$sudo apt install python-pip

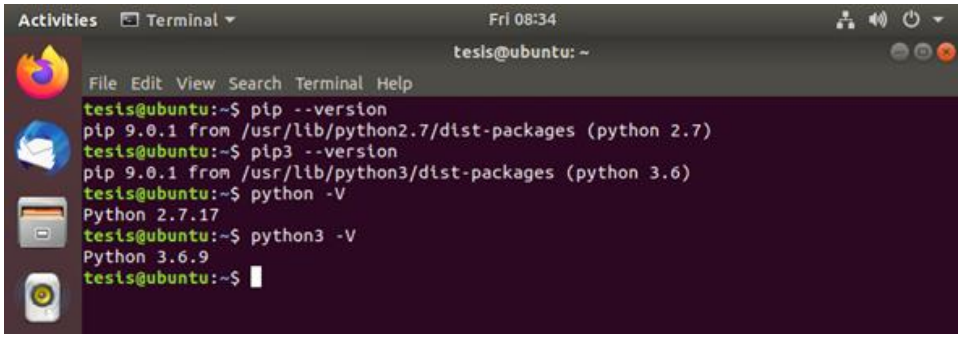

Figura B-0-3 Verificación de instalación de python

Mininet-wifi

:~/Downloads$sudo apt-get update
:~/Downloads$sudo apt-get upgrade
:~/Downloads$git clone https://github.com/intrig-unicamp/mininet-wifi
:~/Downloads$sudo chmod 775 mininet-wifi

:~/Downloads$cd mininet-wifi
:~/Downloads/mininet-wifi$git config –global –add safe.directory /tu-ruta-hacia/mininet-wifi
:~/Downloads/mininet-wifi$sudo apt-get install autoconf
:~/Downloads/mininet-wifi$sudo util/install.sh -Wnfvl
:~/Downloads/mininet-wifi$sudo util/install.sh -6
:~/Downloads/mininet-wifi$sudo util/install.sh -p
:~/Downloads/mininet-wifi$sudo chown -R proyectog:proyectog mininet-wifi
:~/Downloads/mininet-wifi$sudo mn --wifi

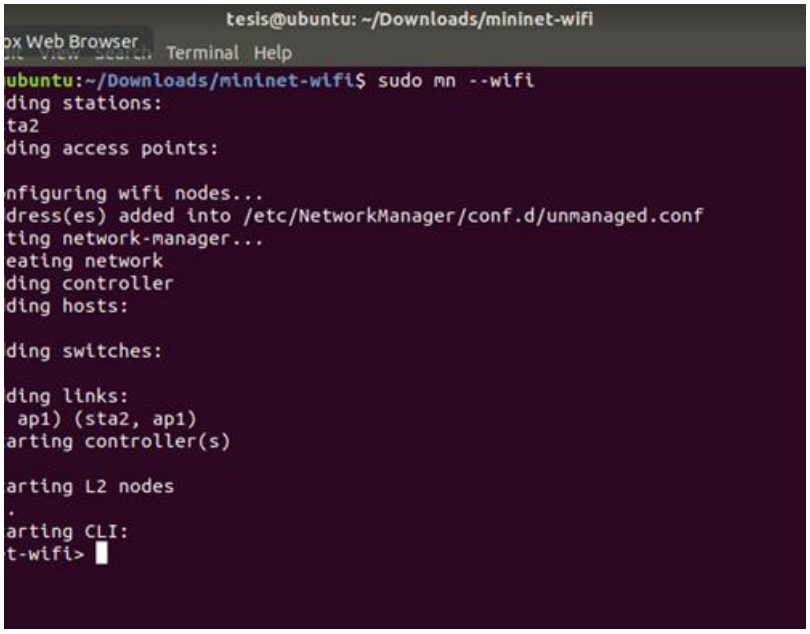

Figura B-0-4 Comando de prueba de Mininet

MiniEdit

Para acceder al entorno de trabajo de MiniEdit, primero se debe verificar que Python esté funcionando correctamente, para ello se deben ingresar lo siguientes comandos

:~/Downloads/mininet-wifi$sudo rm -rf /usr/bin/python3
:~/Downloads/mininet-wifi$sudo rm -rf /usr/bin/python
:~/Downloads/mininet-wifi$sudo ln -s /usr/bin/python3.6 /usr/bin/python3
:~/Downloads/mininet-wifi$sudo ln -s /usr/bin/python3.6 /usr/bin/python

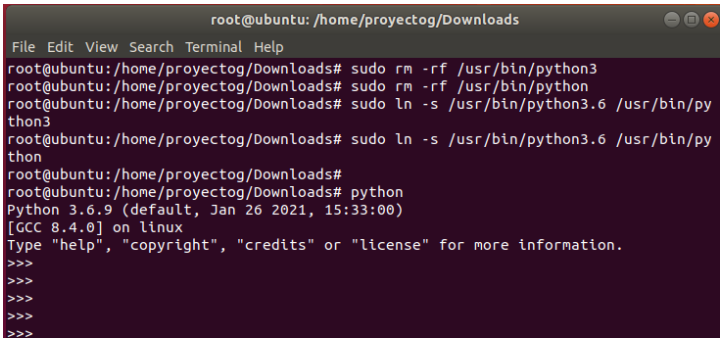

Figura B-0-5 Comando de verificación de Python

Una vez verificado que Python esté funcionando correctamente procedemos a ingresar al entorno de trabajo de *MiniEdit*, para ello se ingresa a la carpeta donde este instalado *mininet-wifi* después a la carpeta *examples* y finalmente *MiniEdit*, es este caso se tiene la carpeta en *Downloads* por tanto los comando para ingresar son los siguientes:

:~/$cd Downloads
:~/Downloads$cd mininet-wifi
:~/Downloads/mininet-wifi$cd examples
:~/Downloads/mininet-wifi/examples$./miniedit.py

Figura B-0-6 Entorno de trabajo de MiniEdit

Generador de Tráfico D-ITG

Los comandos que se listan a continuación son los que se deben seguir para lograr la correcta instalación del D-ITG

```
:~/Downloads$sudo apt-get install unzip
:~/Downloads$sudo apt-get install g++
:~/Downloads$wget  http://traffic.comics.unina.it/software/ITG/codice/D-ITG-2.8.1-r1023-src.zip
:~/Downloads$unzip D-ITG-2.8.1-r1023-src.zip
:~/Downloads$cd D-ITG-2.8.1-r1023/src
:~/Downloads/ D-ITG-2.8.1-r1023/src$make
```

Este generador de tráfico da la facilidad de configurar los equipos terminales como emisores o receptores para poder generar tráfico dentro de la red, para ello se debe tener en cuenta algunos parámetros básicos, los cuales se listan a continuación

- Protocolo de Comunicaciones: se usa el comando “-T” seguido del protocolo
- Dirección IP del equipo a donde se va a enviar: se usa el comando “-a” seguido de la dirección IP del equipo seleccionado
- Tamaño del paquete en Bytes que se va a trasmitir: se usa el comando “-c” seguido de la cantidad de datos
- Cantidad de Paquetes por segundo: se usa el comando “-C” seguido por el número y paquetes de se va a enviar
- Tiempo: se usa el comando “-t” seguido por la cantidad de tiempo que va a durar la transmisión
- Almacenar estadísticas: para realizar el registro de los resultados de la transmisión se emplean los comandos “-l sebder.log” y “-x receiver.log” para guarda los datos transmitidos y recibidos respectivamente

Para hacer uso del generador se ejecuta el comando *xterm* en el *CLI* de mininet seguido del nombre de los equipos terminales que se quieren configurar, en la se observa el resultado de lo mencionado anteriormente

ANEXO C. INSTALACIÓN Y CONFIGURACIÓN DEL CONTROLADOR ONOS

Una vez instalados todos los paquetes anteriores, se procede a descargar e instalar el archivo *onos-2.5.1.tar.gz;* adicional a ello, se procede a realizar un cambio de nombre de la carpeta onos y a almacenarla en un nuevo fichero para garantizar que el propietario sea el usuario normal de *Linux*. A continuación, se muestran los comandos a seguir para la instalación.

:~$cd Downloads
:~/Downloads$sudo wget https://repo1.maven.org/maven2/org/onosproject/onos-releases/2.5.1/onos-2.5.1.tar.gz
:~/Downloads$tar -xvzf onos-2.5.1.tar.gz
:~/Downloads$sudo mkdir -p /opt && cd /opt
:~/opt$sudo mv ~/Downloads/onos-2.5.1 /opt/onos
:~/opt$sudo chown -R tesis:tesis onos

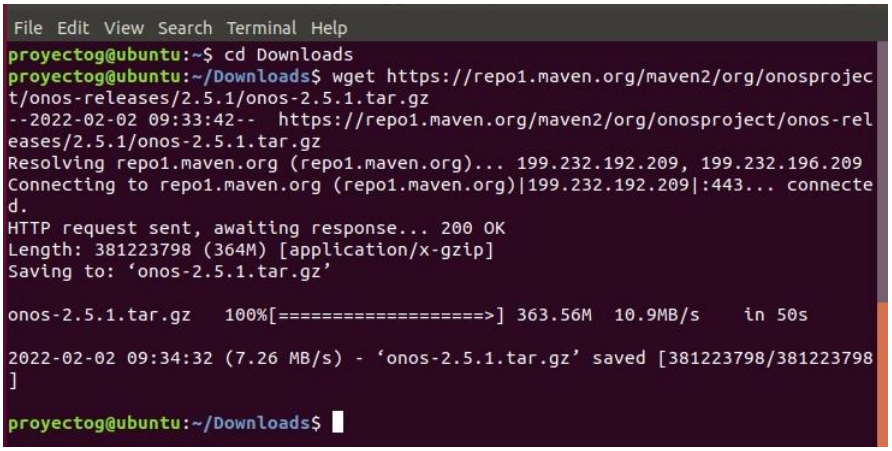

Figura C-0-1 Comando de descarga de onos

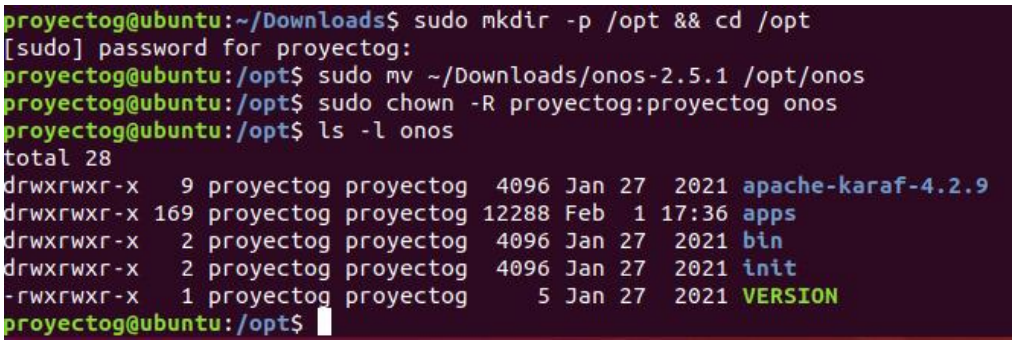

Figura C-0-2 Comandos para el cambio de carpeta y propietario

El siguiente paso será configurar las opciones de inicio del controlador, es decir en la consola de comandos se deberá ejecutar las líneas que aparecen a continuación, esto con el fin de que el sistema cuando lea la palabra *sdn* u *onos* sepa cómo comportarse

:~/opt$export ONOS_USER=sdn
:~/opt$export ONOS_APP=drivers,openflow

Los siguientes comandos se ejecutarán para inicializar los archivos de servicio del controlador y para que se pueda ejecutar el controlador.

