ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

IMPLEMENTACIÓN DE UNA SOLUCIÓN PARA EL PROBLEMA SHARED BOTTLENECK DEL PROTOCOLO MP-TCP UTILIZANDO SDN SOBRE UNA INFRAESTRUCTURA DE COMPUTACIÓN EN LA NUBE

TESIS DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE MAGÍSTER EN CONECTIVIDAD Y REDES DE TELECOMUNICACIONES

FERNANDO VINICIO BECERRA CAMACHO

fernandobecerracv@gmail.com

DIRECTOR: IVÁN MARCELO BERNAL CARRILLO Ph.D.

ivan.bernal@epn.edu.ec

Quito, julio 2018

AVAL

Certifico que el presente trabajo fue desarrollado por Fernando Vinicio Becerra Camacho, bajo mi supervisión.

IVÁN MARCELO BERNAL CARRILLO

DECLARACIÓN DE AUTORÍA

Yo, Fernando Vinicio Becerra Camacho, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

FERNANDO VINICIO BECERRA CAMACHO

DEDICATORIA

Esta tesis de maestría se la dedico a toda mi familia, quienes siempre han estado pendientes de mí; y en especial a mis Padres, quienes han sido el mejor ejemplo que he tenido y que con su ejemplo me han sabido inculcar todos los valores que tengo.

AGRADECIMIENTO

El primer agradecimiento es a Dios por ayudarme en todo lo que he alcanzado hasta el momento, sin su ayuda nada de esto hubiese sido posible. Después a mi familia, quienes siempre se ha preocupado por mí y me han apoyado incondicionalmente en especial a mis Padres, Sebas, Anita y Gogo. A mis tíos, quienes siempre estuvieron pendientes Gonza, Susy, Pachu, Moni, Marcelo, Amparito y a los que se les quiso mucho pero ya no están ya con nosotros.

También deseo agradecer sobre todo al Doctor Iván Bernal director de esta tesis, quien me ha ayudado mucho y ha sido un ejemplo para mí dentro y fuera de la universidad, quien considero como una persona Íntegra y que tiene muchos conocimientos.

También, otra persona que agradezco mucho es al Ingeniero David Mejía, que me ha ayudado en muchas partes de la tesis, como también conversando y armando legos.

Quiero agradecer a mis amigos que siempre estuvieron pendientes en especial a los Juniors de Boca (Multi, Chumpa, Esclavo, Volteo, Lucho, Lui, Pudule, Robert, Quaker, Dora y Pancho), Gato, Pichón, Robert, Gaby, Dani, Aldrin, Master Robert Guachi, Súper Ingeniero de Dispo, David y Wolverine.

Gracias Totales

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGOS

ÍNDICE DE TABLAS

RESUMEN

En la actualidad los usuarios finales tanto de dispositivos fijos como móviles requieren grandes velocidades de conexión, por este motivo fue creado un nuevo protocolo denominado MP-TCP el cual solventa este problema cuando el dispositivo tenga más de una interfaz de red. Pero cuando el dispositivo tiene una sola interfaz de red se produce cuellos de botella compartido los cuales limitan la utilización de la infraestructura de red y reduce la velocidad de transmisión. En esta tesis se plantea solventar el problema de los cuellos compartidos en MP-TCP mediante la utilización de SDN específicamente con el controlador OpenDayLight. Cuando ya exista la solución para los cuellos de botella compartidos en OpenDayLight se creará una función de red mediante DevOps que facilite la implementación en cualquier ambiente de forma rápida y probada. Además se plantea la instalación de OpenStack para la creación de una nube en la que se desplegará la función de red virtualizada. Las pruebas se realizaron tanto con equipos físicos como también con equipos virtuales, además se realizarán las pruebas dentro de la nube generada con OpenStack.

PALABRAS CLAVE: shared bottleneck, MP-TCP, SDN, OpenStack, OpenDayLight, DevOPs

ABSTRACT

Currently the end users of both fixed and mobile devices require high speed connections, to solve this requirement a new protocol called MP-TCP was created, which works efficiently as long as the device has more than one network interface. But if the device has a single network interface, shared bottlenecks occur which limit the use of the network infrastructure and reduce transmission speeds. In this project of titling work it was proposed to solve the problem of shared nets in MP-TCP by using SDN specifically with the OpenDayLight controller. Once the problem of shared bottlenecks in OpenDayLight was solved, a network function was created through DevOps that facilitated the implementation in any environment quickly and reliably. In addition, OpenStack was installed to create a cloud in which the virtualized network function was deployed. The tests were made with physical equipment and also with virtual equipment, in addition tests were performed with OpenStack.

KEYWORDS: shared bottleneck, MP-TCP, SDN, OpenStack, OpenDayLight, DevOps

1. MARCO TEÓRICO

1.1. Introducción

En el presente capítulo se realiza una representación de los fundamentos teóricos necesarios para el desarrollo del Trabajo de Titulación. La estructura de este capítulo se detalla a continuación: se presentará una breve descripción del protocolo MP-TCP (*MultiPath – Transmission Control Protocol*), con énfasis en el problema denominado *shared bottleneck*; aspectos fundamentales sobre Computación en la Nube (*Cloud Computing*); teoría introductoria sobre SDN (*Software Defined Networking*) y NFV (*Network Functions Virtualization*); introducción sobre el controlador OpenDayLight así como también el protocolo OpenFlow y el simulador de red Mininet; finalmente, se realizará una breve introducción sobre OpenStack y DevOps.

1.2. MultiPath – Transmission Control Protocol (MP-TCP)

1.2.1. Introducción

En la actualidad, los dispositivos que se utilizan en la vida diaria tales como: computadoras de escritorio, *laptops* y teléfonos inteligentes, a menudo son *multihomed*[1] y utilizan la arquitectura TCP/IP[2] (*Transmission Control Protocol/Internet Protocol*) [1]. Esta arquitectura se ha utilizado durante mucho tiempo ya que tiene ventajas como la utilización de diversidad de caminos para establecer comunicación extremo a extremo [2]. Algunas desventajas que presenta TCP/IP se describen a continuación:

- Al momento de realizar una conexión TCP[3] solo se crea una conexión, con esta operación se limita tanto el *throughput*[4] *y* solo existe una ruta sin un enlace de *backup*[5] lo que limita la redundancia y la tolerancia a errores, además evita que varios posibles caminos funcionen de forma simultánea y no permite balancear la carga con lo que se desperdicia recursos.
- Al momento de crear una conexión TCP se asocia a cada paquete con 5 elementos de una tuplas: direcciones IP de origen y destino, puertos de origen y destino, e identificador de protocolo de capa de transporte; la tupla permite

[1] *Multihomed*: *Hosts* que posean más de una dirección IP, a cada dirección se asignada a una interfaz de red para conectarse al Internet o a una red determinada.
[2] TCP/IP: Modelo de cuatro capas que presenta un conjunto de reglas generales, para que un equipo se comunique en la red.
[3] *Transmission Control Protocol* (TCP): Protocolo de la capa de transporte, que permite realizar una conexión de manera confiable y entrega los datos de forma ordenada.
[4] *Throughput:* se lo define como la velocidad efectiva de transporte de datos en una red datos.
[5] *Backup*: un enlace que se utiliza cuando la ruta principal tiene algún tipo de falla o se encuentra fuera de servicio.

diferenciar cada conexión, y es utilizada por el *host* receptor para demultiplexar los paquetes de acuerdo al criterio de las 5-tuplas.

En la actualidad, las redes de información tienen una alta demanda de los usuarios para que mejoren la tasa efectiva de envío de datos (*throughput*) así como también la resistencia a fallos en la comunicación; para lograrlo es necesario la utilización de todos los recursos que dispongan los *hosts,* en particular, las diferentes interfaces de red disponibles en cada equipo con el aprovechamiento óptimo de sus múltiples rutas.

Para cumplir con las demandas anteriormente citadas, se desarrolló un nuevo protocolo llamado MP-TCP, el cual permite la utilización de varios caminos o subflujos de una red de datos. En la Figura 1.1 se puede observar un teléfono inteligente denominado “Host MP-TCP”, el cual va a establecer una conexión con el servidor “Server MP-TCP”. El teléfono inteligente dispone de dos interfaces de red, la primera de ellas se conecta mediante TCP a la red móvil LTE[6] y la otra crea una conexión TCP mediante una LAN[7] inalámbrica. El protocolo MP-TCP crea y administra a cada una de estas conexiones TCP a las que se les denomina subflujos, estos caminos son independientes porque utilizan el concepto de las 5-tuplas para diferenciarlos entre sí. Para utilizar este protocolo, tanto el *host* cliente así como el servidor tienen que soportar conexiones MP-TCP. Al utilizar el protocolo mencionado, el *throughput* aumenta ya que los paquetes se envían simultáneamente por diferentes caminos; además, agrega una resistencia a fallos, si se pierde la conectividad con uno de los enlaces, los paquetes se enviarán por la ruta restante. En los *hosts* cliente se maximiza la utilización de recursos mediante el empleo de todas las interfaces de red.

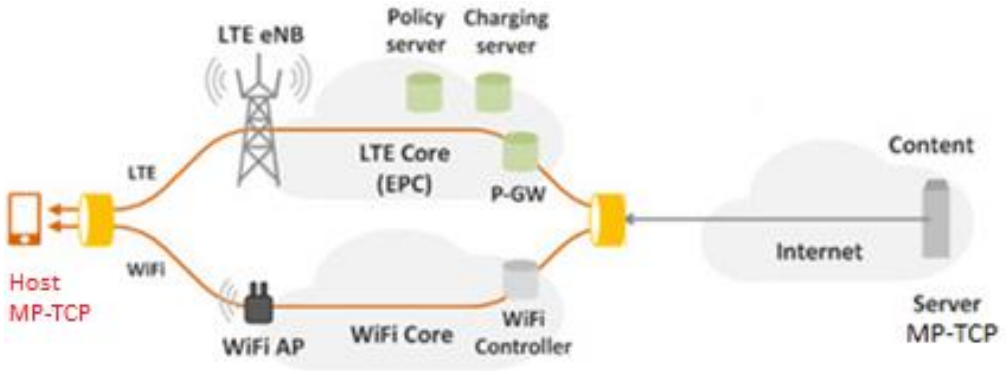

Figura 1.1 Red con dos rutas inalámbricas utilizando el protocolo MP-TCP **[3]**

[6] LTE (*Long Term Evolution*): Es un estándar para comunicaciones inalámbricas de transmisión de datos de alta velocidad para teléfonos y terminales de datos móviles.
[7] LAN (*Local Area Network*): es una red de computadoras que generalmente abarca un área reducida a una casa, un departamento o un edificio.

El escenario antes descrito no es el único que puede ser utilizado para la creación de conexiones MP-TCP; en realidad, el protocolo no requiere de varias interfaces de red para ser empleado, también se puede utilizar una sola interfaz de red para poder enviar varios subflujos. Esto es posible modificando los parámetros de configuración del protocolo MP-TCP tanto en el *host* cliente así como en el servidor. Para crear y enviar los subflujos por una sola interfaz de red no es necesario modificar las 5-tuplas que se envían, ya que tres de estos parámetros se mantienen constantes (direcciones IP de destino y origen y el identificador de protocolo) y solo cambian los puertos de origen y destino. Pero al utilizar solo una interfaz de red se presenta un problema al cual se lo denomina *shared bottleneck* [4], [2], [5] el cual va a ser descrito más adelante.

1.2.2. Generalidades

MP-TCP es un protocolo de la capa de transporte del modelo TCP/IP el cual tiene como finalidad la creación de múltiples subflujos con una sesión de TCP. El IETF[8] puso en marcha el desarrollo de este protocolo en el año 2009, con el objetivo de plantear una solución que permita la división del tráfico en varias posibles rutas con una única sesión TCP. La definición y estructura del protocolo se encuentra en el RFC[9] 6824 [6]; el estado actual de este protocolo es el de experimental, estado adquirido desde enero de 2013.

El protocolo MP-TCP no fue la primera solución que presentó el IETF, existe el protocolo SCTP (*Stream Control Transmission Protocol*) [7] el cual tiene la misma idea que MP-TCP en la creación de múltiples caminos. La diferencia entre estos dos protocolos es que por un lado MP-TCP se encuentra enfocado en mejorar las prestaciones tanto de *throughput* como robustez de TCP por medio del envío simultaneo de subflujos, mientras que el protocolo SCTP trata de implementar redundancia y movilidad a nodos, pero la divergencia esencial es que SCTP no permite la utilización simultánea de subflujos.