:~/opt$sudo cp /opt/onos/init/onos.initd /etc/init.d/onos
:~/opt$sudo cp /opt/onos/init/onos.service /etc/systemd/system/
:~/opt$sudo systemctl daemon-reload
:~/opt$sudo systemctl enable onos

Una vez instalados los archivos, no se debería tener inconveniente alguno con el controlador y su funcionamiento, para corroborar el estado de este se ejecutan los siguientes comandos

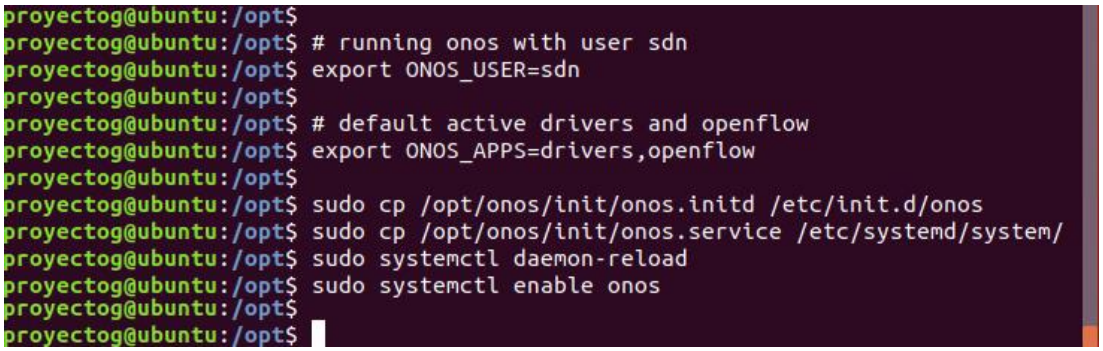

Figura C-0-3 Comandos de configuración e inicio del controlador

:~/opt$sudo systemctl status onos.service

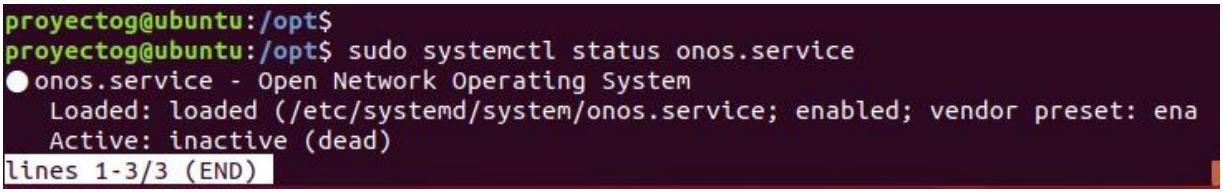

Figura C-0-4 Verificación del estado del controlador

se puede ver en la que el controlador está instalado correctamente, sin embargo, no está activo para ello se ejecuta el siguiente comando y luego se vuelve e a verificar el estado en el que se encuentra

:~/opt$sudo systemctl start onos.service

:~/opt$sudo systemctl status onos.service

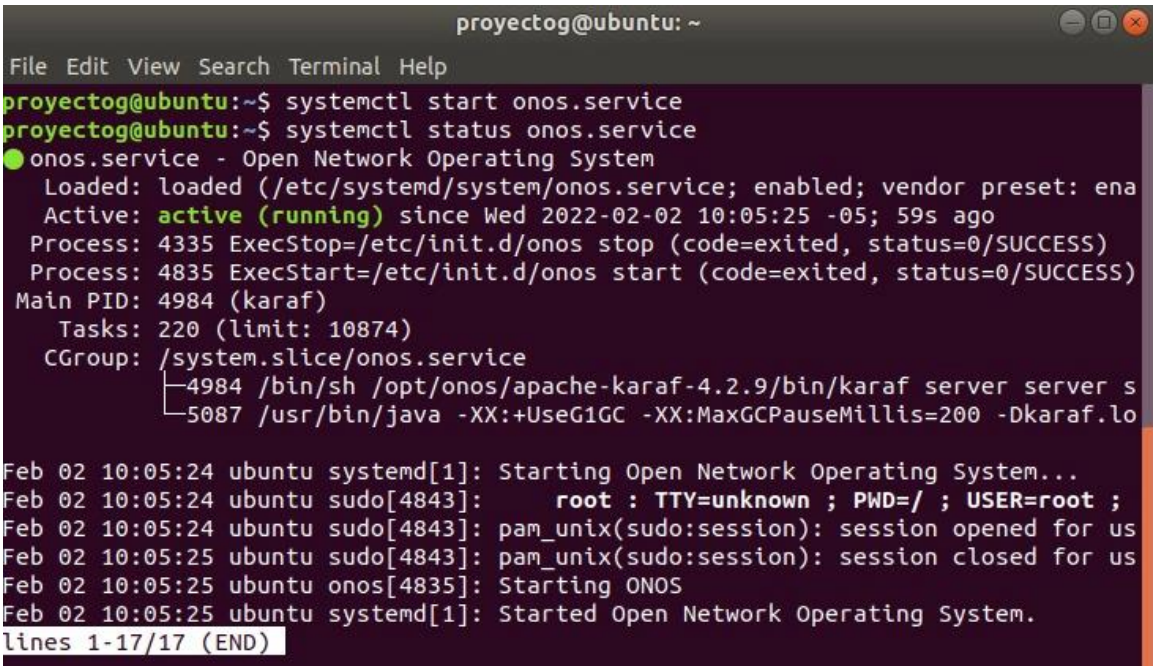

Figura C-0-5 Arranque de controlador y verificación

En la Figura C-0-5 se muestra que ya se tiene el controlador funcionando y corriendo, sin embargo, aún no se tiene un Shell para poder acceder a él. Entonces para poder acceder al controlador por medio del CLI se debe crear una clave de inicio de sesión, para eso se deben ejecutar los siguientes comandos.

:~$ssh-keygen -t rsa
:~$/opt/onos/bin/onos-user-key $USER ~/.ssh/id.rsa.pub

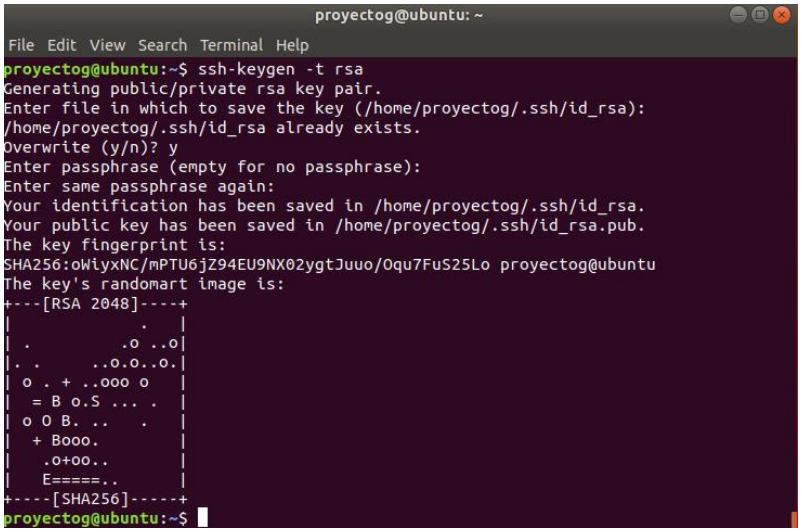

Figura C-0-6 Configuración de la clave de onos y del archivo público

Una vez configurada la clave, ya se puede acceder al Shell del onos con el siguiente comando.

:~/opt$/opt/onos/bin/onos

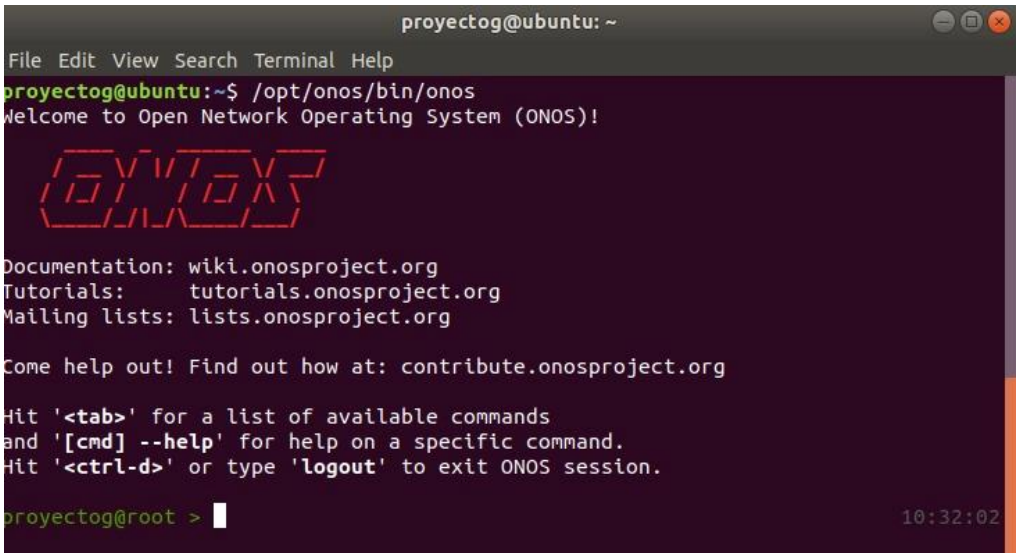

Figura C-0-7 Ventana de comando del controlador

Dado que la instalación del controlador se hizo por medio del *tar.gz* no se cuenta con algunos archivos que son necesarios para el desarrollo de aplicaciones; por tanto, se procede a clonar la carpeta onos con el comando *gerrit*, el cual se presenta a continuación

:~$git clone https://gerrit.onosproject.org/onos

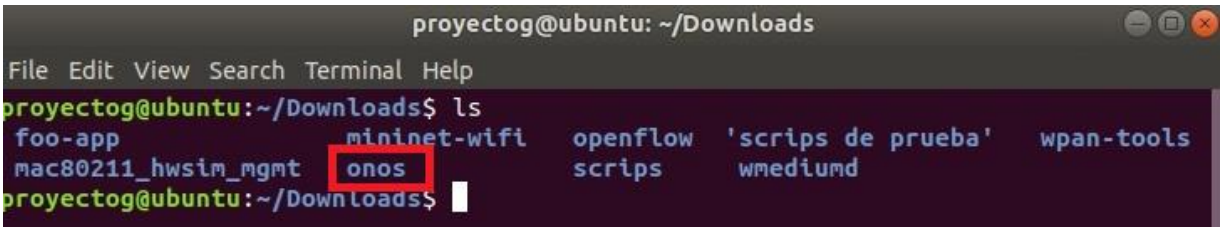

Figura C-0-8 Verificación de la clonación de la carpeta onos

finalmente, para evitar tener que estar ejecutando los comando para el arranque del controlador cada que se inicia la máquina, sino que por el contrario se realiza la siguiente configuración en el archivo *bashrc* para que cada que se inicie la mv se inicie el controlador.

nano ~/.bashrc

export ONOS_ROOT=~/Downloads/onos

source ~/Downloads/onos/tools/dev/bash_profile

source ~/.bashrc

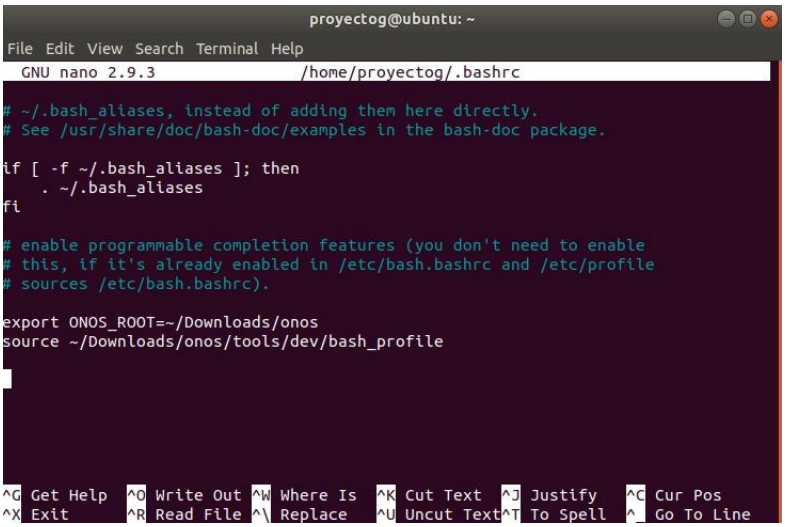

Figura C-0-9 Cambios en el archivo bashrc

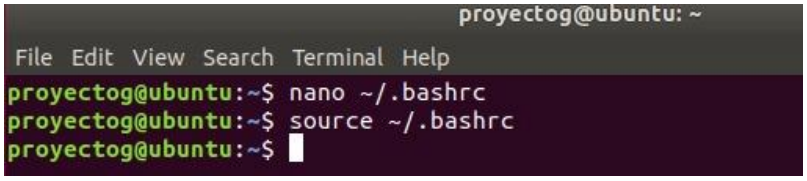

Figura C-0-10 Verificación del archivo bashrc

Establecer la conexión de mininet con ONOS

Una vez iniciados ONOS; es necesario verificar que el protocolo OpenFlow este activado, para ello en la GUI se escribirá el siguiente comando ***apps -a -s*** el cual mostrará la lista de aplicaciones activadas que tiene el controlador, en la Figura C-0-11 se observa que en este caso solo hay dos aplicaciones activadas, por tanto se ejecuta el comando ***app -s*** el cual desplegara la lista de aplicaciones disponibles para activarse; para hacer la respectiva activación se usara el comando ***app activate org.onosproject.fwd;*** este comando se podrá usar para activar cualquier aplicación que se necesite, en este caso como se necesita el protocolo *OpenFlow* se activa el *fwd* que es el de renvío de paquetes, en la se observa la activación de la aplicación.

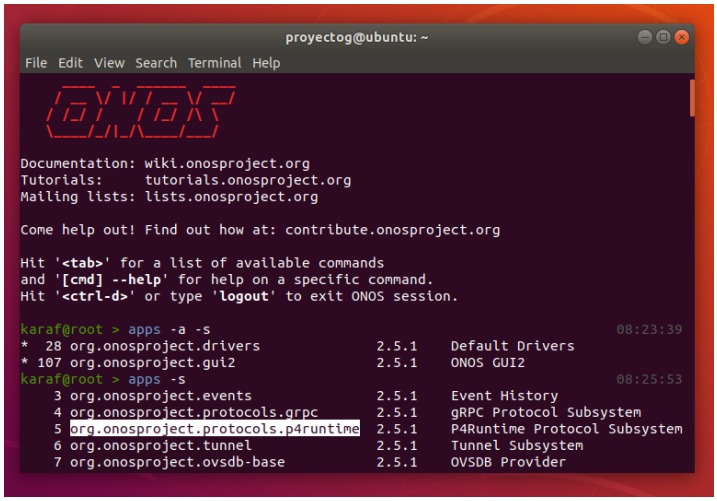

Figura C-0-11 Comando para mostrar aplicaciones en el CLI de ONOS

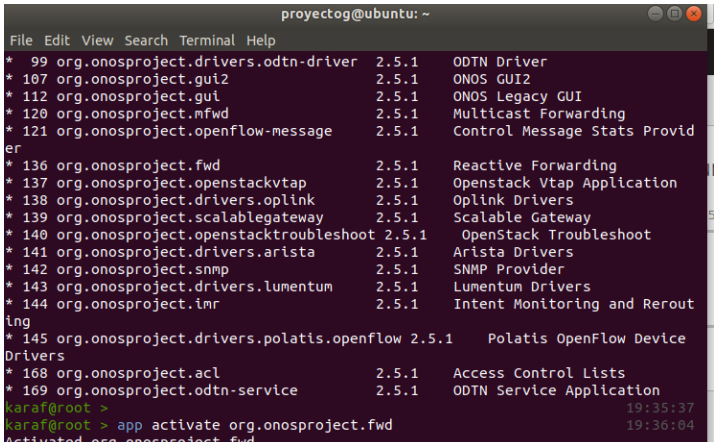

Figura C-0-12 Comando para la activación de aplicaciones

ANEXO D. CONFIGURACIÓN DE LA APLICACIÓN

Para la creación de una nueva aplicación en la herramienta onos, se procede a descargar la plantilla que los desarrolladores crearon, esto con el fin de tener la estructura base que maneja onos y a partir de ahí crearla según los requerimientos que se necesiten.

```
:~/Downloads$export OOS_POM_VERSION=2.5.1
:~$onos-create-app app org.foo foo-app 1.0-SNAPSHOT org.foo.app
:~$cd foo-app
:~$mvn clean install
:~$onos-app localhost install! Target/foo-app-1.0-SNAPSHOT.oar
```

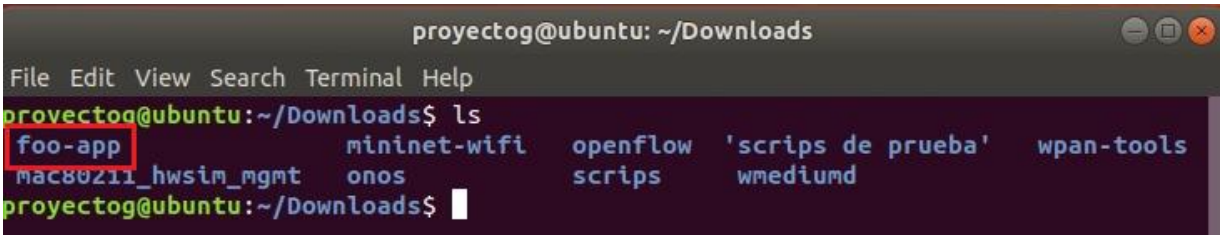

Figura D-0-1 Visualización en el directorio de la carpeta de la aplicación

Una vez se halla descargado la plantilla, se creará una carpeta con el nombre de la aplicación en el directorio en el cual se realizó el proceso como en la Figura D-0-1, en este caso la carpeta tiene el nombre foo-app y ahí se encontrarán todos los archivos de la platilla de la aplicación, que se usarán para crear las nuevas tal y como se puede ver en la Figura D-0-2 y Figura D-0-3.

Figura D-0-2 Carpeta de la aplicación

Figura D-0-3 Archivos base de la aplicación

Lo que procede a continuación es editar el archivo *pom.xml* ; para ello, en un editor de código se abre el archivo y se hacen los cambio que se muestran en la Figura D-0-4 Estructura del archivo pom.xml, eso se hace para configurar en la plantilla la versión de onos que se esté manejando, para el desarrollo de este trabajo se implementó la version 2.5.1 del controlador.

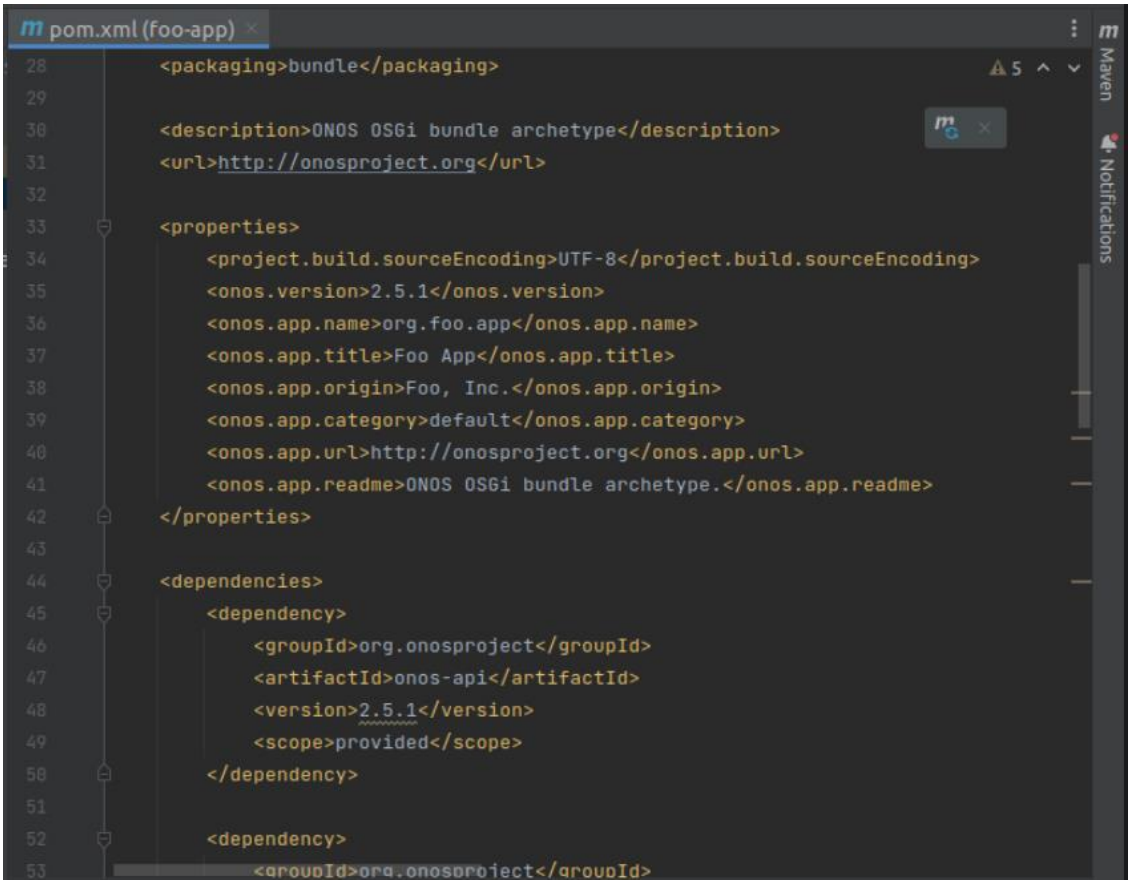

Figura D-0-4 Estructura del archivo pom.xml

Lo que se hace a continuación es instalar la aplicación dentro del controlador; para ello, primero se hace uso del comando *mvn clean install* para compilar y posterior a ello se instala con el comando *onos-app localhost install! target/foo-app-1.0-SNAPSHOT.oar* una vez realizado este proceso en el CLI de onos se podrá apreciar la nueva aplicación con el comando *app -s -a*, el cual arroja todas las aplicaciones instaladas en el controlador; en las Figura D-0-6, Figura D-0-5 y Figura D-0-7 se observa el ejemplo de cómo se hizo la respectiva ejecución de los comandos de instalación.

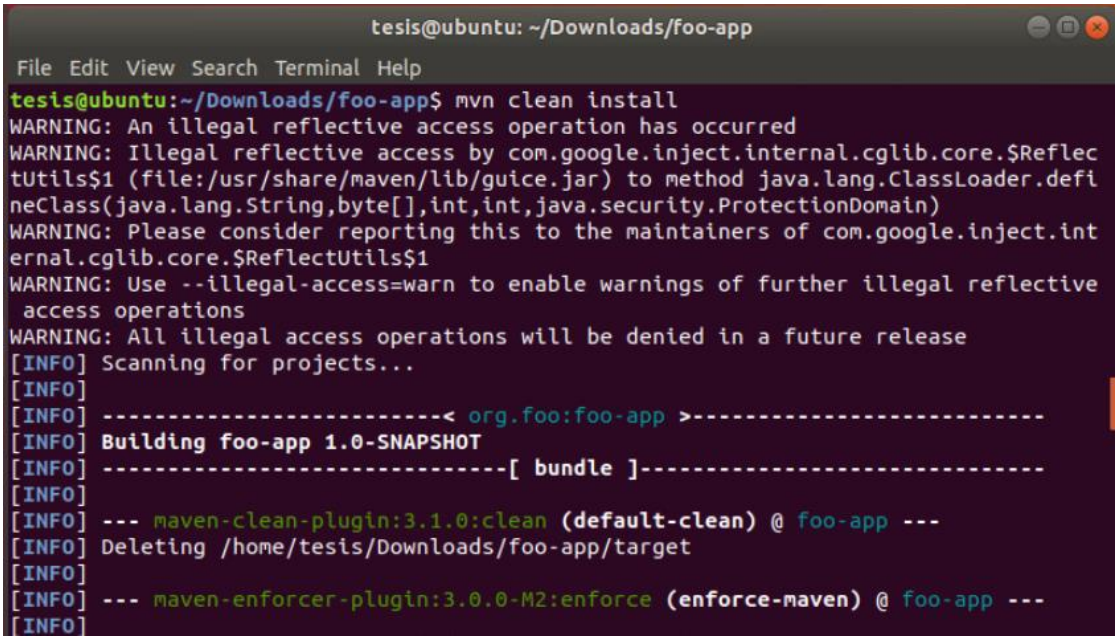

Figura D-0-6 Comando de compilación de la aplicación

Figura D-0-5 Comando de instalación de la aplicación

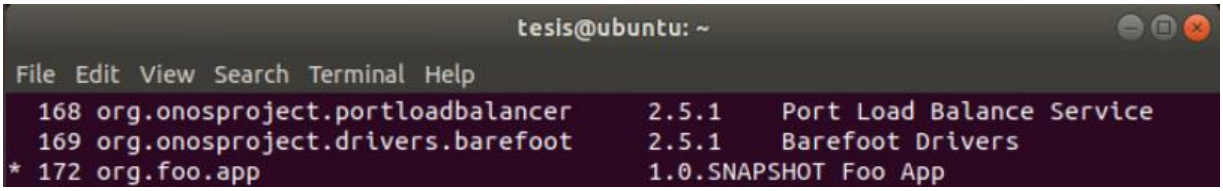

Figura D-0-7 Verificación de la instalación de la aplicación en el CLI de ONOS

Una vez verificado que todo lo anterior se haya realizado correctamente, se procede a configurar la aplicación para que sea posible agregar comandos al CLI de onos, este proceso se hace ejecutando el comando *onos-create-app cli org.foo.app foo-app 1.0.0*, esto permitirá superponer la interfaz CLI, una vez se corra se comandó, se procede a ejecutar nuevamente el comando *mvn clean install* y dado que ya se ha instalado previamente la aplicación, lo que sigue es correr el comando para reinstalarla, el cual es *onos-app localhost reinstall target/foo-app-1.0-SNAPSHOT.oar,* en las Figura D-0-8 y Figura D-0-9 se muestra este proceso; finalmente, para asegurarse de que haya quedado correctamente se ejecuta el comando *sample* en el CLI de onos y deberá arrojar un *hello world* tal como se ve en la Figura D-0-10.