Como el protocolo MP-TCP se encuentra en la capa de transporte, el cambio de TCP a MP-TCP es transparente tanto en la capa superior así como en la inferior de la arquitectura TCP/IP; con esto, no se necesitan realizar modificaciones en las demás capas del modelo TCP/IP. A nivel de usuario final, el cambio también es transparente ya que solo se necesita que los equipos tanto del *host* cliente como del servidor dispongan del protocolo MP-TCP y que este se encuentre configurado. Un escenario común donde

[8] IETF(*Internet Engineering Task Force*): es una organización internacional abierta de normalización, que tiene como objetivos el contribuir a la ingeniería de Internet.
[9] RFC (*Request for Comments*): son una serie de publicaciones del grupo de trabajo de ingeniería de internet que describen diversos aspectos del funcionamiento de Internet.

se aplica este protocolo es cuando se utiliza un teléfono inteligente, esto se puede observar en la Figura 1.1.

1.2.3. Arquitectura

La arquitectura del protocolo MP-TCP se encuentra fundada en las propuestas realizadas por Tng (*Transport nextgeneration*) [6], [8]. En la Figura 1.2 se presenta el modelo TCP/IP utilizando para la capa de transporte del protocolo TCP y de MP-TCP; además, se observa que cuando se utiliza MP-TCP, la capa de transporte se divide en dos secciones bien definidas:

- **Subnivel superior:** también conocido como *Semantic Layer* encargado de realizar las funciones orientadas a la aplicación, estas funciones son: inicialización/finalización de las sesiones, establecimiento de los subflujos, detección y uso de múltiples caminos, entre las más importantes.
- **Subnivel inferior:** o también conocido como *Flow+endpoint Layer,* se encuentra enfocado a las tareas orientadas a la red, se encargará de gestionar los subflujos creados durante la fase de inicialización de la conexión. Para ello, se dividirá la operación de control en tantas entidades como subflujos sean creados.

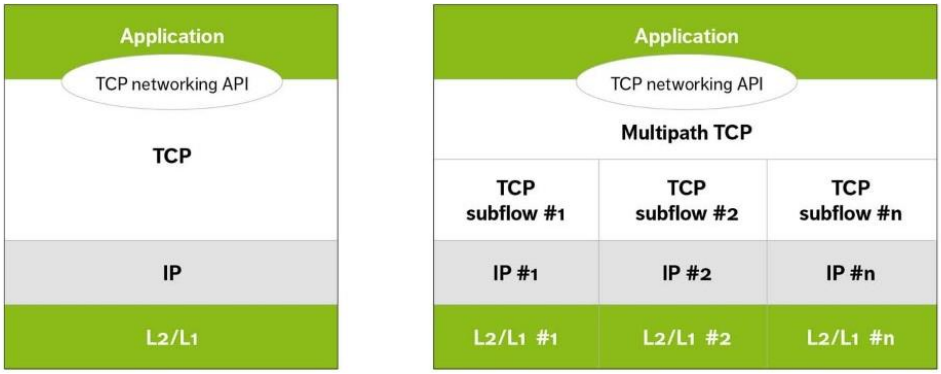

Figura 1.2 Protocolo MP-TCP aplicado al modelo TCP/IP **[3]**

En la Figura 1.2 también se aprecia como el protocolo MP-TCP crea múltiples subflujos, con esto se obtiene una transparencia en la red ya que cada subflujo se comporta como una conexión tradicional TCP.

1.2.4. Subtipos de Mensaje MP-TCP

Al utilizar el protocolo MP-TCP no se modifica la estructura del segmento original del protocolo TCP, solo se altera el contenido del campo de opciones de TCP. En la Figura 1.3 se puede observar la cabecera de TCP y en la Figura 1.4 se observar la cabecera de MP-TCP que mantiene la estructura de TCP, pero en algunos campos tienen otros significados los cuales se encuentran detallados dentro de la Figura 1.4.

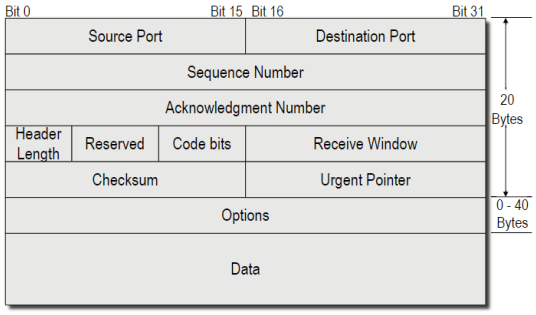

Figura 1.3 Cabecera del segmento TCP **[9]**

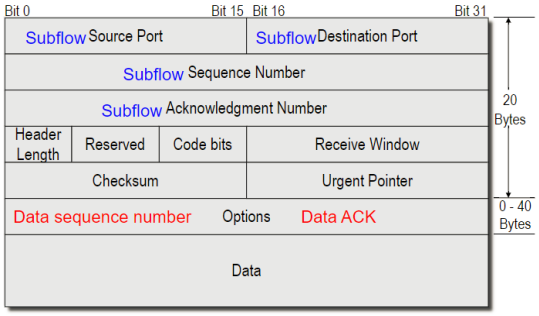

Figura 1.4 Cabecera del segmento MP-TCP **[9]**

El protocolo MP-TCP tiene mensajes específicos para realizar funciones de conexión y envío de datos. En la Tabla 1.1 se presentan los subtipos de mensajes MP-TCP, la primera columna es el valor del identificador del subtipo, en la siguiente columna se especifica el símbolo y finalmente en la columna final se agrega el nombre asignado al mensaje; esta información está incluida en el campo de opciones de TCP.

Tabla 1.1 Subtipos de mensajes MP-TCP

Valor	Símbolo	Nombre
0x0	MP_CAPABLE	*Multipath Capable*
0x1	MP_JOIN	*Join Connection*
0x2	DSS	*Data Sequence Signal*
0x3	ADD_ADDRESS	*Add Address*
0x4	REMOVE_ADDRESS	*Remove Address*
0x5	MP_PRIO	*Change Subflow Priority*
0x6	MP_FAIL	*Fallback*
0x7	MP_FASTCLOSE	*Fast Close*

Para la codificación de los diferentes tipos de mensajes MP-TCP se utilizará el esquema TLV (*type-length-value*)[10]. Los subtipos de mensajes MP-TCP tienen una estructura básica la cual se puede observar en la Figura 1.5. Los campos se describen a continuación:

- El campo “Tipo” tiene una longitud de 8 bits, el valor asignado es 30. Este valor es único para todos los mensajes MP-TCP ya que IANA[11]lo reservó para poder diferenciarlo de otro tipo de mensajes;
- El campo “Longitud” tiene una longitud de 8 bits donde se asigna el tamaño del mensaje MP-TCP en bytes;
- El campo “Subtipo” tiene una longitud de 4 bits, se utiliza para poder diferenciar entre los mensajes MP-TCP;
- Finalmente existe un campo donde se encuentra la información propia de cada mensaje MP-TCP.

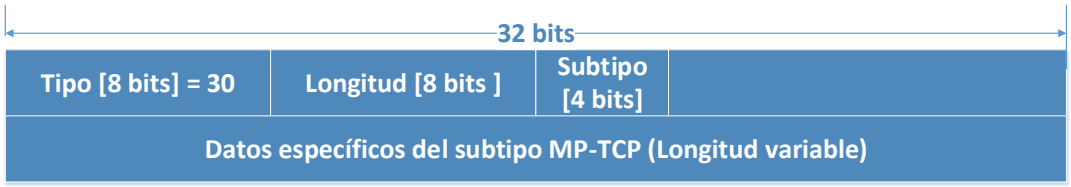

Figura 1.5 Esquema general de los subtipos de mensajes MP-TCP

Para esta Tesis, se describirán y utilizarán únicamente tres tipos de mensajes MP-TCP, los cuales son: MP_CAPABLE, MP_JOIN y MP_DSS; los otros mensajes no se los exponen ya que no se los utiliza [1].

1.2.5. Establecimiento de una conexión MP-TCP

El proceso de establecimiento del subflujo principal se presenta en la Figura 1.6. Como primer paso para establecer una conexión MP-TCP se realiza una conexión TCP de tres vías, el *host* cliente envía un mensaje MP_CAPABLE como se observa en la Figura 1.7, dentro de la cabecera de este mensaje se envían las llaves del *host* cliente al *host* servidor incluido en el segmento SYN [1] [10], con dicho mensaje se notifica al *host* servidor que el cliente puede establecer una conexión MP-TCP. Si el *host* servidor soporta el protocolo MP-TCP responde al *host* cliente con un mensaje MP_CAPABLE como el que se presenta en la Figura 1.7, en este mensaje se agrega la clave del *host*

[10] TLV: se denomina valores de longitud de tipo, el cual es un formato de representación de información, esta codificación se encuentra en la norma 8825-1.
[11] IANA(*Internet Assigned Numbers Authority*): es la entidad que supervisa la asignación global de direcciones IP, sistemas autónomos, servidores raíz de nombres de dominio DNS y otros recursos.

servidor incluido en el segmento SYN-ACK. Finalmente, se envía el tercer mensaje desde el *host* cliente al *host* servidor con la estructura de mensaje que se presenta en la Figura 1.8, donde se envían las llaves del *host* cliente y la del *host* servidor incluido en el segmento ACK [11].

Las llaves que se intercambian en los mensajes MP_CAPABLE son utilizadas posteriormente para agregar subflujos y asociarlos a cada sesión de una conexión MP-TCP. La información de las llaves del *host* cliente y del *host* servidor son enviadas en texto plano, el tamaño de estas llaves es de 64 bits.

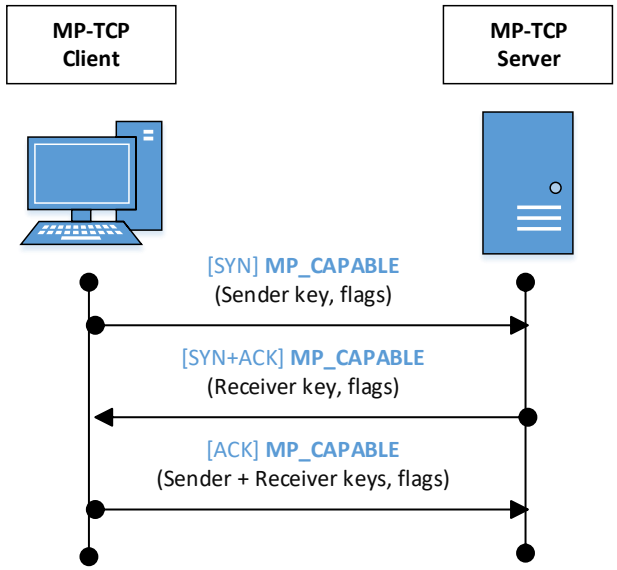

Figura 1.6 Proceso de establecimiento del subflujo principal

1.2.5.1. MP_CAPABLE

Existen tres tipos de mensajes MP_CAPABLE los cuales son utilizados para el establecimiento de la conexión. El primer mensaje ([SYN] MP_CAPABLE) y segundo mensaje ([SYN+ACK] MP_CAPABLE) tienen una longitud de 12 bytes y se presentan en la Figura 1.7. El tercer mensaje ([ACK] MP_CAPABLE) se observa en la Figura 1.8 y tiene una longitud de 20 bytes.

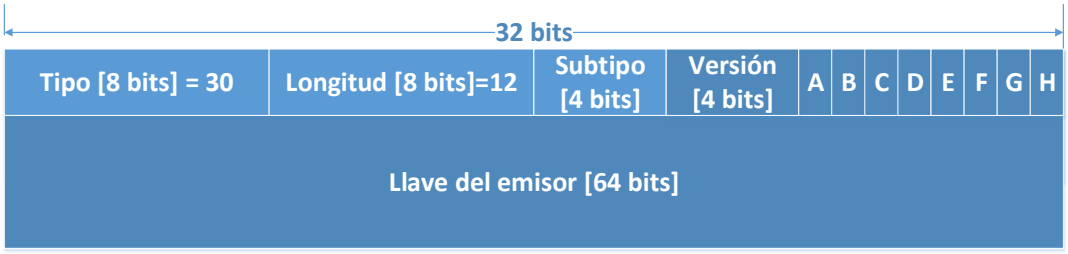

Figura 1.7 Primer y segundo mensaje MP_CAPABLE

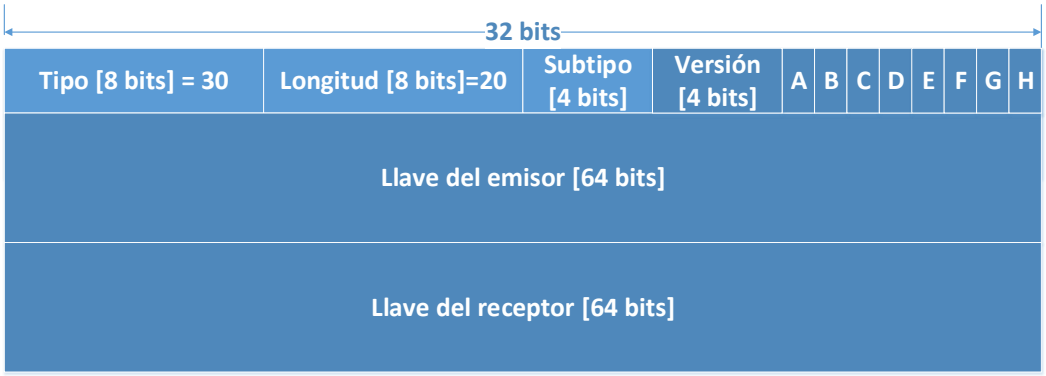

Figura 1.8 Tercer mensaje MP_CAPABLE

Los mensajes MP_CAPABLE tienen varios campos, a continuación solo se detallan los campos que son relevantes para esta Tesis:

- **Subtipo** [4 bits]: el valor de este campo se encuentra establecido en 0 para los mensajes MP_CAPABLE.
- **Versión** [4 bits]: el valor de este campo se encuentra establecido en 0, este campo determina la versión de la implementación del protocolo MP-TCP.
- Las banderas A hasta H se encuentran descritas en [1].
- **Llave del emisor** [64 bits]: este campo es generado por el *host* emisor, se transmite en texto plano, tanto en el primer y en el segundo mensaje MP_CAPABLE. Esta llave se utiliza para asociar subflujos a una conexión MP-TCP.
- **Llave del receptor** [64 bits]: este campo es generado por el *host* receptor, se transmite en texto plano en el segundo y tercer mensaje MP-CAPABLE. Como en el campo anterior se utiliza la información para asociar subflujos a una conexión MP-TCP.

1.2.6. Adición de subflujos

Una vez que se crea el primer subflujo mediante los mensajes MP_CAPABLE, es necesario crear nuevos subflujos a la conexión MP-TCP, para esta acción se utilizan los mensajes MP_JOIN [1]. Los pasos que se necesitan implementar para la creación de un nuevo subflujo se presentan en la Figura 1.9.

Para añadir un nuevo subflujo, en primer lugar, el *host* cliente envía un mensaje MP_JOIN con la estructura que se puede observar en la Figura 1.10. En este primer mensaje se incluye en el segmento SYN, además contiene información como el token del servidor, un número aleatorio generado por el *host* cliente y un identificador unívoco que representa a la dirección IP [11], [10].

El token que se envía en el primer mensaje MP_JOIN, se encuentra conformado por los 32 bits más significativos del *hash* generado por el algoritmo SHA-1 empleando la llave del *host* servidor, esta llave se la obtiene cuando se establece el primer subflujo de la conexión MP_TCP. El número aleatorio generado por el *host* cliente al cual se lo conoce como Nonce, de 32 bits de longitud, será utilizado por el *host* servidor cuando el responda al *host* cliente. El identificador de dirección se encuentra relacionado con la IP de origen del paquete, es un identificador único generado por el *host* transmisor.

Si el primer mensaje MP_JOIN llega al *host* servidor, lo procesa y verifica que el token que se envía desde el *host* cliente es válido. El servidor responde con un mensaje MP_JOIN con la estructura que se presenta en la Figura 1.11 se incluye en el segmento SYN+ACK. En este mensaje envía información como: el número truncado del código de autenticación generado mediante la función HMAC-A (*hash* SHA-1); el número aleatorio (Nonce) generado por el *host* servidor; y, también se agrega un número aleatorio denominado Rand-B que es generado por el *host* servidor.

Después que el *host* servidor envía la respuesta al *host* cliente mediante un mensaje MP_JOIN, el *host* cliente responde por medio de otro mensaje MP_JOIN con la estructura que se presenta en la Figura 1.12, además incluye el segmento ACK y se añade el HMAC-B completo, el cual tiene una longitud de 160 bits. Una vez enviado el tercer mensaje MP_JOIN desde el *host* cliente, el nuevo subflujo se encuentra en un estado **Preestablecido (*Preset*)** a la espera que el *host* servidor envíe un cuarto mensaje TCP incluyendo el segmento ACK. Al momento que el *host* cliente recibe el cuarto mensaje, el subflujo cambia de estado de **Preestablecido** ha **Establecido (*Established*)** habilitando el envío de datos por este nuevo subflujo.

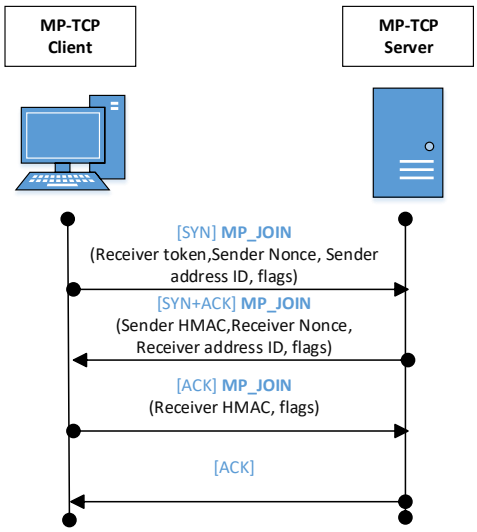

Figura 1.9 Establecimiento de subflujos adicionales

1.2.6.1. MP_JOIN

Durante el establecimiento del subflujo principal, el protocolo MP-TCP puede añadir nuevos subflujos. Los *hosts* que intervienen en la conexión MP-TCP conocen las direcciones IP que se encuentran asignadas a cada interfaz, para poder notificar de esta información a los *hosts* que intervienen en la comunicación se utilizan los mensajes MP_JOIN.

Existen tres tipos de mensajes MP_JOIN, el primer mensaje ([SYN] MP_JOIN) se observa en la Figura 1.10, el tamaño del mensaje es de 12 bytes y contiene los siguientes campos:

- **Subtipo** [4 bits]: el valor asignado para este campo se encuentra en 1.
- **R** [3 bits]: este campo es de tipo experimental, es reservado para futuras implementaciones, al momento se encuentra asignado con el valor de 0.
- **B** [1 bit]: esta bandera se utiliza para indicar que el *host* emisor del mensaje MP-TCP desea crear un nuevo subflujo, si el valor se encuentra en 1 se usa como camino de *backup*; 0 en un evento donde fallen los subflujos.
- **Identificador de dirección** [8 bits]: este campo tiene un valor como un identificador para cada conexión MP-TCP, es un valor entero el cual identifica de forma única la dirección IP del *host* emisor en una sola conexión MP-TCP.
- **Token del receptor** [32 bits]: a este campo se le asigna el valor de los 32 bits más significativos del hash SHA-1[12] que se generan a partir de la llave del *host* receptor.
- **Número aleatorio del emisor** [32 bits]: a este campo se le asigna un valor aleatorio generado por el *host* emisor.

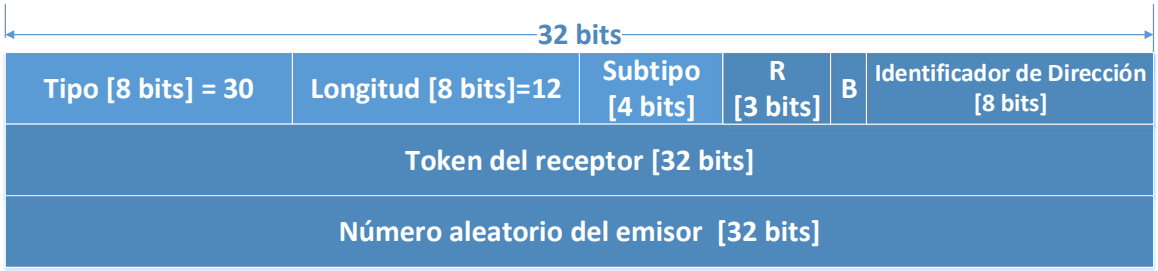

Figura 1.10 Primer mensaje MP_JOIN

[12] SHA-1 (*Secure Hash Algorithm* 1): es una función criptográfica hash diseñada por la Agencia de Seguridad Nacional de los Estados Unidos y es un Estándar Federal de Procesamiento de Información publicado por el NIST de los Estados Unidos.

La estructura del segundo mensaje MP_JOIN ([SYN+ACK] MP+TCP) se presenta en la Figura 1.11. La longitud de este mensaje es de 16 bytes, los campos que cambian con respecto al primer mensaje MP_JOIN se exponen a continuación:

- **HMAC[13] truncada del emisor** [64 bits]: a este campo se le asignan los 64 bits más significativos de HMAC-SHA1el cual tiene un tamaño de 160 bits. El valor de HMAC-SHA1 se obtiene al generar operaciones sobre las llaves del emisor y del receptor, las cuales se obtienen al iniciar la conexión MP-TCP.

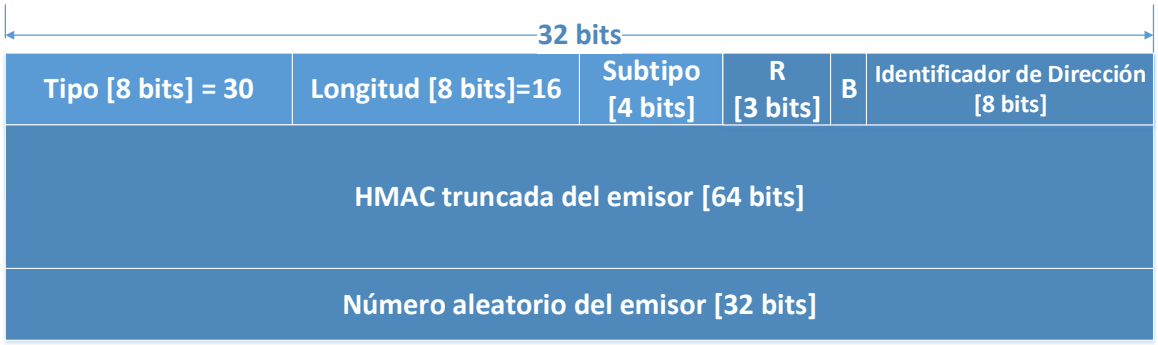

Figura 1.11 Segundo mensaje MP_JOIN

El tercer y último mensaje ([ACK] MP_JOIN) se presenta en la Figura 1.12. Este mensaje tiene una longitud 24 bytes. En este mensaje se agregan nuevos campos, los cuales se explican a continuación:

- **Reservado** [12 bits]: el valor para este campo es reservado, se utiliza para futuras implementaciones (experimental), solo se presenta en el tercer mensaje MP_JOIN.
- **HMAC del emisor** [160 bits]: En este campo se asigna el valor completo de HMAC del emisor el cual tiene una longitud de 160 bits. El protocolo MP-TCP genera HMAC-SHA1 en base a dos campos los cuales se obtienen tanto del emisor como del receptor, estos son: las llaves y los números aleatorios que se generan a partir de cada mensaje, a continuación se presenta la forma como se genera el HMAC tanto del emisor (A) así como del receptor (B):
- HMAC-A = HMAC(Llaves=(Llave-A+Llave-B), Mensaje=(Rand-A+Rand-B))
  HMAC-B = HMAC(Llaves=(Llave-B+Llave-A), Mensaje=(Rand-B+Rand-A))
- De la expresión anterior se puede observar que las Llaves se generan a partir de la concatenación (+) de las llaves dependiendo si es del emisor o del receptor. Los

[13] HMAC (*keyed-hash message authentication code*): HMAC es una construcción que se utiliza para calcular un código de autentificación de mensaje (MAC) que implica una función hash criptográfica en unión con una llave secreta.

Mensajes se generan a partir de la concatenación de los números aleatorios según el HMAC.

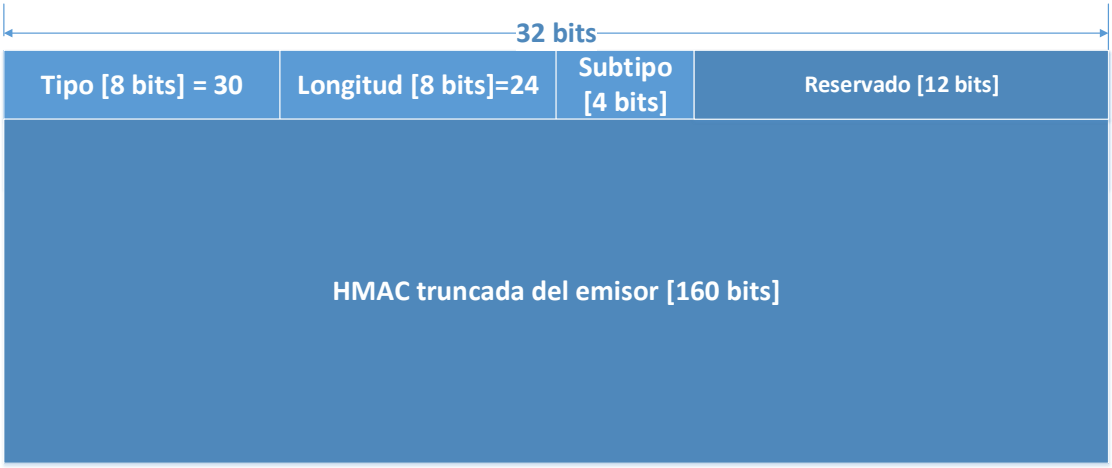

Figura 1.12 Tercer mensaje MP_JOIN

1.2.7. Transferencia de información en MP-TCP

Para la transferencia de información mediante el protocolo MP-TCP, el *host* cliente, o también llamado como *host* emisor, toma la información de la capa superior (capa aplicación) y la envía por los subflujos administrados y creados por MP-TCP. La información que agrega el *host* cliente a cada subflujo será administrada por el protocolo MP-TCP para que el *host* servidor pueda re-ensamblar la información y pasarla a la capa superior de forma confiable como es el principio de una conexión TCP. Para obtener lo antes mencionado, se utiliza el *Data Sequence Mapping* (DSM), que es información de control que se encuentra en los mensajes MP_DSS [1], [10].