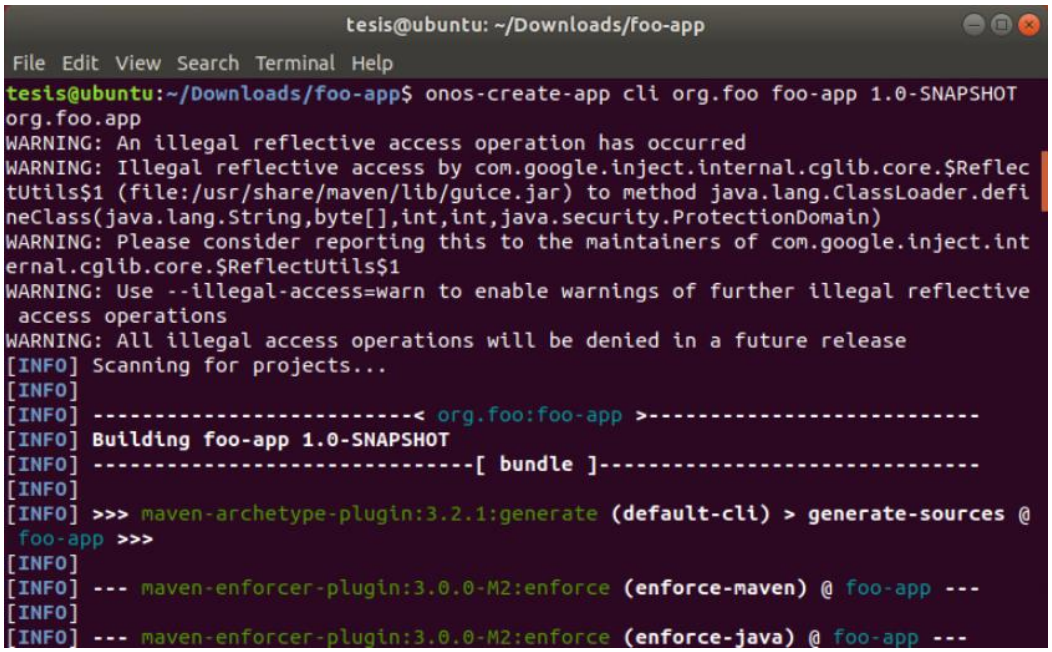

Figura D-0-8 Comando para superponer comandos de la aplicación en el CLI de ONOS

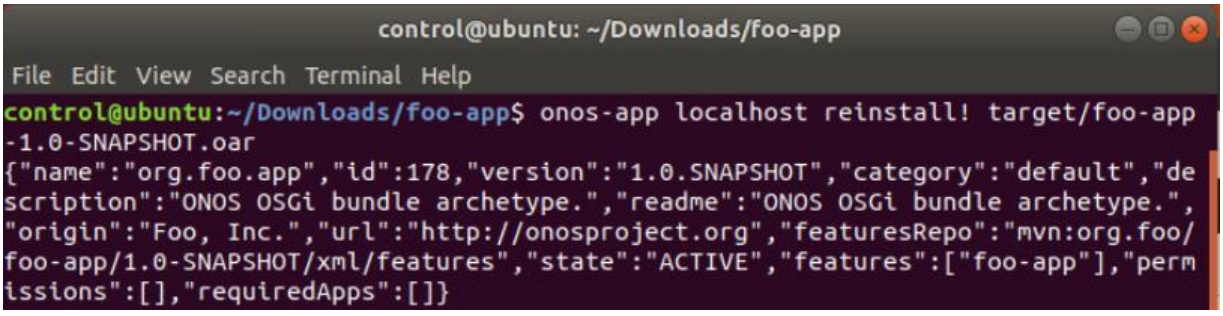

Figura D-0-9 Comando de reinstalación de la aplicación

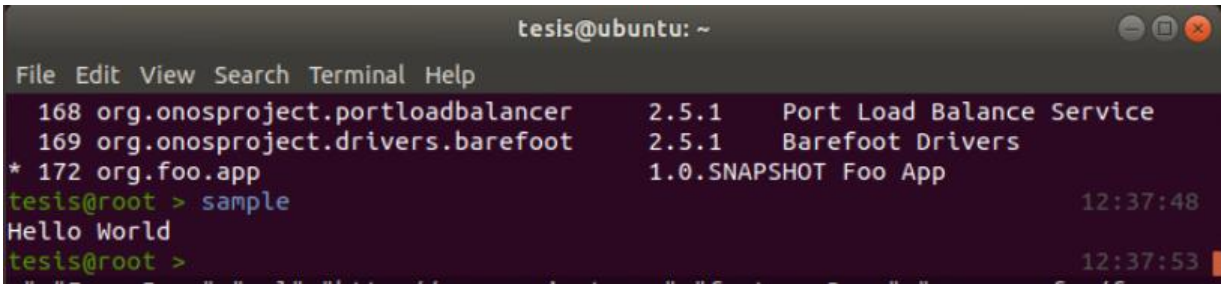

Figura D-0-10 Verificación de la configuración

Finalmente, en el directorio */Downloads/foo-app/scr/main/java/org/foo/app/AppComponent.java* se encuentra el esqueleto de la aplicación donde se debe incorporar todo lo relacionado con la lógica que se requiere para una nueva aplicación.

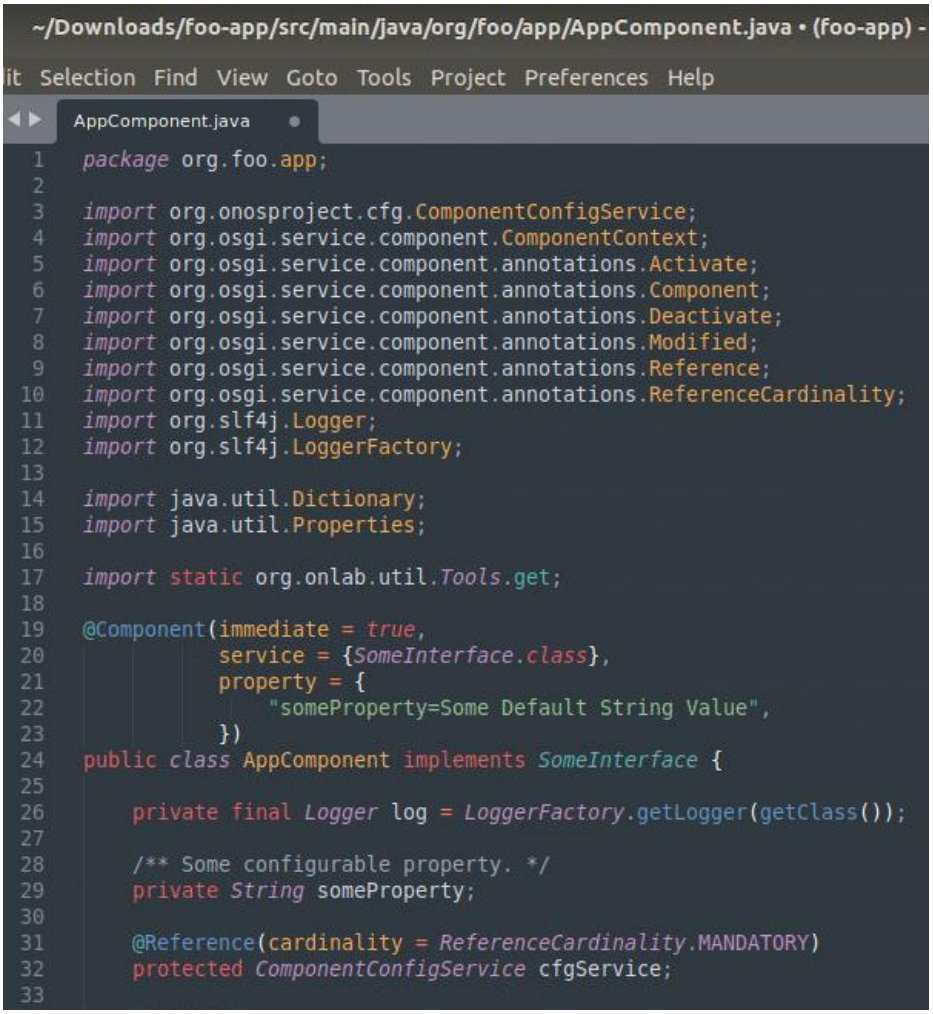

Figura D-0-11 Clase AppComponent del esqueleto de la aplicación

```
    @Activate
    protected void activate() {
        cfgService.registerProperties(getClass());
        log.info("Started");
    }

    @Deactivate
    protected void deactivate() {
        cfgService.unregisterProperties(getClass(), false);
        log.info("Stopped");
    }

    @Modified
    public void modified(ComponentContext context) {
        Dictionary<?, ?> properties = context != null ?
        context.getProperties() : new Properties();
        if (context != null) {
            someProperty = get(properties, "someProperty");
        }
        log.info("Reconfigured");
    }

    @Override
    public void someMethod() {
        log.info("Invoked");
    }

}
```

Figura D-0-12 Métodos de activación y desactivación dentro del esqueleto de la aplicación

ANEXO E. INTERFAZ GRÁFICA DEL CONTROLADOR

Otra forma de acceder al controlador es a través de la interfaz gráfica (GUI, *Graphical User interface*), para ello, se hace uso de un localizador de recursos uniforma (URL, *Uniform Resorce Locator*), el cual *http://localhost:8181/onos/ui* o *http://IP-ONOS:8181/onos/ui*, donde IP-ONOS se reemplaza por la dirección IP que tiene definido el controlador. Se debe tener en cuenta que para que esta funcione correctamente se debe verificar primero que la app (*org.onosproject.gui*) asignada este activada, generalmente esta se activa por defecto al arrancar el controlador, pero se recomienda verificar ya que sin ella no es posible acceder a la interfaz . Una vez acceda le mostrará algo similar a la Figura E-0-1, ahí debe iniciar sesión con las siguientes credenciales:

- **User: onos**
- **Password: rocks**

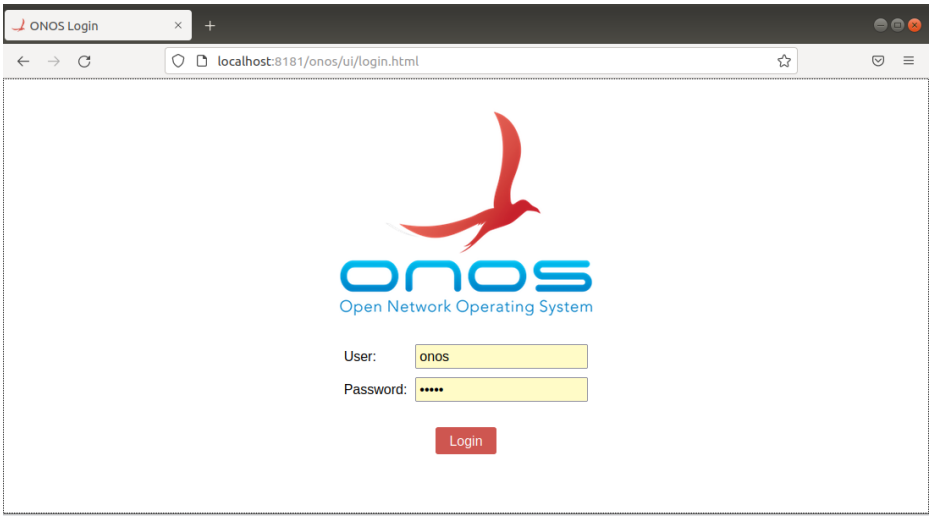

Figura E-0-1 Ventana de login de la GUI de ONOS

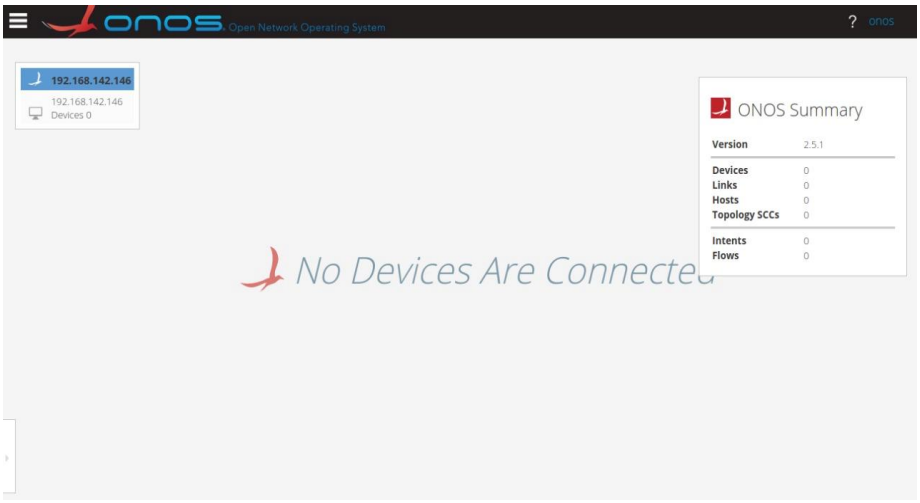

Figura E-0-2 Ventana de Inicio de la GUI de ONOS

Mientras no se halla cargado una topología en el sistema, la interfaz aparecerá en blanco como en la Figura E-0-2, una vez se cargue algún diseño de topología realizado en mininet deberá aparece en esa ventana como se puede en el ejemplo mostrado en la Figura E-0-3, le cual consta de dos switch y cinco hosts.