Existen dos espacios de secuencias en MP-TCP que son:

- ***Subflow Sequence Number*** (SSN): este campo asigna un valor de secuencia para cada subflujo y solo tiene relevancia en el subflujo aplicado.
- ***Data Sequence Number*** (DSN): a este campo se asigna el valor de secuencia de datos de forma global, el DSN permite la numeración de todos los paquetes que se encuentra en una conexión MP-TCP.

La estructura del mensaje MP_DSS se presenta en la Figura 1.13, donde se puede apreciar que un mensaje *Data Sequence Mapping* (DSM) se encuentra conformado por los siguientes campos: *Data Sequence Number* (DSN), el *Subflow Sequence Number* (SSN), el *Data-level Length* y el checksum. Los elementos antes expuestos se utilizan para la reconstrucción de los mensajes a partir de los subflujos creados por el protocolo MP-TCP [11].

1.2.7.1. MP_DSS

Los mensajes MP_DSS (*Data Sequence Signal*) son los encargados de transportar la información generada por el usuario, la estructura del mensaje se presenta en la Figura 1.13. El mensaje MP_DSS tiene campos específicos que se presentan a continuación:

- **Subtipo** [4 bits]: el valor para este campo se encuentra establecido en 2.
- **Reservado** [7 bits]: campo experimental reservado para nuevas implementaciones.
- Se puede obtener información de las banderas F, m, M, a, y A en [1].
- **Confirmación de recibo de datos (ACK)** [32 o 64 bits]: a este campo se le asigna un número que representa la cantidad de datos que se recibieron correctamente en el *host* emisor de los mensajes MP_DSS.
- ***Data Sequence Number* (DSN)** [32 o 64 bits]**:** a este campo se le asigna el número de todos los datos a nivel de la conexión MP-TCP.
- ***Subflow Sequence Number* (SSN)** [32 o 64 bits]**:** a este campo se le asigna el valor de la numeración de los datos sobre cada subflujo de una sesión MP-TCP.
- ***Data-level Length*** [16 bits]: a este campo se le asigna el valor de la longitud del tamaño del *payload* en bytes, el cual es numerado con un DSN.
- **Checksum** [16 bits]: este campo es utilizado para detectar errores producidos en la cabecera y en los datos del mensaje MP_DSS.

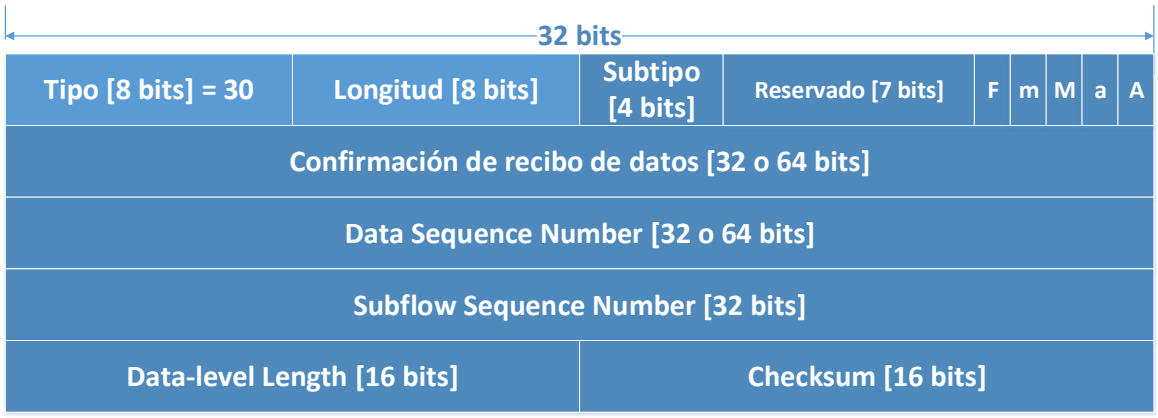

Figura 1.13 Mensaje MP_DSS

1.2.8. Cierre de una conexión MP-TCP

Para la finalización de una conexión MP-TCP de forma usual [1] [11], existen dos posibilidades que se describen a continuación:

- En la primera opción el *host* cliente envía toda la información requerida al *host* servidor. El *host* cliente envía un mensaje de tipo DATA_FIN para el cierre de una conexión; este elemento es una bandera dentro del mensaje MP_DSS.
- La segunda opción para cerrar una conexión MP-TCP es que el *host* cliente envíe un mensaje MP_FASTCLOSE al *host* servidor; este tipo de mensaje es utilizado cuando la conexión es abruptamente cerrada y, por tanto, no se aceptaran datos adicionales de la conexión MP-TCP.

1.3. Problema del Shared Bottleneck del protocolo MP-TCP

El problema denominado *shared bottleneck* del protocolo MP-TCP se presenta cuando no existen caminos disjuntos en una conexión MP-TCP [2], [4], esto quiere decir que los caminos o rutas que utilizan los paquetes de datos no comparten ningún nodo o elemento de red en común. Este escenario se presenta cuando tanto el *host* cliente así como el *host* servidor disponen de una única interfaz de red, por tanto, los flujos creados poseen las mismas direcciones IP de origen y de destino [1].

En la Figura 1.14 se observa un esquema de red en el cual se presenta el problema *shared bottleneck* ya que tanto el *host* cliente como el *host* servidor solo cuentan con una interfaz de red, además entre el nodo SW1 y SW2 existen tres posibles caminos, pero la información solo tiende a transferirse por un camino ya que los puertos de los nodos SW3 y SW4 se encuentran bloqueados [5].

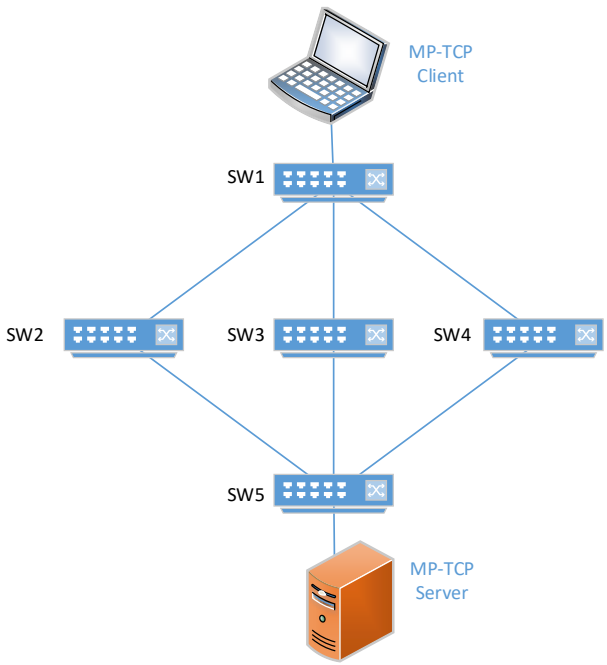

Figura 1.14 Problema del *shared bottleneck* en una red con protocolo MP-TCP

En la Figura 1.14, el protocolo MP-TCP, mediante los mensajes MP_CAPABLE, crea el subflujo principal, el cual pasará por los nodos SW1, SW2 y SW5. Ya establecido el primer subflujo, el *host* cliente envía un mensaje MP_JOIN para la creación de un nuevo subflujo.

Al llegar al nodo SW1 solo tiene una ruta para enviar la información, ya que se asume que los puertos que se conectan a los nodos SW3 y SW4 se encuentran bloqueados para evitar bucles.

Por esta razón los mensajes MP_JOIN tomarán el mismo camino que el subflujo principal. Los bucles que se originan en la topología de red se evitan con la utilización del protocolo STP (*Spanning Tree Protocol*)[14].

Por este motivo al tener un solo camino determinado por STP se produce el problema *shared bottleneck* de MP-TCP, el *throughput* no se incrementa si se compara con una conexión TCP.

Tampoco se incrementa la resistencia a fallos; estos inconvenientes pueden ser superados al utilizar múltiples subflujos del protocolo MP-TCP, para conseguir esto, es necesario resolver el problema de *shared bottleneck* de MP-TCP.

1.4. Software Defined Networking (SDN)

Las SDN se encuentran definidas por la ONF[15] (*Open Networking Foundation*), en esta fundación se especifica que las SDN son redes en las cuales se encuentra separado el plano de control del plano de datos.

El plano de datos es manejado por los dispositivos de red como switches y el plano de control es transferido a un servidor, al cual se lo denomina controlador que es el encargado de implementar la lógica de la red en caso que toda la infraestructura sea controlada por el controlador [12], [13].

En la Figura 1.15 se presenta la arquitectura de una Red Definida por Software donde se observan los planos de control y de datos que se encuentran separados, para interconectar estos planos es necesario la utilización de un protocolo, uno de los más utilizados es OpenFlow [14].

[14] STP (*Spanning Tree Protocol*) es un protocolo de red de nivel 2 del modelo OSI, su función es la de gestionar la presencia de bucles en topologías de red debido a la existencia de enlaces redundantes.
[15] ONF aprovecha los principios y la desagregación de SDN, utilizando plataformas de código abierto y estándares definidos para construir redes de operadores.

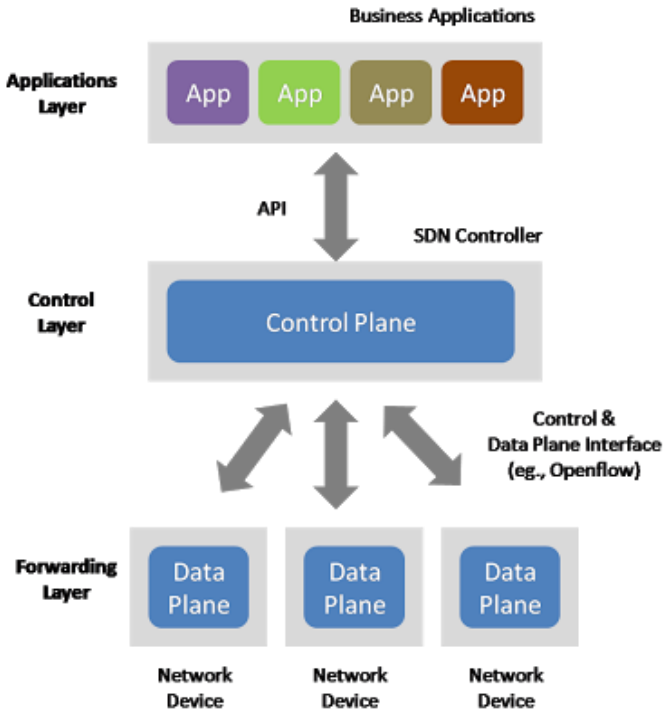

Figura 1.15 Arquitectura de una Red Definida por Software **[15]**

Lo que se pretende al utilizar SDN es simplificar las operaciones de red ya que en la actualidad cada vez que se requieren implementar nuevos servicios en la infraestructura como por ejemplo un *firewall,* balanceadores de carga, NAT, DHCP entre otros, es necesario en algunos casos esperar que los fabricantes realicen cambios en los dispositivos de red, incrementando el costo de operación y el tiempo de ejecución. Al utilizar las SDN también se optimizan los recursos ya que los dispositivos de red solo se encargan de la conmutación con una mínima inteligencia y las decisiones e inteligencia de la infraestructura se manejan en el controlador SDN, el cual cuando se realice un cambio o se modifique alguna acción, ejecutará los cambios de forma automática en cada dispositivo de red, para obtener los resultados deseados. Además permita la configuración de los switches con lenguajes de alto nivel lo que facilita la operación de los mismos también se pueden desarrollar diferentes aplicaciones que pueden subsistir al mismo tiempo a nivel de la capa de aplicación en un solo controlador.

1.5. OpenFlow

El protocolo OpenFlow fue creado por la Universidad de Stanford en el año 2007. El objetivo que tiene este protocolo es la comunicación entre los planos de control con el de datos en una arquitectura SDN a través de un canal seguro. El protocolo OpenFlow se encuentra en desarrollo para nuevas versiones por ONF y además es un protocolo abierto [1], [16]. En la Figura 1.16 se puede observar las versiones del protocolo

OpenFlow con el año de lanzamiento como también con los cambios que fueron implementados en cada versión nueva.

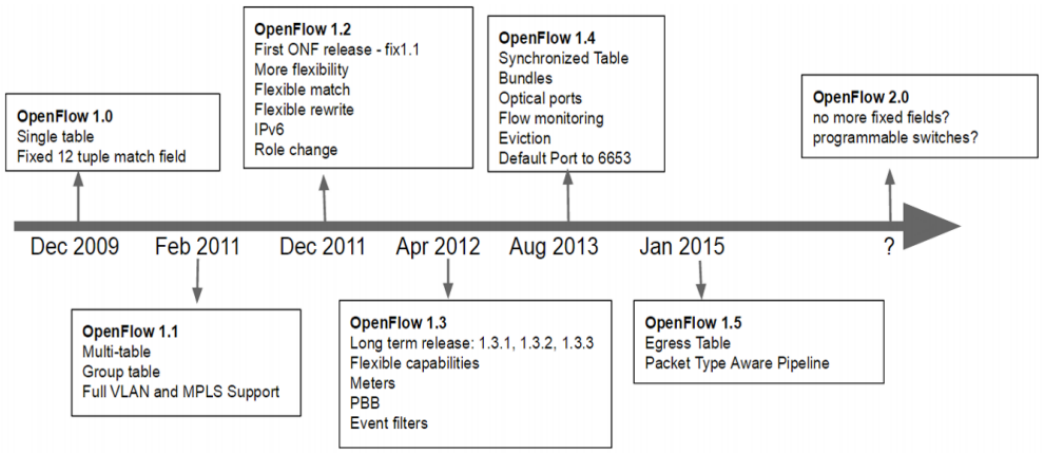

Figura 1.16 Cronología y mejoras del protocolo OpenFlow **[17]**

El protocolo OpenFlow es el medio o interfaz por el cual se pueden conectar a los equipos de red que se encuentran enlazados a un controlador SDN, permitiendo instalar reglas mediante las cuales se establecen acciones predeterminadas que se ejecutarán si los flujos de datos que llegan a un switch cumplen ciertas condiciones (comparaciones). Las reglas se las pueden generar de forma manual mediante un administrador de red o se pueden generar de forma dinámica de acuerdo a condiciones que se presenten en la red mediante la programación del controlador SDN. En la Figura 1.17 se presenta la estructura SDN utilizando OpenFlow.

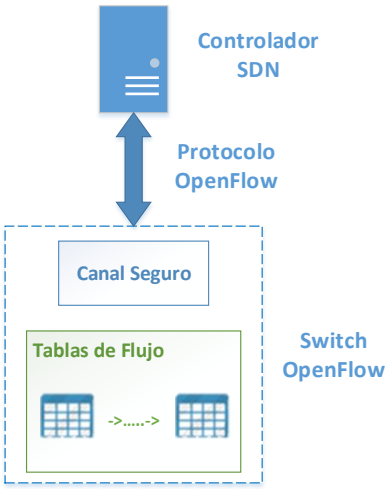

Figura 1.17 Estructura de componentes dentro de un switch OpenFlow

1.5.1. Switch OpenFlow

Los switches OpenFlow son equipos de red que soportan el protocolo OpenFlow, el cual añade características adicionales a las que tiene un switch normal, como son las tablas

de flujo donde se determinan las acciones a realizar cuando un paquete ingresa al switch. Los switches OpenFlow pueden ser físicos así como virtuales. Para manejar los switches virtuales se podría utilizar el software Mininet que se presentará más adelante. A continuación se detallan los módulos que conforman un switch OpenFlow, de acuerdo a lo indicado en la Figura 1.17:

1.5.1.1. Canal Seguro

Canal seguro es una interfaz utilizada por el protocolo OpenFlow para realizar una conexión entre el controlador SDN y los switches OpenFlow para poder instalar reglas o para administrar a los dispositivos de red. Todos los mensajes que ingresen al canal seguro deben tener un formato compatible con el protocolo OpenFlow. Además dentro del canal seguro se pueden implementar diferentes mecanismos de autentificación y control de acceso en la comunicación como por ejemplo con: *Transport Layer Security*[16], *Secure Shell* (SSH)[17] y IPSec[18] [18].

1.5.1.2. Tabla de Flujos

Las tablas de flujos contienen las reglas instaladas por el controlador SDN que proporciona acciones determinadas para los paquetes que ingresan a los switches OpenFlow. En la Figura 1.18 se presenta la estructura de la tabla de flujos.

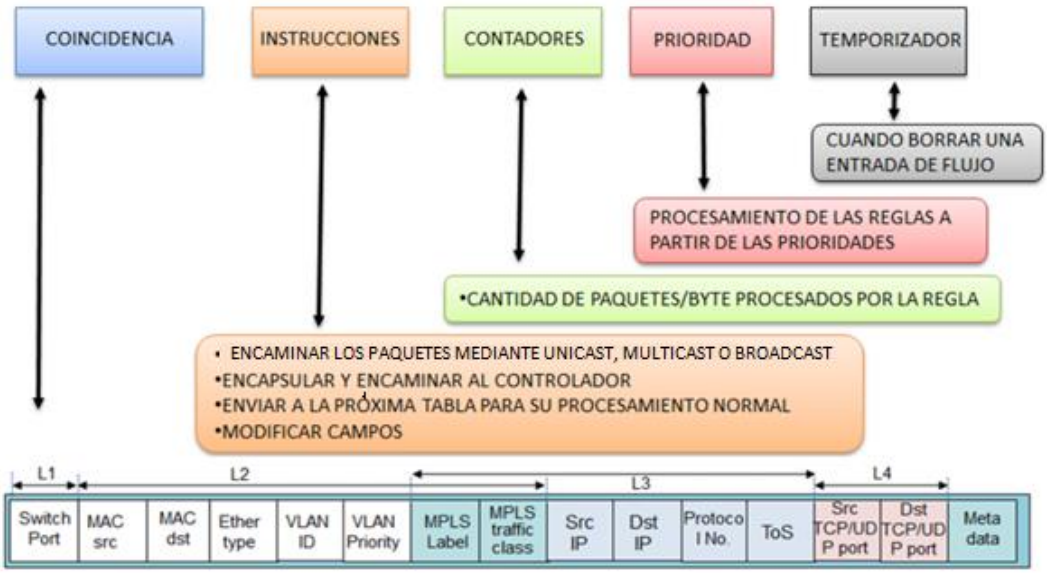

Figura 1.18 Estructura de la tabla de flujos **[16]**

[16] TLS (*Transport Layer Security* o Seguridad de la capa de transporte): este protocolo permite la comunicación con integridad y privacidad de la información a transmitir.
[17] SSH (*Secure Shell*): Es un protocolo de red el cual proporciona un canal seguro dentro de una infraestructura no segura.
[18] IPSec: Es un conjunto de protocolos de red que autentica y encripta los paquetes de datos enviados a través de una red.

Una tabla de flujos contiene varios argumentos que a continuación se los describen:

- **Campos de emparejamiento o coincidencia** [*Matching fields*]: este campo es utilizado para realizar coincidencias con los paquetes que ingresan a los switches OpenFlow. Los elementos comunes de este argumento se presentan en la Figura 1.18 en la sección coincidencia que abarca elementos desde la capa física hasta la capa de transporte del modelo OSI, además existe un campo adicional denominado como metadata el cual es utilizado para la transferencia de información entre tablas de flujos.
- **Instrucciones** [*Instructions*]: contiene una serie de acciones a ejecutarse cuando se produce una correspondencia en los campos de emparejamiento.
- **Contadores** [*Counters*]: este valor se incrementa cada vez que ingresa un paquete y se realiza el proceso de emparejamiento exitoso.
- **Prioridad** [*Priority*]: este parámetro es utilizado para definir prioridades de una regla a otra dentro de una tabla de flujo.
- **Temporizador** [*Timeouts*]: permite la eliminación de una regla dentro de una tabla de flujo dentro de un tiempo determinado.
- ***Cookie***: a este parámetro se le asigna un valor para filtrar las estadísticas, modificaciones y la eliminación de subflujos. Este campo no es utilizado cuando se procesan los paquetes.

El funcionamiento de la tabla de flujos es el siguiente: en primer lugar un paquete llega al switch OpenFlow, el cual busca en el campo de emparejamiento los datos que coincidan con los valores obtenidos en los paquetes, si existe un subconjunto de valores similares los cuales cumplen con condiciones ya establecidas en el controlador se dirige al campo de Prioridad para seleccionar la instrucción más adecuada y ejecutar las acciones indicadas.

1.6. Mininet

En este Proyecto de Titulación se utiliza redes virtuales, para la creación de estos entornos se maneja el simulador Mininet [19], con esta herramienta se facilita la creación y administración de redes.

Este simulador puede crear tanto switches como *hosts*, estos últimos utilizan un esquema de virtualización en el cual se crean copias limitadas del kernel anfitrión. La instalación

común de este simulador es en máquinas con Linux donde se puede crear una red mediante la programación en el lenguaje Python[19] o por CLI[20] (*Command-Line Interface*).

Una ventaja que tiene Mininet es la utilización de switches virtuales como por ejemplo Open vSwitch (OVS) que tienen soporte para el protocolo OpenFlow.

Además si el sistema operativo tiene disponibilidad y se encuentra activado el protocolo MP-TCP, los *host* creados en Mininet también estarán disponibles.

1.7. Controlador OpenDayLight

El controlador OpenDayLight nace en el año 2013 como un proyecto colaborativo en el que intervinieron varias empresas importantes del sector de telecomunicaciones entre ellas Cisco, HP, IBM, Intel, Juniper, Microsoft, Red Hat y VMware, entre las más importantes [1], [20].

El objetivo principal que tenía el proyecto era el desarrollo y la aceptación de las SDN, de este modo es creado el controlador OpenDayLight el cual se encuentra basado en código abierto para que cualquier persona o entidad pueda mejorarlo o utilizarlo [21].

El controlador OpenDayLight se ha constituido en una plataforma compleja lo cual permite trabajar en varios entornos de infraestructura de red.

OpenDayLight contiene una infraestructura modular la cual incluye protocolos y estándares emergentes así como también nuevos servicios de red y aplicaciones dedicadas [22], [23].

La versión del controlador OpenDayLight que se utilizó para la presente Tesis se utilizó Beryllium, el controlador fue desarrollado el 22 de febrero del 2016, se utilizó esta versión del controlador ya que es una versión estable.

1.7.1. Arquitectura de OpenDayLight

La arquitectura del controlador OpenDayLight se presenta en la Figura 1.19, donde se puede observar su división en módulos que a su vez se dividen en capas más pequeñas.

A continuación se presentan cada uno de estos módulos importantes con una descripción de lo que realizan.

[19] Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible

[20] CLI (*Command-Line Interface*): es un método que permite a los usuarios escribir instrucciones a algún programa informático por medio de una línea de texto simple.

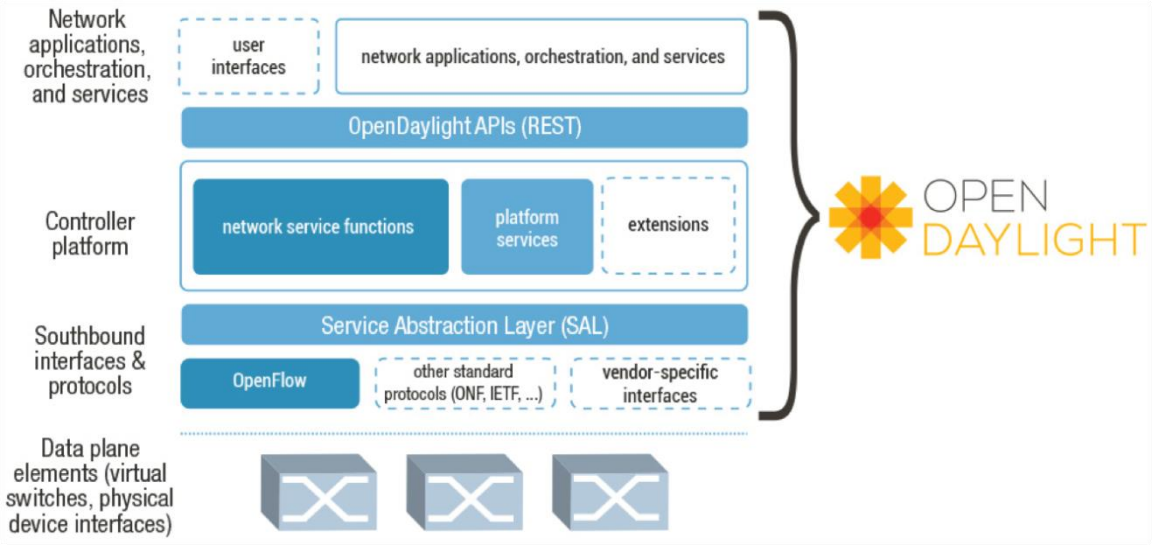

Figura 1.19 Arquitectura del controlador OpenDayLight **[24]**

1.7.1.1. Servicios de gestión y aplicaciones

El módulo superior del controlador OpenDayLight se encuentra conformado por aplicaciones que pueden controlar y administrar a la red por medio del plano de control de OpenDayLight.