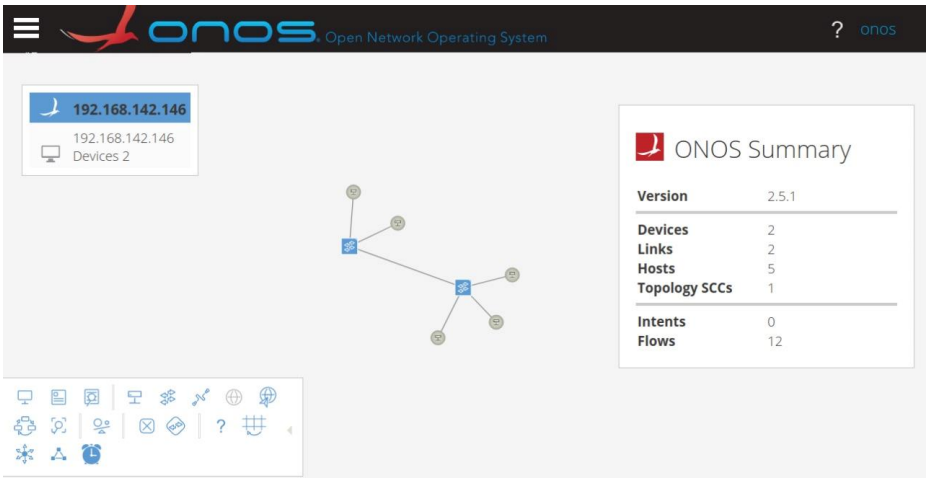

Figura E-0-3 Vista de una topología de red desde el controlador

En la parte superior izquierda de la ventana, se tiene la opción de desplegar el menú de opciones para interactuar con la interfaz, este se puede ver en la Figura E-0-4, ahí se podrá ver las aplicaciones que están disponibles y cuáles de ellas están activadas, para eso solamente se debe seleccionar a primera opción del menú que dice *Applications* y automáticamente mostrara la ventana que se muestra en la Figura

E-0-5. Ahí se observa el total de aplicaciones que se tienen disponibles, que para este caso con 168, también indica cuales están activadas, esto lo hace colocando una marca de verificación en color verde al lazo izquierdo del nombre de la aplicación, para el caso contrario, es decir si la aplicación esta inactiva a lado izquierdo del nombre de la aplicación se tendrá un recuadro de color rojo.

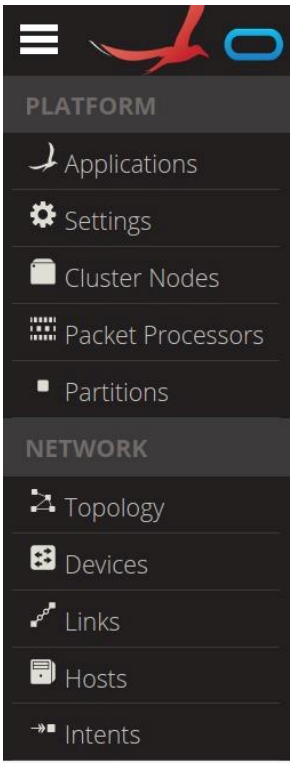

Figura E-0-4 Menú de opciones de la interfaz

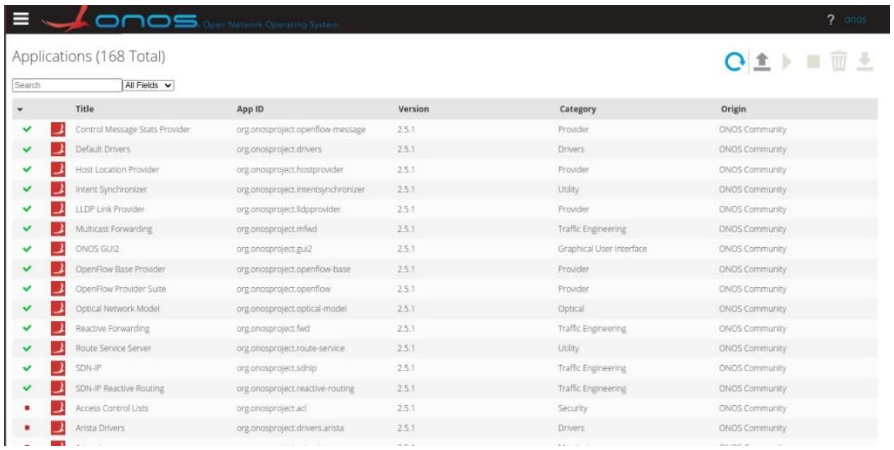

Figura E-0-5 Aplicaciones del controlador vistas desde al interfaz

De igual manera, esta interfaz permite mirar los dispositivos que se tiene conectados, los enlaces, y los hosts cada uno con sus respectivas características. Para esto solo es necesario dar clic en el menú en la opción de *Devices* y mostrara la ventana que aparece en la Figura E-0-6; si lo que se requiere es mirar los hosts conectados, solamente se da clic en la opción de Hosts del menú de opciones y mostrará algo similar a la Figura E-0-8; finalmente si se quiere ver los enlaces establecidos entre dispositivos y si están activos, se da clic en la opción Links del menú y mostrara los enlaces que se tienen como en la Figura E-0-7.

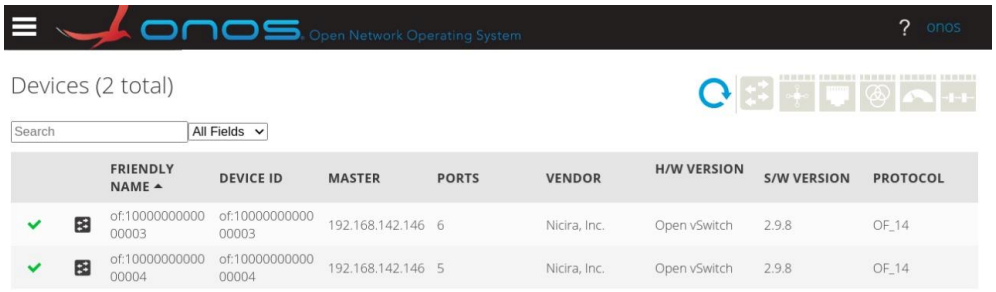

Figura E-0-6 Dispositivo conectados vistos desde la interfaz web

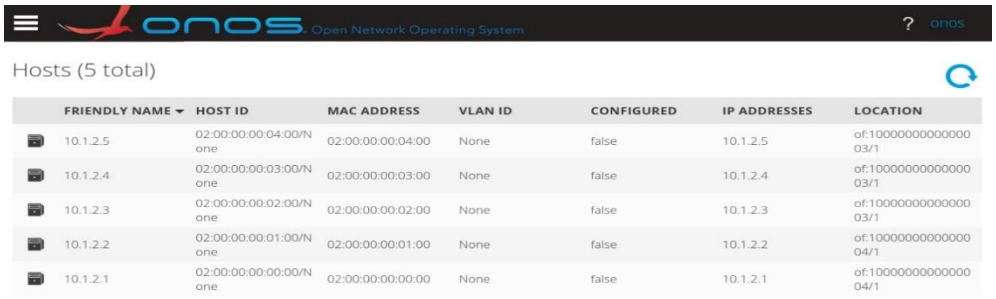

Figura E-0-8 Host vistos desde la interfaz web

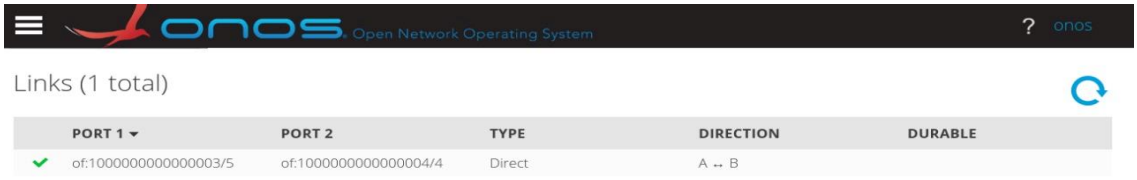

Figura E-0-7 Enlaces entre los dispositivos vistos desde la interfaz web

Ahora bien, como se mencionó anteriormente, la interfaz me permite ver las características de los dispositivos y los hosts, en la Tabla E-0-1 se lista los detalles que aparecen.

Tabla E-0-1 Lista de los detalles de lo elementos de red visto desde la interfaz web

Elemento	Característica	Descripción
Devices	ID del dispositivo	Es la cadena de caracteres que van a identificar el dispositivo
	Master	Es la dirección IP del controlador
	Ports	Corresponde al número de puertos que tiene el OVS
	Vendor	Es el fabricante del dispositivo
	H/W versión	Hace referencia al modelo del dispositivo que en este caso es un Open vSwitch
	S/W versión	Indica la versión del del OVS
	Protocolo	Indica la versión del protocolo OpenFlow con el cual se está trabajando.
Hosts	Host ID	Es el identificador del Host y se compone por la dirección MAC/VLAN a la que pertenece
	MAC Address	Es la dirección MAC asignada al terminal
	VLAN ID	En caso de que tenga más red conectadas, esto indicara a cuál de ellas está asociado.
	IP Address	Es la dirección IP que se le asigna a cada host al diseñar la topología
	Location	Se conforma por el dispositivo al que está conectado/el puerto al que está conectado.

Por otro lado, dentro de la venta de dispositivos se va a encontrar otro menú de opciones, este aparece en la parte superior derecha de la pantalla, esto se ve en la Figura E-0-9.

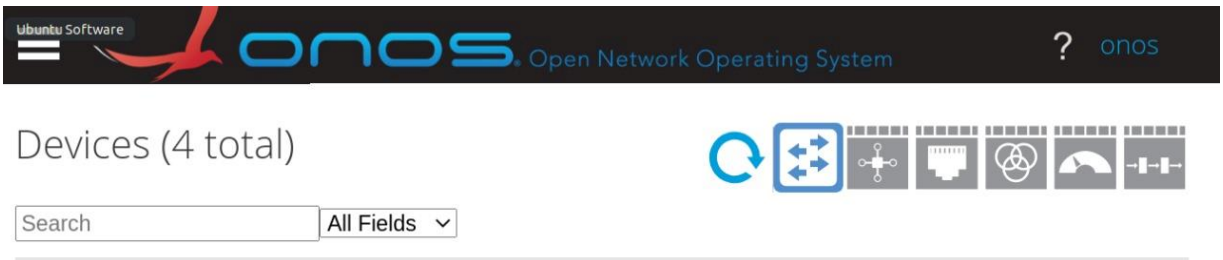

Figura E-0-9 Menú de opciones para ver el comportamiento de los dispositivos desde la interfaz web

Para realizar la descripción de estas opciones se tomarán en orden de izquierda a derecha según la imagen, tomando eso consideración se tiene:

Esta es la vista de entradas de las diferentes tablas de flujo del dispositivo seleccionado, aquí se encontrarán características como selector y prioridad que serán usados para el desarrollo de aplicaciones.

Aquí se encontrará las estadísticas de los puertos del dispositivo seleccionado, aquí se podrá ver el número de paquetes y bytes enviados y recibidos.

En esta pestaña se tiene la tabla de grupos, en caso de que se creen grupo es aquí donde se mostraran todos los datos relacionados con estos

Las otras funciones, son la vista de los meters y la configuración del pipeline que en este trabajo de grado no se usan y por ello no hace mayor referencia a ellos.