1.7.1.2. Interfaz Northbound (NB)

Esta sección cuenta con una cantidad amplia de aplicaciones desarrolladas mediante una API REST[21] que puede ser utilizada por la capa superior para controlar y administrar la red, o se pueden desarrollar nuevas aplicaciones que utilicen esta API para modificar la lógica de la red con cualquier lenguaje de programación que pueda establecer una conexión con los servicios northbound.

1.7.1.3. Plataforma de Servicios

En este módulo el controlador brinda un conjunto de servicios de red, estos elementos se encuentran contenidos en *bundles*[22] , los cuales realizan una función específica según fueron diseñados. A continuación se presentan los servicios más importantes y utilizados:

L2Switch: Este *bundle* brinda el servicio de conmutación a nivel de capa 2 del modelo OSI a través de los dispositivos SDN conectados al controlador. Este *bundle* no trabaja solo, sino con un conjunto de *bundles* que se describen a continuación [25]:

[21] API REST: son aplicaciones o servicios los cuales permiten obtener o escribir datos mediante el protocolo HTTP.
[22] *Bundle:* Son archivos JAR (Java ARchive) que ejecutan aplicaciones escritas en lenguaje Java. Entiempo de ejecución pueden cargarse o descargarse de memoria.

- **PacketHandler:** Este *bundle* es el encargado de decodificar los paquetes que llegan del switch del controlador; también envía los paquetes al *bundle* correcto para que los procese. El paquete puede ser decodificado de tres formas: Ethernet, ARP[23] e IP, pueden existir combinaciones entre estas tres formas.
- **ARP Handler:** Este *bundle* realiza el procesamiento y manejo de los paquetes ARP. Este *bundle* trabaja conjuntamente con PacketHandler y L2Switch, del primero se obtienen los paquetes que son transmitidos por el switch y del segundo *bundle* envía información que será utilizada para conseguir ciertas funciones de ARP Handler.
- **Host Tracker:** Este *bundle* es el encargado de localizar a todos los *hosts* dentro de la red, la información que almacena es: dirección MAC, dirección IP, switch y el puerto al que está conectado el host.
- **LoopRemover:** Este *bundle* se encarga de descubrir y eliminar los lazos creados en la red mediante el protocolo STP.
- **L2SwitchMain:** Este *bundle* es el encargado de instalar las reglas en cada switch, generadas con toda la información recolectada a través de los *bundles.*

Topology Manager: Este *bundle* maneja información sobre todos los dispositivos de red que se encuentran conectados al controlador, como es el caso de los elementos conectados al switch o la velocidad que trabajan cada puerto. Esta *bundle* se encuentra conformada por los siguientes servicios: *Topology Discovery*, *Topology Update*, *Topology Database*, *Topology Northbound* APIs.

Statistics Manager: Este *bundle* maneja y suministra un conjunto de estadísticas, las cuales son obtenidas a través de peticiones a los switches que conforman la red SDN.

Switch Manager: Este elemento mantiene un registro de los nodos así como también de los detalles sobre su conexión. Cada vez que un nodo se conecta al controlador, este *bundle* almacena la información para utilizarla en un futuro.

Forwarding Rules Manager (FRM): Este *bundle* es el encargado de gestionar la programación de los flujos en la infraestructura.

Además, almacena en un repositorio la información de las reglas instaladas en los switches OpenFlow. Se lo puede considerar como un pre-compilador para la instalación de flujos

[23] ARP (*Address Resolution Protocol*) es un protocolo de la capa de enlace, responsable de encontrar la dirección MAC que corresponde a una dirección IP.

Además del módulo de servicio de red existen otros módulos los cuales brindan servicios básicos de red y proporcionan características específicas a la infraestructura de SDN.

1.7.1.4. Sección de Abstracción de Servicios

La creación de SAL (*Service Abstraction Layer*) es la estrategia para simplificar el desarrollo de la interfaz programable de OpenDayLight.

Este módulo es uno de los conceptos centrales de la arquitectura de OpenDayLight ya que es el encargado de establecer conexiones entre los módulos de red y la interfaz *southbound* (SB).

Esta capa tiene el objetivo de determinar las instrucciones para la instalación de las reglas por parte de un servicio, módulo o aplicación sin tomar en cuenta el tipo protocolos se encuentren utilizando en el módulo *southbound*, el controlador y los switches OpenFlow.

SAL tiene dos tipos de arquitecturas: *Application Driven-Service Abstraction Layer* (AD-SAL) y *Model-Driven Service Abstraction Layer* (MD-SAL).

AD-SAL en las nuevas versiones del controlador OpenDayLight ha sido remplazada por MD-SAL. En la Figura 1.20 se presentan la arquitectura de MD-SAL.

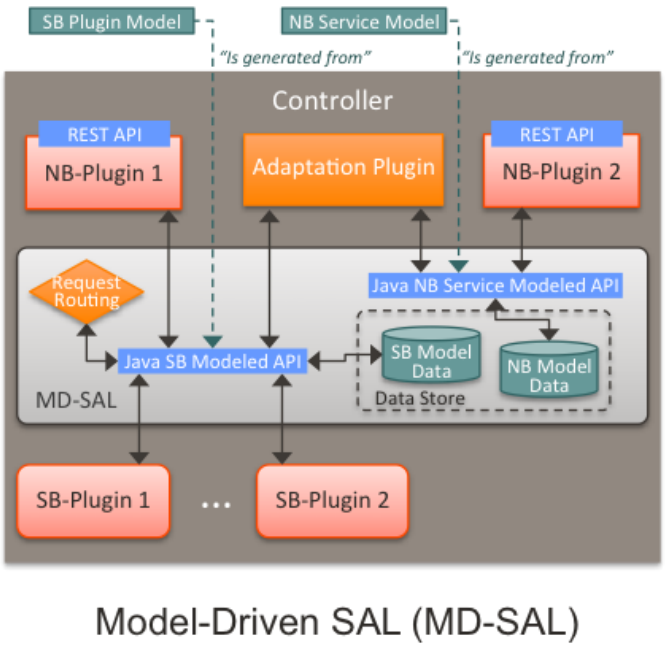

Figura 1.20 Arquitectura MD-SAL **[26]**

MD-SAL suministra una solicitud de *routing* entre los consumidores y proveedores de servicios mediante modelos que se los define utilizando el lenguaje YANG y la infraestructura para realizar la adaptación del servicio que se va a utilizar. El servicio de

adaptación es empleado por los *plugins* tanto SB (*southbound*) como NB (*northbound*), los cuales suministran información a SAL y adquiere datos mediante las APIs.

La arquitectura MD-SAL solo admite una API para *plugins* tanto *northbound* y *southbound*, por este motivo el primero de ellos se convierte en proveedor de API y el otro se convierte en una API consumidora, con esto se elimina la necesidad de crear nuevas API [20]. La arquitectura MD-SAL posee dos componentes principales [21]: *Data Store* y *Message Bus*.

Data Store: Es un repositorio compartido por todos los módulos y servicios del controlador OpenDayLight. El *Data Store* se encuentra dividido en dos partes que se describen a continuación:

- **Configurational Data Store:** Este repositorio cuenta con una representación del estado esperado de la infraestructura. Como una alternativa se puede utilizar una API REST con la que se puede modificar esta estructura e implementar estos cambios mediante *Operational Data Store* en cada elemento de la SDN.
- **Operational Data Store:** Este repositorio cuenta con una representación del estado real en la que se encuentra la infraestructura en funcionamiento, es decir que guarda configuraciones reales de los equipos conectados al controlador.

Message Bus: Este componente permite que varios módulos o plugins se comuniquen entre sí en un mismo controlador.

Interfaz Southbound (SB): En esta capa se encuentran localizados muchos plugins que utilizan ciertos protocolos para poder controlar la infraestructura a la que el controlador se encuentra conectado.

Entre los plugins que dispone OpenDayLight se encuentran los siguientes: OpenFlow, OVSDB[24], NetConf [25] y SNMP[26] los cuales son los más importantes para la gestión de la infraestructura SDN.

Dispositivos de Red: En la capa final de la arquitectura del controlador OpenDayLight se encuentran todos los dispositivos de conectividad de la infraestructura de red, estos pueden ser físicos o virtuales.

[24] OVSDB (*Open vSwitch Database Management Protocol*) es un protocolo de configuración OpenFlow que está diseñado para gestionar implementaciones de Open vSwitch.
[25] NetConf (*Network Configuration Protocol*) es un protocolo de gestión de red desarrollado y estandarizado por el IETF.
[26] SNMP (*Simple Network Management Protocol*) es un protocolo de la capa de aplicación que facilita el intercambio de información para la administración y gestión entre dispositivos de red.

1.7.2. Herramientas para el manejo de OpenDayLight

El controlador OpenDayLight está implementado en Java y se ejecuta a través de una Máquina Virtual Java (JVM). Las ventajas que se tienen al emplear esta tecnología es que el controlador puede ser utilizado en cualquier sistema operativo que tenga soporte para Java.

El controlador OpenDayLight utiliza herramientas para la construcción, ejecución y comunicación entre las aplicaciones desarrolladas como también con las aplicaciones propias del controlador; en esta sección se describirán las herramientas que utiliza el controlador.

Para la construcción de aplicaciones utiliza Apache Maven; para la gestión de los servicios utiliza el entorno de ejecución OSGi [27]al cual lo llama Karaf, dicha herramienta proporciona la comunicación entre las aplicaciones en ejecución en el mismo espacio de direcciones.

1.7.2.1. Maven

Maven es una herramienta que se utiliza para automatizar el proceso de compilación y generación de ejecutables mediante código fuente de proyectos de Java [27].

Esta herramienta dispone de un archivo llamado pom.xml (*Project Object Model*), el cual tiene un formato XML[28] y está encargado de configurar la construcción del proyecto así como también de las dependencias con otros módulos y componentes externos. Maven tiene dos puntos importantes, el primero describe la arquitectura del software y el segundo punto es la descripción de las dependencias [28].

Maven se encuentra basada en una secuencia de fases bien definidas para obtener un resultado esperado, a esto se lo define como ciclo de vida que a continuación se lo describe:

- **Validación:** Se encarga de comprobar que todos los *plugins* como los códigos fuentes necesarios se encuentren en el proyecto.
- **Compilación:** Compila todos los archivos con extensión java
- **Test:** Se encarga de realizar pruebas sobre el código compilado utilizando pruebas unitarias.
- **Empaquetado:** Con el código ya compilado se genera un archivo ejecutable.

[27] OSGi OSGi define una infraestructura para el desarrollo de aplicaciones basadas en bundles dentro de una máquina virtual Java JVM

[28] XML (*eXtensible Markup Language*) es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por el *World Wide Web Consortium*.

- **Pruebas de integración:** Se ejecutan pruebas de integración en el proyecto.
- **Verificación:** Se verifica tanto la validez como la calidad del proyecto desarrollado.
- **Instalación:** El archivo ejecutable se copia en el repositorio local de Maven y por medio de esta acción los artefactos[29] generados pueden ser utilizados en otros proyectos.

En la Figura 1.21 se puede observar la estructura de directorios y archivos que genera Maven para un proyecto, la cual tiene una estructura en la organización para la creación de los archivos. Los directorios que se generan con Maven se encuentran constituidos por dos subdirectorios que se presentan a continuación:

- **scr**: este subdirectorio se encuentra constituido por el código desarrollado para la implementación del proyecto Maven. Además, dentro de este directorio, se encuentran dos subdirectorios: **main**, en este subdirectorio se encuentra el archivo principal del proyecto; **test,** en este se encuentran todas las pruebas realizadas al proyecto.
- **target:** este subdirectorio es utilizado para almacenar los archivos ejecutables de un proyecto Maven.

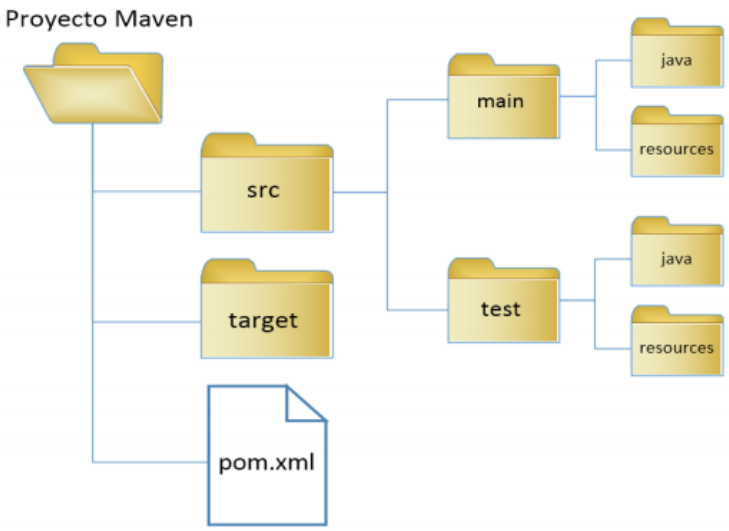

Figura 1.21 Estructura de archivos de Maven **[20]**

1.7.2.2. OSGi

OSGi es un sistema modular para aplicaciones Java, al elemento fundamental se lo denomina *bundles*, además contiene un archivo con el nombre de manifiesto donde se

[29] Artefacto: es un componente de software que se lo puede incluir en un proyecto como una dependencia usualmente tienen la extensión jar o war.

encuentran especificadas las interacciones con otros modelos y con el ciclo de vida. OSGi se encuentra en desarrollo desde 1999 por OSGi *Alliance* [29], la cual tenía como objetivo la definición de descripciones abiertas de software para el desarrollo de plataformas compatibles [30], [31].

El controlador OpenDayLight utiliza la especificación OSGi para el despliegue, ejecución, carga y descarga de forma dinámica de los *bundles*, de forma más concreta OSGi se encarga de la gestión de los *bundles* que se encuentran en el controlador OpenDayLight.

1.7.2.3. Apache Karaf

Karaf es un entorno de ejecución basado en OSGi que implementa un contenedor ligero que permite almacenar varios tipos de *bundles* los cuales pueden ser desplegados dentro del controlador OpenDayLight [32].

Karaf permite utilizar las implementaciones de OSGi: Apache Felix o Eclipse Equinox, proporcionando opciones y características complementarias a OSGi. Karaf es un contenedor ligero que brinda un conjunto de de servicios como el empaquetamiento e instalación de *bundles* y *feature*[30] [33].

El controlador OpenDayLight utiliza Karaf desde la versión Helium y se mantiene hasta la última versión Nitrogen.

En la Figura 1.22 se puede observar la interacción que existe en el controlador OpenDayLight tanto con OSGi y Apache Karaf donde la subcapa SAL es un feature de Karaf. Karaf ofrece las siguientes características: configuración dinámica, gestión remota, consola de ejecución, despliegue dinámico, sistemas de registros, y seguridad entre otras.

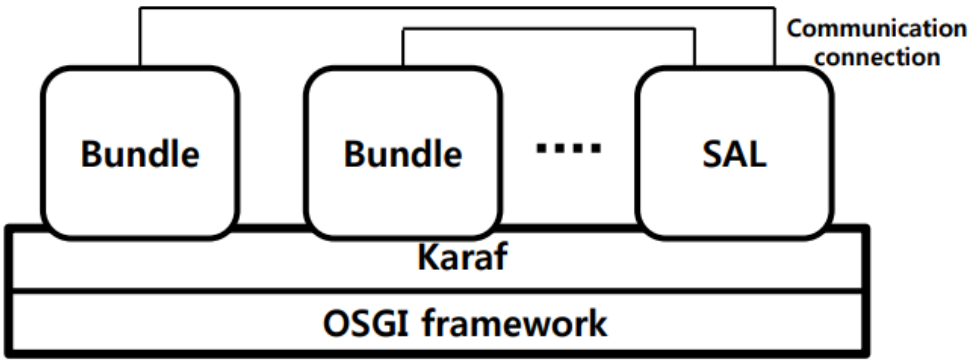

Figura 1.22 Estructura de Karaf dentro del controlador OpenDayLight **[34]**

[30] *Feature*: En el contexto de OpenDayLight un *Feature* es un conjunto de bundles que actúan como una unidad instalable.

1.8. NFV (Network Functions Virtualization)

Los conceptos iniciales de NFV surgen en el año 2012, donde se agruparon los principales proveedores de servicios de telecomunicaciones para generar las especificaciones de NFV dentro de la ETSI[31] [35].

El objetivo que tiene el grupo es la creación de recomendaciones que permitan la implementación de NFV de forma rápida. El objetivo que tiene NFV es separar o desacoplar las funciones de red de los dispositivos dedicados y transferirlos a servidores virtuales en los cuales se pueden albergar múltiples funciones de red en un mismo servidor físico [36].

En las redes que se implementan actualmente, las empresas proveedores de telecomunicaciones cuando desean brindar un nuevo servicio con altas prestaciones con propósitos particulares como son: cortafuegos, balanceadores de carga, IDS[32], DPI[33], NAT[34], Security Appliance, Home Gateway, entre otros servicios de red a implementarse tienen algunos problemas que a continuación se presenta algunos de ellos.

Se incrementa la complejidad de la infraestructura así como también se extiende el tiempo de implementación. Además aumenta considerablemente tanto el OPEX[35] y CAPEX[36] de los operadores o de la empresa debido a que los nuevos dispositivos a utilizar son de uso específico, esto conlleva a los siguientes problemas: son poco flexibles, problemas de eficiencia energética y su gestión es costosa [37] [38].

Según la ETSI *Network Functions Virtualization* utiliza técnicas de virtualización para convertir a los dispositivos de red en simples módulos virtuales [38]. Los beneficios que se obtienen al utilizar NFV son los siguientes:

- Reducción del CAPEX y del OPEX
- Reducción del consumo energético

[31] ETSI (*European Telecommunications Standards Institute*) es una organización de estandarización independiente, sin fines de lucro de la industria de las telecomunicaciones de Europa, con proyección mundial.
[32] IDS (*Intrusion Detection System*): es un programa de detección de accesos no autorizados a un computador o a una red.
[33] DPI (*Deep packet inspection*): es una forma de filtrado de paquetes de red que examina la parte de datos o la cabecera de un paquete buscando el incumplimiento del protocolo, virus, spam, intrusiones o criterios definidos para decidir si el paquete puede ser transmitido.
[34] NAT (*Network Address Translation*): es un mecanismo utilizado por los routers IP para intercambiar paquetes entre dos redes que asignan mutuamente direcciones incompatibles.
[35] OPEX (*OPerating EXpense*) es un costo permanente para el funcionamiento de un producto, negocio o sistema.
[36] CAPEX (*CAPital EXpenditures*): son inversiones de capital que crean beneficios. Un CAPEX se ejecuta cuando un negocio invierte en la compra de un activo fijo.

- Reducción del tiempo de implementación de un nuevo servicio de telecomunicaciones
- Incremento de las ganancias
- Despliegue de nuevos servicios con un bajo riesgo de problemas
- Flexibilidad para escalar rápida y dinámicamente diferentes servicios en función de la demanda

NFV, tiene el objetivo de virtualizar a las funciones de red implementadas en los equipos de infraestructura, convirtiéndoles en aplicaciones de software que tienen la capacidad de ejecutarse en hardware de propósito general.

Con NFV se consigue que cuando se desee implementar una nueva función de red no se necesite adquirir un nuevo equipo propietario, solo se requiera el software, el cual se lo implementará como una función de red virtualizada la cual funcionará sobre un servidor de propósito general.

En la Figura 1.23 se presenta un ejemplo de implementación de NFV, donde se tiene elementos de red que tradicionales van que encuentran virtualizados.

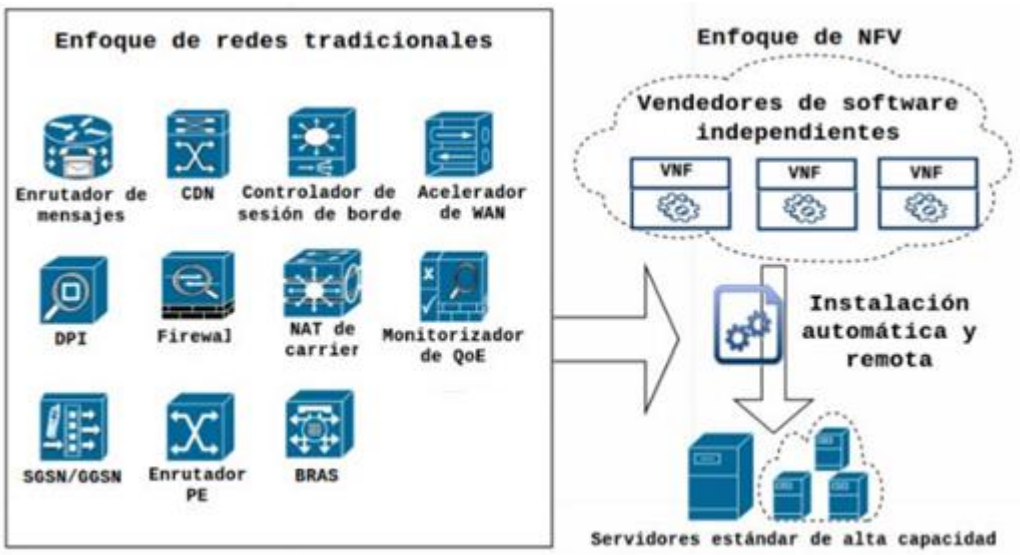

Figura 1.23 Ejemplo de implementación de NFV **[35]**

1.8.1. Arquitectura NFV

La arquitectura NFV tiene tres componentes principales de alto nivel: las VNFs (*Virtual Network Function*), NFVI (*Network Function Virtualization Infrastructure*) y la orquestación y gestión de NFV (NFV-MANO (*Management and Network Orchestration*)). En la Figura 1.24 se puede observar a cada uno de los elementos antes expuestos, a cada elemento se le ha asignado un calor específico para poder diferenciarlo.

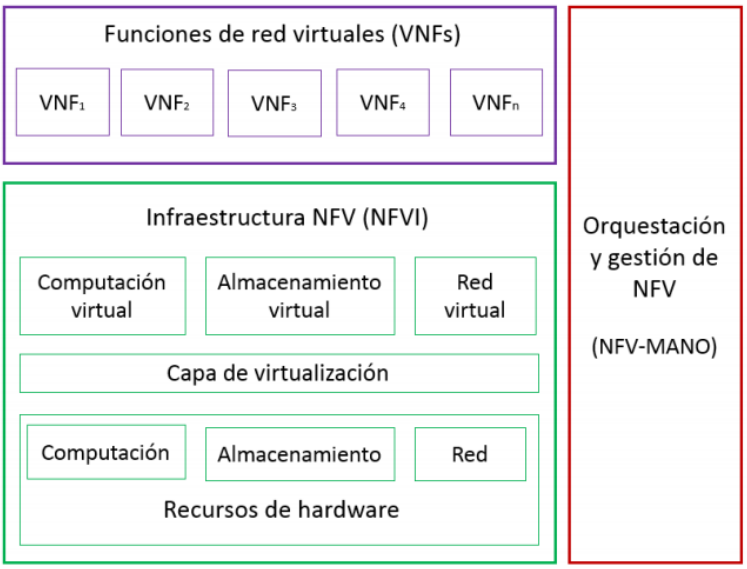

Figura 1.24 Arquitectura de NFV **[35]**

A continuación se describe cada uno de los elementos mencionados anteriormente:

1.8.1.1. VNF

VNF es una instancia de función de red la cual puede estar instalada en una sola máquina virtual.

Existen diferentes escenarios en los que se puede desplegar una VNF, una de ellas es que se instancie en un sistema operativo dedicado, el cual se encuentra sobre una máquina virtual o que se instale sobre un *bare-metal*[37] directamente.

1.8.1.2. NFVI

NFVI es el entorno en el que se ejecuta una VNF, esto incluye de recursos físicos, virtuales y la capa de virtualización. Los recursos físicos de NFVI se pueden virtualizar paro instanciar las VNF. Los recursos virtuales en NFVI se utilizan cuando se abstrae los recursos físicos (cómputo, almacenamiento y conectividad) que serán utilizados por las VNF. Para la implementación de NFV se utiliza la capa de virtualización como que fuera un hipervisor para establecer recursos de hardware de forma dinámica a cada VNF que lo requiera.

1.8.1.3. Orquestación y Gestión de NFV

NFV-MANO, u Orquestación y Gestión de NFV, es el módulo que se encarga de la administración y gestión de NFV; se encuentra constituido por un Orquestador y por gestores de VNFs y de infraestructura virtualizada. NFV-MANO gestiona y brinda

[37] *Bare-metal*: también conocido como hipervisor de Tipo 1, es un software de virtualización que se ha instalado directamente en el hardware de un equipo informático.

mantenimiento de los repositorios de datos, también se utiliza para el intercambio de información entre todos los módulos de la estructura de una NFV.

Usualmente existe un solo Orquestador dentro de la arquitectura NFV, el cual maneja todos los módulos, también se encarga del ciclo de vida de VNF, actualizaciones, consultas y escalamiento de servicios.

El Orquestador proporciona transparencia dentro de la gestión de la infraestructura y maneja la administración de recursos, incluyendo:

- Supervisión de los recursos para la NFVI.
- Asignación de herramientas de virtualización.
- Gestión continua y cambios de asignación para optimizar la utilización y eficiencia de los recursos de la infraestructura.