ANEXO F. COMANDOS DE UTILIDAD

Mininet

dpctl: Permite el control sobre el conmutador openflow, lo que permite agregar tablas y consultar las características de estas.

dump: Devuelve la información de los nodos de la red.

xterm y gterm: Permite abrir una ventana de comando de un nodo especifico.

intfs: Lista las interfaces de los diferentes nodos.

iperf: Con este comando se realiza una prueba simple de TCP entre dos hosts, permitiendo medir la velocidad máxima de conexión.

iperfupd: Se utiliza para realizar pruebas simples de UDP entre dos hosts.

links: Muestra el estado de todos los enlaces

net: Muestra las conexiones de la red

nodes: Lista los nodos de la red

pingall: Se utiliza para realiza run ping entre todos los nodos y devuelve el resumen de los pings realizados

pingallfull: También se emplea para realizar un ping entre todos los nodos, pero a diferencia del anterior este devuelve todos los resultados de la operación
ports: Muestra la información de los puertos y las interfaces de los diferentes switch

Onos

devices: se usa para mirar que dispositivos están controlados por el controlador

hosts: permite mirar cuales son los terminales que están conectados a los dispositivos o han enviado algún paquete

flows: Devuelve los flujos OpenFlow que se han instalado sobre los dispositivos, las diferentes aplicaciones que gestiona el comportamiento de la red

log:tail: Permite ver información sobre los procesos que se están llevando a cabo en el controlador

app actívate nombre_aplicación: Es la línea de comando utilizada para activar las aplicaciones en el controlador

app deactivate nombre_aplicación: Esta se utiliza para desactivar las aplicaciones en el controlador

ANEXO G. INSTALACIÓN DE OTRAS APLICACIONES

Se debe tener en cuenta que el desarrollo de las aplicaciones para el presente trabajo de grado se realizará en el lenguaje de programación *JAVA*; por tanto, se necesita de un entorno de desarrollo apropiado para trabajar en ellas, para tal efecto se decidió instalar la plataforma de software llamada *eclipse.* Por otro lado, ´para el diseño de la topología utilizada para la simulación se empleó el lenguaje de programación PYTHON, el entorno de desarrollo elegido para esta parte fue Atom. A continuación, se encuentran las líneas de comandos para instalar estas herramientas en el sistema operativo UBUNTU 18.04.

Instalación de Atom

:~$sudo apt install software-properties-common apt-transport-https wget
:~$wget -q https://packagecloud.io/AtomEditor/atom/gp pkey -o- | sudo apt-key add-
:~$sudo add-apt-repository "deb[arch=amd64]https://packagecloud.io/AtomEditor/atom/any/any any main"
:~$sudo apt install atom

Instalación de elipse

:~$sudo apt install default-jre
:~$sudo snap install –classic eclipse
:~$eclipse 2019-03 from Snapcrafters installed

En la Figura F-0-1 se muestra que ya están instaladas y lista para usarse las herramientas, se debe tener en cuenta que ambas herramientas consumen altos recursos del sistema y requiere de espacio en el disco.

Figura F-0-1 Verificación de la instalación de las herramientas

ANEXO H. CÓDIGOS DE SIMULACIÓN

Código de la Red

```
#!/usr/bin/python

from mininet.node import RemoteController
from mininet.log import setLogLevel, info
from mn_wifi.net import Mininet_wifi
from mn_wifi.node import Station, OVSKernelAP, UserAP
from mn_wifi.cli import CLI
from mininet.link import TCLink
from mn_wifi.link import wmediumd, WifiDirectLink
from mn_wifi.wmediumdConnector import interference
from subprocess import call
from time import sleep


def myNetwork():

    net = Mininet_wifi(topo=None,
                       build=False,
                       link=wmediumd,
                       wmediumd_mode=interference)

    info( '*** Adding controller\n' )
    onos = net.addController(name='onos',
                             controller=RemoteController,
                             ip='192.168.240.129',
                             protocol='tcp',
                             port=6653)

    info( '*** Add switches/APs\n')
    ap1 = net.addAccessPoint('ap1', cls=UserAP, ssid='ap1-ssid',
    datapath='user', channel='1', mode='n2', position='30.0,37.0,0')
    ap2 = net.addAccessPoint('ap2', cls=UserAP, ssid='ap2-ssid',
    datapath='user', channel='1', mode='n2', position='1047.0,36.0,0')
    ap3 = net.addAccessPoint('ap3', cls=UserAP, ssid='ap3-ssid',
    datapath='user', channel='1', mode='n2', position='541.0,428.0,0')
    ap4 = net.addAccessPoint('ap4', cls=UserAP, ssid='ap4-ssid',
    datapath='user', channel='1', mode='n2', position='24.0,813.0,0')
    ap5 = net.addAccessPoint('ap5', cls=UserAP, ssid='ap5-ssid',
```

```
    datapath='user', channel='1', mode='n2', position='1046.0,815.0,0')


    info( '*** Add hosts/Stations\n')
    sta1 = net.addStation('sta1', ip='10.1.0.101/24', position='120.0,14.0,0')
    sta2 = net.addStation('sta2', ip='10.1.0.102/24', position='119.0,92.0,0')
    sta3 = net.addStation('sta3', ip='10.1.0.103/24', position='77.0,131.0,0')
    sta4 = net.addStation('sta4', ip='10.1.0.104/24', position='11.0,132.0,0')
    sta5 = net.addStation('sta5', ip='10.1.0.105/24', position='959.0,21.0,0')
    sta6 = net.addStation('sta6', ip='10.1.0.106/24',
position='964.0,101.0,0')
    sta7 = net.addStation('sta7', ip='10.1.0.107/24',
position='1037.0,141.0,0')
    sta8 = net.addStation('sta8', ip='10.1.0.108/24',
position='1112.0,115.0,0')
    sta9 = net.addStation('sta9', ip='10.1.0.109/24',
position='467.0,357.0,0')
    sta10 = net.addStation('sta10', ip='10.1.0.110/24',
position='457.0,445.0,0')
    sta11 = net.addStation('sta11', ip='10.1.0.111/24',
position='490.0,528.0,0')
    sta12 = net.addStation('sta12', ip='10.1.0.112/24',
position='573.0,528.0,0')
    sta13 = net.addStation('sta13', ip='10.1.0.113/24',
position='631.0,448.0,0')
    sta14 = net.addStation('sta14', ip='10.1.0.114/24',
position='625.0,363.0,0')
    sta15 = net.addStation('sta15', ip='10.1.0.115/24',
position='40.0,729.0,0')
    sta16 = net.addStation('sta16', ip='10.1.0.116/24',
position='114.0,749.0,0')
    sta17 = net.addStation('sta17', ip='10.1.0.117/24',
position='146.0,829.0,0')
    sta18 = net.addStation('sta18', ip='10.1.0.118/24',
position='34.0,902.0,0')
    sta19 = net.addStation('sta19', ip='10.1.0.119/24',
position='117.0,913.0,0')
    sta20 = net.addStation('sta20', ip='10.1.0.120/24',
position='1057.0,731.0,0')
    sta21 = net.addStation('sta21', ip='10.1.0.121/24',
position='980.0,735.0,0')
    sta22 = net.addStation('sta22', ip='10.1.0.122/24',
position='954.0,820.0,0')
```

```
    sta23 = net.addStation('sta23', ip='10.1.0.123/24',
position='962.0,900.0,0')
    sta24 = net.addStation('sta24', ip='10.1.0.124/24',
position='1043.0,900.0,0')

    info("*** Configuring Propagation Model\n")
    net.setPropagationModel(model="logDistance", exp=3)

    info("*** Configuring wifi nodes\n")
    net.configureWifiNodes()

    info( '*** Add links\n')
    ap1ap2 = {'bw':15}
    net.addLink(ap1, ap2, 2,2, cls=TCLink , **ap1ap2)
    ap1ap4 = {'bw':15}
    net.addLink(ap1, ap4, 4,2, cls=TCLink , **ap1ap4)
    ap4ap5 = {'bw':15}
    net.addLink(ap4, ap5, 3,3, cls=TCLink , **ap4ap5)
    ap5ap2 = {'bw':15}
    net.addLink(ap5, ap2, 2,4, cls=TCLink , **ap5ap2)
    ap1ap3 = {'bw':15}
    net.addLink(ap1, ap3, 3,2, cls=TCLink , **ap1ap3)
    ap3ap2 = {'bw':15}
    net.addLink(ap3, ap2, 3,3, cls=TCLink , **ap3ap2)
    ap4ap3 = {'bw':15}
    net.addLink(ap4, ap3, 4,4, cls=TCLink , **ap4ap3)
    ap5ap3 = {'bw':15}
    net.addLink(ap5, ap3, 4,5, cls=TCLink , **ap5ap3)

    net.plotGraph(min_x=-300, max_x=1500, min_y=-300, max_y=1300)

    info( '*** Starting network\n')
    net.build()
    info( '*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info( '*** Starting switches/APs\n')
    net.get('ap1').start([onos])
    net.get('ap2').start([onos])
    net.get('ap3').start([onos])
    net.get('ap4').start([onos])
    net.get('ap5').start([onos])
```

```
    # Conexion Ap-Sta

    cmd = "iw dev {} connect {} {}"
    sta1.cmd(cmd.format("sta1-wlan0", "ap1-ssid", "02:00:00:00:18:00"))
    sta2.cmd(cmd.format("sta2-wlan0", "ap1-ssid", "02:00:00:00:18:00"))
    sta3.cmd(cmd.format("sta3-wlan0", "ap1-ssid", "02:00:00:00:18:00"))
    sta4.cmd(cmd.format("sta4-wlan0", "ap1-ssid", "02:00:00:00:18:00"))

    sta5.cmd(cmd.format("sta5-wlan0", "ap2-ssid", "02:00:00:00:19:00"))
    sta6.cmd(cmd.format("sta6-wlan0", "ap2-ssid", "02:00:00:00:19:00"))
    sta7.cmd(cmd.format("sta7-wlan0", "ap2-ssid", "02:00:00:00:19:00"))
    sta8.cmd(cmd.format("sta8-wlan0", "ap2-ssid", "02:00:00:00:19:00"))
    sta9.cmd(cmd.format("sta9-wlan0", "ap3-ssid", "02:00:00:00:1a:00"))

    sta10.cmd(cmd.format("sta10-wlan0", "ap3-ssid", "02:00:00:00:1a:00"))
    sta11.cmd(cmd.format("sta11-wlan0", "ap3-ssid", "02:00:00:00:1a:00"))
    sta12.cmd(cmd.format("sta12-wlan0", "ap3-ssid", "02:00:00:00:1a:00"))
    sta13.cmd(cmd.format("sta13-wlan0", "ap3-ssid", "02:00:00:00:1a:00"))
    sta14.cmd(cmd.format("sta14-wlan0", "ap3-ssid", "02:00:00:00:1a:00"))

    sta15.cmd(cmd.format("sta15-wlan0", "ap4-ssid", "02:00:00:00:1b:00"))
    sta16.cmd(cmd.format("sta16-wlan0", "ap4-ssid", "02:00:00:00:1b:00"))
    sta17.cmd(cmd.format("sta17-wlan0", "ap4-ssid", "02:00:00:00:1b:00"))
    sta18.cmd(cmd.format("sta18-wlan0", "ap4-ssid", "02:00:00:00:1b:00"))
    sta19.cmd(cmd.format("sta19-wlan0", "ap4-ssid", "02:00:00:00:1b:00"))

    sta20.cmd(cmd.format("sta20-wlan0", "ap5-ssid", "02:00:00:00:1c:00"))
    sta21.cmd(cmd.format("sta21-wlan0", "ap5-ssid", "02:00:00:00:1c:00"))
    sta22.cmd(cmd.format("sta22-wlan0", "ap5-ssid", "02:00:00:00:1c:00"))
    sta23.cmd(cmd.format("sta23-wlan0", "ap5-ssid", "02:00:00:00:1c:00"))
    sta24.cmd(cmd.format("sta24-wlan0", "ap5-ssid", "02:00:00:00:1c:00"))
    sleep(2)


    info( '*** Post configure nodes\n')

    CLI(net)
    net.stop()


if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()
```

Código de la Aplicación

AppComponent.java

```
package org.foo.app;

import org.onosproject.cfg.ComponentConfigService;
import org.osgi.service.component.ComponentContext;
import org.osgi.service.component.annotations.Activate;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Deactivate;
import org.osgi.service.component.annotations.Modified;
import org.osgi.service.component.annotations.Reference;
import org.osgi.service.component.annotations.ReferenceCardinality;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Dictionary;
import java.util.Properties;

import static org.onlab.util.Tools.get;

@Component(immediate = true,
           service = {SomeInterface.class},
           property = {
               "someProperty=Some Default String Value",
           })
public class AppComponent implements SomeInterface {

    private final Logger log = LoggerFactory.getLogger(getClass());

    private String someProperty;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected ComponentConfigService cfgService;

    @Activate
    protected void activate() {
        cfgService.registerProperties(getClass());
        log.info("Started");
    }

    @Deactivate
```

```
    protected void deactivate() {
        cfgService.unregisterProperties(getClass(), false);
        log.info("Stopped");
    }

    @Modified
    public void modified(ComponentContext context) {
        Dictionary<?, ?> properties = context != null ?
context.getProperties() : new Properties();
        if (context != null) {
            someProperty = get(properties, "someProperty");
        }
        log.info("Reconfigured");
    }

    @Override
    public void someMethod() {
        log.info("Invoked");
    }

}
```

AppCommand.java

```
package org.foo.app;

import org.apache.karaf.shell.api.action.Command;
import org.apache.karaf.shell.api.action.lifecycle.Service;
import org.onosproject.cli.AbstractShellCommand;
import org.onosproject.net.flow.FlowRuleService;
import org.onosproject.net.device.DeviceService;
import org.onosproject.net.device.PortStatistics;
import org.onosproject.net.Device;
import org.onosproject.net.DeviceId;
import org.onosproject.net.Host;
import org.onosproject.net.host.HostProbingService;
import org.onosproject.net.host.HostService;
import org.onosproject.net.Port;
import org.onosproject.net.PortNumber;
import static org.onosproject.cli.net.DevicesListCommand.getSortedDevices;
import static org.onosproject.net.HostId.hostId;

import static org.onosproject.net.DeviceId.deviceId;
import java.util.Comparator;
import java.util.concurrent.TimeUnit;
import java.util.List;
import java.util.ArrayList;
import com.google.common.collect.Lists;
import org.onlab.packet.MacAddress;
import org.onlab.packet.VlanId;
import org.onlab.packet.Ethernet;
import org.onosproject.net.packet.PacketContext;
import org.onosproject.net.packet.PacketProcessor;
import org.onosproject.net.packet.PacketService;
import org.osgi.service.component.annotations.Reference;
import org.osgi.service.component.annotations.ReferenceCardinality;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.onosproject.core.ApplicationId;
import org.onosproject.core.CoreService;
import org.onosproject.net.flow.DefaultTrafficSelector;
import org.onosproject.net.flow.DefaultTrafficTreatment;
import org.onosproject.net.flow.TrafficSelector;
import org.onosproject.net.flow.TrafficTreatment;
import org.onosproject.net.flowobjective.FlowObjectiveService;
import org.onosproject.net.HostId;
```

```java
import java.util.Set;
import org.onosproject.net.Path;
import org.onosproject.net.flowobjective.DefaultForwardingObjective;

import org.onosproject.net.flowobjective.ForwardingObjective;
import org.onosproject.net.topology.Topology;
import org.onosproject.net.topology.TopologyService;
import static org.foo.app.OsgiPropertyConstants.FLOW_PRIORITY_DEFAULT;
import static org.foo.app.OsgiPropertyConstants.FLOW_TIMEOUT_DEFAULT;

@Service
@Command(scope = "onos", name = "sample",
         description = "Sample Apache Karaf CLI command")
public class AppCommand extends AbstractShellCommand {

   private static long t_s = 5;

   public static boolean flag = true;
   public static ArrayList<Boolean> flag_time = new ArrayList<Boolean>();
   public static ArrayList<Integer> t_c = new ArrayList<Integer>();
   public static ArrayList<Float> PktRx = new ArrayList<Float>();
   public static ArrayList<Float> PktTx= new ArrayList<Float>();
   public static ArrayList<Float> PktRxError = new ArrayList<Float>();
   public static ArrayList<Float> PktTxError = new ArrayList<Float>();
   public static ArrayList<Float> BytesRx = new ArrayList<Float>();
   public static ArrayList<Float> BytesTx = new ArrayList<Float>();
   public static ArrayList<Float> PktRxDrp = new ArrayList<Float>();
   public static ArrayList<Float> PktTxDrp = new ArrayList<Float>();

   private int flowTimeout = FLOW_TIMEOUT_DEFAULT;
   private int flowPriority = FLOW_PRIORITY_DEFAULT;
   private ApplicationId appId;

   @Reference(cardinality = ReferenceCardinality.MANDATORY)
   protected CoreService coreService;

   @Reference(cardinality = ReferenceCardinality.MANDATORY)
   protected DeviceService deviceService;

   @Reference(cardinality = ReferenceCardinality.MANDATORY)
   protected HostService hostService;

   @Reference(cardinality = ReferenceCardinality.MANDATORY)
   protected HostProbingService hostProbingService;
```

```
   @Reference(cardinality = ReferenceCardinality.MANDATORY)
   protected FlowObjectiveService flowObjectiveService;

   @Reference(cardinality = ReferenceCardinality.MANDATORY)
   protected FlowRuleService flowRuleService;

   @Reference(cardinality = ReferenceCardinality.MANDATORY)
   protected PacketService packetService;

   @Reference(cardinality = ReferenceCardinality.MANDATORY)
   protected PacketContext context;

   @Reference(cardinality = ReferenceCardinality.MANDATORY)
   protected TopologyService topologyService;

    int priority = 65000;

    @Override
   protected void doExecute() {

    appId = get(CoreService.class).getAppId("org.foo.app");

    DeviceService deviceService = get(DeviceService.class);

      int numHosts = get(HostService.class).getHostCount();
      Host HosDts =
get(HostService.class).getHost(hostId("02:00:00:00:13:00/None"));
      Host HosSrt =
get(HostService.class).getHost(hostId("02:00:00:00:00:00/None"));
      Device deviceId =
get(DeviceService.class).getDevice(deviceId("of:000000acfec00001"));
      installRule(deviceId, HosDts, HosSrt );
        int Nhost_Real = numHosts;
        Topology topology = get(TopologyService.class).currentTopology();

        Set<Path> paths =get(TopologyService.class).getPaths(topology,
                    HosSrt.location().deviceId(),
                    HosDts.location().deviceId());

        System.out.println("\n paths"+paths);

        if(flag) {
```

```
         float aux = 0;
          for(int i=1;i<=(25);i++) {

             BytesTx.add(aux);
             PktRx.add(aux);
             PktTx.add(aux);
             BytesRx.add(aux);
             PktTxError.add(aux);
             PktRxError.add(aux);
             PktRxDrp.add(aux);
             PktTxDrp.add(aux);

          }
        }
        flag = false;

          int i=1;
          int cont = 1;
          for (Device d : getSortedDevices(deviceService)) {
             if (cont<=Nhost_Real) {
                printPortStats(d.id(),
                     deviceService.getPortDeltaStatistics(d.id()),cont,i);
                      }
             cont++;
                  }
           Qlearning obj = new Qlearning();

                      obj.run(BytesTx,BytesRx);
                      obj.printResult();
                      obj.showPolicy();
          try {
             Thread.sleep(t_s);
          } catch (InterruptedException ignored) {
         }
    }
    private void printPortStats(DeviceId deviceId, Iterable<PortStatistics>
portStats, int Ndevice, int sequence) {

        int cont2=0;
        cont2 = (Ndevice*5)-5+cont2;
        List<Port> ports = get(DeviceService.class).getPorts(deviceId);
     for (Port p: ports) {
            if(p.isEnabled()){
```

```
                PortStatistics traffic =
get(DeviceService.class).getDeltaStatisticsForPort(deviceId, p.number());
                System.out.println("  \n trafico del puerto "+traffic);
            }
        }
    for (PortStatistics stat : sortByPort(portStats)) {

              float pktRx = stat.packetsReceived();
              float pktTx = stat.packetsSent();
              float bytesRx = stat.bytesReceived();
              float bytesTx = stat.bytesSent();
              float pktRxError= stat.packetsRxErrors();
              float pktTxError= stat.packetsTxErrors();
              float pktRxDrp = stat.packetsRxDropped();
              float pktTxDrp = stat.packetsTxDropped();
              float i=1;

              if (bytesRx==0){
                bytesRx=i;
              }
              if (bytesTx==0){
                bytesTx=i;
              }
              if (pktRx==0){
                pktRx=i;
              }
              if (pktTx==0){
                pktTx=i;
              }
              if (bytesRx==0){
                bytesRx=i;
              }
              if (pktTxError==0){
                pktTxError=i;
              }
              if (pktRxError==0){
                pktRxError=i;
              }
              if (pktRxDrp==0){
                pktRxDrp=i;
              }
              if (pktTxDrp==0){
                pktTxDrp=i;
              }
```

```
                float Tx = pktTxDrp*pktTxError*bytesTx;
                float Rx = pktRxDrp*pktRxError*bytesRx;
                BytesTx.set(cont2,Tx);
                BytesRx.set(cont2,Rx);
                cont2++;
            }
        }

        private static List<PortStatistics> sortByPort(Iterable<PortStatistics>
portStats) {
            List<PortStatistics> portStatsList = Lists.newArrayList(portStats);
            portStatsList.sort(Comparator.comparing(ps ->
ps.portNumber().toLong()));
            return portStatsList;
        }

        private void installRule( Device deviceId,Host HosSrt, Host HosDts) {

            MacAddress macAddressSrc= HosSrt.mac();
            MacAddress macAddressDts= HosDts.mac();

            TrafficSelector.Builder selectorBuilder =
DefaultTrafficSelector.builder();
            long puerto = 1;
            selectorBuilder.matchEthSrc(macAddressSrc)
            .matchEthDst(macAddressDts);
            TrafficTreatment treatment;

            treatment = DefaultTrafficTreatment.builder()
                            .setOutput(PortNumber.portNumber(puerto))
                            .build();

            ForwardingObjective forwardingObjective =
DefaultForwardingObjective.builder()
            .withSelector(selectorBuilder.build())
            .withTreatment(treatment)
            .withPriority(flowPriority)
            .withFlag(ForwardingObjective.Flag.VERSATILE)
            .fromApp(appId)
            .makeTemporary(flowTimeout)
            .add();
        }
}
```

Qlearning.java

```
package org.foo.app;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;

import javax.ws.rs.GET;

public class Qlearning {
    final DecimalFormat df = new DecimalFormat("#.##");

    final double alpha = 0.1;
    final double gamma = 0.9;
    final int ap1 = 0;
    final int ap2 = 1;
    final int ap3 = 2;
    final int ap4 = 3;
    final int ap5 = 4;

    final int statesCount = 5;
    final int[] states = new int[]{ap1,ap2, ap3, ap4, ap5};

    int[][] R = new int[statesCount][statesCount];
    double[][] Q = new double[statesCount][statesCount];
    int[][] portTx = new int[statesCount][statesCount];

    int[] actionsFromAp1 = new int[] { ap2, ap3, ap4 };
    int[] actionsFromAp2 = new int[] { ap1, ap3, ap5 };
    int[] actionsFromAp3 = new int[] { ap1, ap2, ap4, ap5 };
    int[] actionsFromAp4 = new int[] { ap1, ap3, ap5 };
    int[] actionsFromAp5 = new int[] { ap3, ap4, ap2 };

    int[][] actions = new int[][] { actionsFromAp1, actionsFromAp2,
actionsFromAp3,
            actionsFromAp4, actionsFromAp5};

    String[] stateNames = new String[] { "AP1", "AP2", "AP3", "AP4", "AP5" };

    public Qlearning() {
        init();
    }
```

```
    public void init() {

        R[ap2][ap5] = 100;
        R[ap3][ap5] = 100;
        R[ap4][ap5] = 100;

        portTx[ap1][ap2]=1;
        portTx[ap1][ap3]=2;
        portTx[ap1][ap4]=3;

        portTx[ap2][ap1]=6;
        portTx[ap2][ap3]=7;
        portTx[ap2][ap5]=8;

        portTx[ap3][ap1]=11;
        portTx[ap3][ap2]=12;
        portTx[ap3][ap4]=13;
        portTx[ap3][ap5]=14;

        portTx[ap4][ap1]=16;
        portTx[ap4][ap5]=17;
        portTx[ap4][ap3]=18;

        portTx[ap5][ap2]=21;
        portTx[ap5][ap4]=22;
        portTx[ap5][ap3]=23;

    }

     void run(ArrayList<Float> BytesTx,ArrayList<Float> BytesRx) {

        Random rand = new Random();
        for (int i = 0; i < 1000; i++) {
            int state = rand.nextInt(statesCount);
            int[] actionsFromState = actions[state];
            int index = rand.nextInt(actionsFromState.length);
            int action = actionsFromState[index];
            int nextState = action;
            double q = Q(state, action);
            double maxQ = maxQ(nextState);
            int r = R(state, action);
            int portTx = portTx(state, action);
            int portRx = portTx( action,state);
            float costo ;
```

```
            costo= (1000000)/(BytesTx.get(portTx)*BytesRx.get(portRx));
            double value = (q + alpha * costo*(r + gamma * maxQ - q));
            setQ(state, action, value);
            state = nextState;

        }
    }

    private int portTx(int state, int action) {
        return portTx[state][action];
    }

    double maxQ(int s) {
        int[] actionsFromState = actions[s];
        double maxValue = Double.MIN_VALUE;
        for (int i = 0; i < actionsFromState.length; i++) {
            int nextState = actionsFromState[i];
            double value = Q[s][nextState];

            if (value > maxValue)
                maxValue = value;
        }
        return maxValue;
    }

    int policy(int state) {
        int[] actionsFromState = actions[state];
        double maxValue = Double.MIN_VALUE;
        int policyGotoState = state;
        for (int i = 0; i < actionsFromState.length; i++) {
            int nextState = actionsFromState[i];
            double value = Q[state][nextState];

            if (value > maxValue) {
                maxValue = value;
                policyGotoState = nextState;
            }
        }
        return policyGotoState;
    }

    double Q(int s, int a) {
        return Q[s][a];
    }
```

```
    void setQ(int s, int a, double value) {
        Q[s][a] = value;
    }

    int R(int s, int a) {
        return R[s][a];
    }

    void printResult() {
        System.out.println("Print result");
        for (int i = 0; i < Q.length; i++) {
            System.out.print("out from " + stateNames[i] + ":  ");
            for (int j = 0; j < Q[i].length; j++) {
                System.out.print(df.format(Q[i][j]) + " ");
            }
            System.out.println();
        }
    }

    void showPolicy() {
        System.out.println("\nshowPolicy");
        for (int i = 0; i < states.length; i++) {
            int from = states[i];
            int to =  policy(from);
            System.out.println("from "+stateNames[from]+" go to
"+stateNames[to]);
        }
    }

}
```

Anexo I. ACTAS DE RETROALIMENTACIÓN DE SCRUM

	ACTA DE REUNION PLANEACION SPRINT 1	001

1. Información General

Fecha de Realización: 03 de septiembre de 2021 **Numero de Sprint:** 1 **Asistentes a la reunión:** Laura Camila Artunduaga, Jhon Hamilton Charo, Catalina Muñoz

2. Objetivos de la reunión

Se trataron los siguientes temas

- Se da inicio formalmente a la etapa de diseño del proyecto
- Se definen los roles de cada participante del equipo de trabajo
- Se define el plan de seguimiento del proyecto
- Se establecen las herramientas y tecnologías a utilizar para el desarrollo del proyecto
- Se definen las actividades para Sprint 1

2.1. Inicio del proyecto

Se dio inicio al diseño del proyecto que permitirá dar respuesta a la pregunta de investigación planteada.

2.2. Definición de roles dentro del Proyecto

Se establecieron los siguientes roles para cada integrante del equipo

Product Owner: Catalina Muñoz
Scrum Master: Laura Artunduaga
Equipo de Trabajo: Jhon Hamilton Charo, Laura Camila Artunduaga

2.3. Planeación y seguimiento del proyecto

Se acordó una intensidad de trabajo de lunes a viernes con 6 horas de trabajo por cada miembro del equipo.

El seguimiento se realizará por medio del cumplimiento de actividades o tareas, las cuales una vez definidas se estableció su respectivo cronograma por medio de un diagrama de GANTT.

2.4. Definición de las herramientas y tecnologías.

Dado que es un proyecto de investigación, solo se contará con Back end y despliegue, los cuales se desarrollarán con los siguientes lenguajes de programación:

- Para el Back-end se trabaja con Python y java
- Para el despliegue se tendrá ONOS

2.5. Definición de actividades para el sprint 1

Se definieron las siguientes actividades

- Instalación de las herramientas de simulación
- Instalación del controlador
- Búsqueda de los mecanismos de ML

	Acta de reunión Planeación Sprint 2	002

1. Información General

Fecha de realización 17 de septiembre de 2021 **Número del Sprint:** 2 **Asistentes a la reunión:** Laura Artunduaga, Jhon Hamilton Charo Orozco, Catalina Muñoz

2. Objetivos de la reunión

Los temas tratados fueron los siguientes

- Entrega de las actividades del sprint 1
- Obstáculos encontrados en la realización de las actividades del sprint 1
- Seguimiento de las actividades
- Definición de las actividades del sprint 2

2.1. Entrega de actividades del Sprint 1

- Se instalo de forma correcta la herramienta VMware, la cual facilitara el manejo de la máquina virtual, la cual se configuro correctamente con el sistema operativo indicado.
- Se obtuvo una entrega positiva de la instalación de la herramienta de emulación de la red.
- Se realizo la instalación del controlador, sin embargo, no se tenía acceso.
- Se entrego un resumen con los posibles mecanismos a implementar para el control de tráfico.

2.2. Obstáculos Encontrados

Durante el desarrollo de las actividades se encontraron las siguientes dificultades

Al finalizar a la instalación del controlador, este solicitaba una clave para ingresar a la ventana de comandos; credenciales a las cuales no se tenía acceso puesto que no se conocían.

Inicialmente el quipo en cual se instaló la máquina virtual no contaba con suficientes recursos; y esto hacía que se bloqueara o se apagara el quipo en el cual se estaba trabajando

2.3. Seguimiento de las Actividades

Para dar solución a los problemas encontrados con las actividades planteadas en el sprint 1 se plantean las siguientes soluciones

- Buscar una asesoría en el foro internacional de controlador, para que alguien que también este trabajando con este y haya solucionado el inconveniente con oriente sobre cómo proceder.

- Se solicita acceso al servidor de la universidad para contar con más recursos y así no tener problemas en un futuro.

2.4. Definición Actividades para el Sprint 2

- diseño de la arquitectura de red
- diseño de la topología de la red
- selección del algoritmo ML

	Acta de reunión Sprint 3	003

1. Información General

Fecha de realización
01 de octubre de 2021
Numero de Sprint: 3
Asistentes a la reunión: Laura Artunduaga, Jhon Charo, Mc Catalina Muñoz

2. Objetivos de la reunión

Los temas para tratar fueron los siguientes

- Entrega de actividades de sprint 1
- Seguimiento y entrega de las actividades del Sprint 2
- Definición de las actividades del sprint 3

2.1. Actividades del sprint 1

Una vez recibida la orientación se logró la correcta instalación del controlador ONOS y el acceso al mismo tanto por la interfaz web como por medio del CLI.

Una vez se tuvo acceso al servidor de la universidad, se procedió a instalar nuevamente todas las herramientas en él.

Con esto se dar por entregadas todas las actividades correspondientes al sprint 1

2.2. Seguimiento y entrega de las actividades del Sprint 2

Una vez revisada las distintas arquitecturas de una red IoT se procede a seleccionar la de 3 niveles para realizar la incorporación con la tecnología SDN. Se recomendó ajustar la arquitectura dado que se trabaja con un sistema totalmente emulado.

Se hizo el diseño inicial de la topología la cual incluía una distribución de los 24 equipos terminales y dos puntos de acceso. Sin embargo, se encontró que solo dos puntos de acceso no serían suficientes para la cantidad de dispositivos. Por tanto, se debe ajustar la cantidad de dispositivos y la distribución.

Se buscaron los algoritmos supervisados Random forest de clasificación y regresión para implementar el control tráfico por medio de la clasificación de paquetes. Dado que el controlador esta desarrollado bajo en lenguaje de JAVA, es necesario que el algoritmo este desarrollado en el mismo lenguaje para poder incorporarlo al núcleo de ONOS por medio de una aplicación; teniendo en cuenta ello se be buscar la forma de realizar la integración entre JAVA y Python o buscar un modelo de algoritmo ML desarrollado en mismo lenguaje del controlador.

2.3. Definición de actividades del sprint 3

- Soluciones a las observaciones realizadas a las actividades del sprint 2
- Implementación de la red en el emulador
- Integración de la red con el controlador

	Acta de Reunión Sprint 4	004

1. Información General

Fecha de Realización 15 de octubre de 2021 **Número del sprint:** 4 **Asistentes a la reunión:** Catalina Muñoz, Jhon Hamilton Charo, Laura Camila Artunduaga

2. Objetivos de la reunión

Los temas tratados en la reunión fueron los siguientes:

- Propuestas para dar respuesta a las observaciones realizadas anteriormente
- Seguimiento de las actividades del sprint 3
- Seguimiento del comportamiento de las herramientas instaladas
- Definición de actividades para sprint 4

2.1. Propuestas para dar respuesta a las observaciones realizadas anteriormente

Como se trabaja con un entorno totalmente simulado, al realizar los ajustes queda de la siguiente manera: en la primera capa estarán ubicados todos los dispositivos openflow simulados, en la segunda capa se tendrá en controlador el cual será quien administre la red y finamente se tiene la capa de aplicación donde están las aplicaciones IoT, para este trabajo solo se trabaja con en las capas uno y dos debido que no se tendrán aplicaciones.

Al considerar un ambiente tipo familiar mediano se estableció aumentar los puntos de acceso a 5 y realizar una distribución uniforme de los equipos terminales por punto de acceso.

En cuando al algoritmo dadas las recomendaciones dada, se buscó una forma de realizar la integración de los lenguajes de programación. Sin embargo, las soluciones encontradas no eran lo suficientemente óptimas para el desempeño del sistema, adicional a ello por recomendación de otros usuarios que también trabajaron con este controlador, se decidió a buscar la forma de implementar el algoritmo ML dentro de una aplicación en el núcleo de ONOS, por consiguiente, se debe buscar un esquema de algoritmo ML desarrollado en JAVA y que se puede adaptar a los requerimientos del proyecto.

2.2. Seguimiento de las Actividades Sprint 3

Se implemento la red con los cambios realizados con respecto a la cantidad de dispositivos y se corrió en el sistema, adicional a ello se logró la correcta comunicación de la red con el controlador.

2.3. Seguimiento del comportamiento de las herramientas

Se estuvo realizando el seguimiento al controlador, se encontró que este en ocasiones se dañaba; sin embargo, aún no se sabe el porqué, la única solución que se encontró para ello, fue volverlo a instalar de cero.

2.4. Actividades del Sprint 4

- Seguimiento de las actividades de los sprint 2 y 3
- Implementación del generador de tráfico en la red
- Definición del algoritmo final
- Comienzo de la implementación de la aplicación en el controlador

	Acta de Reunión Sprint 5	005

1. Información General

Fecha de realización 29 de octubre de 2021 **Número del sprint:** 5 **Asistentes a la reunión:** Catalina Muñoz, Jhon Hamilton Charo, Laura Camila Artunduaga

2. Objetivo de la reunión

- Seguimiento de las actividades del sprint 4
- Actividades del Sprint 5

2.1. Seguimiento de las Actividades del Sprint 4

Después de unas pruebas realzadas con el algoritmo de la red diseñado, se encontraron algunos problemas de enrutamiento, para corregir esto, se agregaron

algunas líneas de código para establecer la comunicación entre puntos de acceso de manera fija y para dejar fijo la conexión de equipos terminales a sus respectivos AP, con estos cambios se da por finalizada la red y se entrega funcionando correctamente.

Implementación del generador de trafico

Después de revisar las distintas herramientas que se puede usar para generar tráfico en mininet, se seleccionó usar el D-ITG el cual se instaló y configuro de forma correcta y ya se encuentra funcionando para las pruebas que se vayan a realizar.

Algoritmo final

Luego de una búsqueda exhaustiva, y de no encontrar un algoritmo supervisado en lenguaje JAVA, y dado que hizo un cambio en el enfoque para aplicar el control de tráfico, nos inclinamos por implementar un algoritmo reforzado, que permita abordar de una mejor manera el control de tráfico evitando la congestión en la red. Esto da por finalizada esta actividad y a partir de aquí se trabajará con el algoritmo Q-learning, y del cual si se tiene un esquema en lenguaje JAVA para adaptarlo al proyecto.

Implementación de la Aplicación

Ya con el tipo de algoritmo definido, ahora se debe comenzar con la implementación de la aplicación para poder hacer la incorporación al núcleo del controlador. Siguiendo con una pequeña guía se consigue la completa instalación de la plantilla de la aplicación para iniciar la integración del algoritmo e instalarla en el núcleo de onos.

2.2. Actividades del sprint 5

- Prueba del algoritmo en un compilador JAVA
- Comienzo de la incorporación del algoritmo a la aplicación