1.8.2. Relación entre NFV y SDN

El paradigma NFV puede trabajar con otras tecnologías, como las redes definidas por software [39] [40]. Tanto NFV así como SDN trabajan de forma compatible pero no es indispensable que funcionen al mismo tiempo ni son dependientes entre sí. La arquitectura de NFV suministra la infraestructura sobre la cual se puede implementar las SDN; esta combinación establece un valor agregado a la infraestructura de red que se describe a continuación:

- Como las SDN pueden ser implementadas de forma virtual ayudan a mejorar los tiempos de aprovisionamiento y administración de las NFV por lo que un cambio será fácil de implementarlo.
- El plano de control de los equipos de red (SDN) así como las funciones de red (NFV) se los puede migrar de un equipo físico costoso a un servidor de propósito general, optimizando el uso del equipo como también el consumo de energía.
- El plano de control se abstrae de la infraestructura de red utilizada para el envío y recepción de información, y se normaliza los datos para la innovación en la infraestructura de red como también de las funciones que las incluyen, sin la necesidad de actualizar los equipos de red.

En la Figura 1.25 se puede observar que tanto NFV como SDN son complementarias entre sí, pero como ya se mencionó, pueden trabajar independientemente. Sin embargo, al utilizar estas tecnologías de forma conjunta, se obtiene mayor potencial y valor agregado. La parte del plano de control se encargaría de manejar las redes definidas por software y las funciones de red se encargarían de manejar NFV.

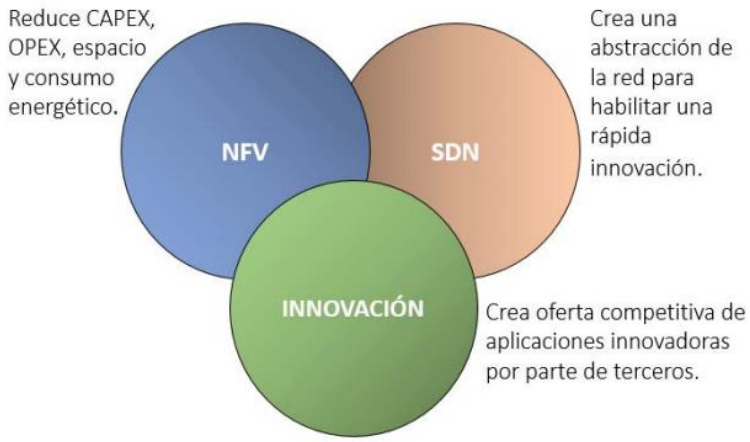

Figura 1.25 Relación existente entre NFV y SDN **[35]**

1.8.3. Proyectos NFV

En la actualidad existen varios proyectos sobre NFV, la mayoría de estos se encuentran basados en código abierto. Los proyectos que complementan a NFV son OpenDayLight, OpenStack, entre los más importantes; estos últimos tienen grupos de trabajo bien estructurados para introducir los cambios necesarios para que se puedan adaptar a la tecnología NFV [41]. NFV-ISG (*Industry Specification Group*) participa directamente con OpenDayLight y OpenStack para la generación de software de código abierto que facilite la adaptación y despliegue de NFV [42].

1.8.3.1. OPNFV

La plataforma OPNFV (*Open Platform for* NFV) fue creada en el año 2014 por Linux Foundation. OPNFV establece una plataforma integral de código abierto para proveedores de servicio, para avanzar en la aceptación de NFV [43]. A continuación se detallan los objetivos que tienen la plataforma OPNFV [35]:

- Desarrollar una plataforma de código abierto que pueda implementar y modificar la funcionalidad y arquitectura de NFV.
- Incluir la participación proactiva de los principales usuarios finales, para validar si OPNFV satisface las necesidades de la comunidad de usuarios.
- Contribuir y participar en proyectos de código abierto que serán aprovechados en la plataforma de referencia OPNFV.
- Establecer un medioambiente abierto de soluciones NFV, basadas en estándares y software de código abierto.
- Promover OPNFV como una plataforma de referencia de código abierto para funciones NFV.

1.9. Computación en la nube

Computación en la nube tiene recursos y servicios de computo como infraestructura de red, aplicaciones y plataformas los cuales son ofrecidos y consumidos como un servicio a través de Internet sin que los usuarios finales tengan conocimiento de lo que están utilizando por debajo [44].

Computación en la nube tiene un nuevo modelo de prestación de servicios ya que se los brinda a través de Internet. Computación en la nube tiene dos conceptos bien definidos los cuales son: abstracción y virtualización que se describen a continuación [45].

- **Abstracción:** este concepto se refiere a que el usuario final no interviene en los detalles de la implementación. Además, las aplicaciones tienen características adicionales que el usuario final desconoce, como las siguientes: si se ejecuta en un hardware físico o virtualizado, se desconoce la ubicación de la información que se almacena, la gestión y administración de la plataforma la realizan terceros. Una ventaja que se obtiene es el acceso desde cualquier lugar que se disponga de conexión a Internet.
- **Virtualización:** este concepto hace referencia a la tecnología para crear múltiples entornos simulados que compartan un mismo hardware pero que sean independientes entre sí mediante la compartición y asignación de recursos físicos de cómputo.

Computación en la nube tiene diferencias marcadas con los servicios computacionales ya que en la computación en la nube cuando se desea cambiar la capacidad de cómputo se lo realiza de forma rápida y eficiente. El catálogo de servicios es amplio y estandarizado, estos se los pueden implementar de forma flexible y adaptiva según el consumo que tenga la aplicación desplegada.

Según la IEEE[38], se define a la Computación en la Nube como un paradigma en el que la información se almacena de manera permanente en servidores de Internet y la información de los clientes se envía a cachés temporales.

Por otro lado, la NIST[39] define a la Computación en la Nube o *Cloud Computing* de la siguiente manera: "*Cloud Computing* es un modelo que permite el acceso bajo demanda y a través de la red a un conjunto de recursos compartidos y configurables (redes,

[38] IEEE (*Institute of Electrical and Electronics Engineers*) es una asociación mundial de ingenieros dedicada a la estandarización y el desarrollo en áreas técnicas

[39] NIST (*National Institute of Standards and Technology*) La misión de este instituto es promover la innovación y la competencia industrial en Estados Unidos mediante avances en metrología, normas y tecnología de forma que mejoren la estabilidad económica y la calidad de vida.

servidores, capacidad de almacenamiento, aplicaciones y servicios) que pueden ser rápidamente asignados y liberados con una mínima gestión por parte del proveedor del servicio" [44].

NIST define cinco características esenciales que cualquier servicio de *Cloud Computing* debe ofrecer a sus usuarios:

- **Servicio bajo demanda:** El usuario puede establecer las capacidades de cómputo como por ejemplo almacenamiento, cantidad de núcleos de procesador o tiempo de trabajo del servidor según las necesidades que tenga, esto se lo tiene que hacer de forma dinámica sin requerir la interacción humana.
- **Amplio acceso a la red:** Los recursos utilizados por la computación en la nube son accesibles, se encuentran disponibles por medio de Internet.
- **Compartición de recursos:** Los recursos de cómputo del proveedor de servicio son compartidos dinámicamente, y pueden ser asignados de acuerdo a la demanda del consumidor. El cliente por lo general no tiene ningún control o conocimiento sobre la ubicación exacta de los recursos asignados.
- **Elasticidad:** Los recursos de cómputo pueden ser asignados y liberados rápidamente de forma automática, para escalar rápidamente, según la demanda de los usuarios. Por lo general, para el usuario final son transparentes los cambios que se realizan por medio de la computación en la nube.
- **Servicio a la medida:** El proveedor de servicio puede calcular el consumo de los recursos de cómputo que los usuarios están utilizando, lo que facilita el pago por el uso de los servicios prestados.

1.10. OpenStack

El proyecto OpenStack inició el desarrollo de sus actividades en el año 2010 por medio de la empresa Rackspace Cloud y por la NASA [46], creando una comunidad abierta en la cual se tiene la colaboración de desarrolladores expertos en Computación en la Nube que tienen como objetivo desarrollar una plataforma de código utilizando la licencia Apache[40] [47].

OpenStack tiene el objetivo de brindar soluciones para cualquier tipo de Computación en la Nube agregando escalabilidad, prestación de servicios amplios y un soporte técnico brindado por una comunidad ya establecida [48].

[40] Licencia Apache: es una licencia de software libre permisiva creada por Apache Software Foundation. La licencia Apache requiere la conservación del aviso de derecho de autor y el descargo de responsabilidad.

En la Figura 1.26 se presenta la arquitectura básica que utiliza la plataforma OpenStack. Esta plataforma puede manejar los siguientes recursos: cómputo, almacenamiento e infraestructura de red.

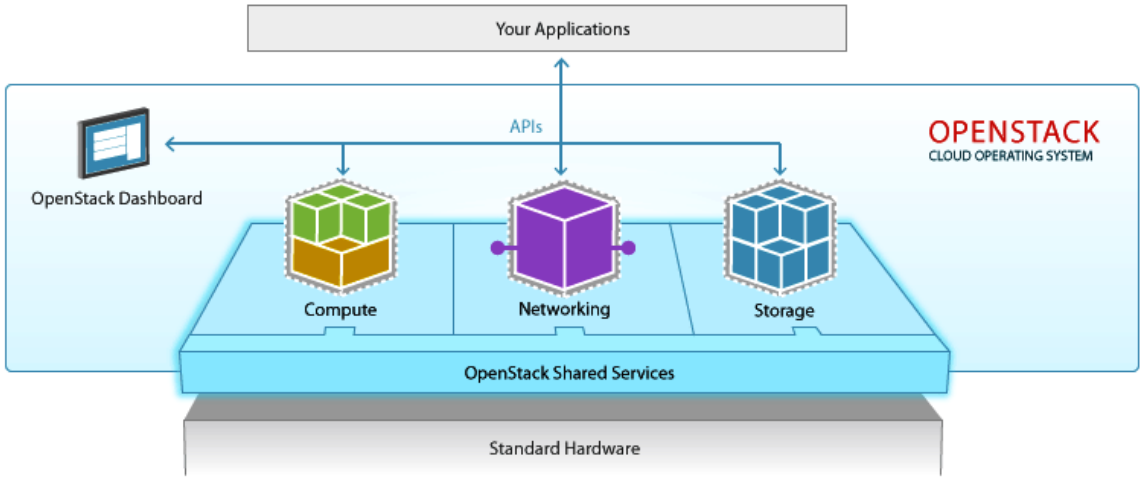

Figura 1.26 Arquitectura de OpenStack **[49]**

1.10.1. Servicios de OpenStack

Los servicios de OpenStack van a depender del tipo de nube que se desea implementar ya que se van a diferenciar por los requerimientos y servicios que se crearan en cada nube. Los servicios básicos que se utilizan para este proyecto de titulación son los siguientes:

1.10.1.1.*Compute* (Nova)

El servicio Nova está encargado de manejar el ciclo de vida de las instancias en la plataforma OpenStack. Este servicio tiene el objetivo de cargar y asignar recursos a cada instancia; también administra los procesos de cómputo entre diferentes instancias.

1.10.1.2.Dashboard (Horizon)

El servicio Horizon es la interfaz gráfica que se visualiza por medio de un portal web que permite interactuar con los servicios instalados en la plataforma OpenStack. Esta interfaz gráfica permite la administración y gestión de Openstack.

1.10.1.3.Networking (Neutron)

El servicio Neutron es el encargado de la infraestructura de red como servicio para los demás módulos de la arquitectura de OpenStack; cuenta con un API el cual es utilizado por los usuarios para definir la infraestructura de red y los dispositivos que van a estar

conectados. Además permite la gestión y administración de la red que interconecta las instancias y los dispositivos virtuales de conectividad.

1.10.1.4.Storage

El servicio de almacenamiento se encuentra dividido en dos servicios que se los describe a continuación:

- **Object storage (Swift):** El servicio Swift almacena y recupera objetos no estructurados mediante la utilización de un API HTTP basada en RESTful[41]. La principal ventaja de *Object storage* es que se pueden distribuir fácilmente objetos a través de varios nodos *back-end* de almacenamiento y se los puede encontrar mediante su ID.
- **Block Storage (Cinder):** El servicio Cinder ofrece almacenamiento en bloques para las instancias que se encuentran en ejecución y cuenta con un driver para facilitar el manejo y la creación del almacenamiento en bloques. *Block Storage* facilita la modificación de archivos ya que tiene acceso a los bloques específicos. Este tipo de almacenamiento es utilizado para operaciones de alta actividad como almacenamiento en caché, operaciones de bases de datos, archivos de registro.

1.11. DevOps

DevOps abarca muchas disciplinas tecnológicas. DevOps es práctico y participativo, por lo que se tiene que comprender tanto los conocimientos técnicos así como los aspectos culturales de colaboración efectiva e ininterrumpida entre el desarrollo y operaciones [50].

La palabra DevOps se encuentra conformada por dos palabras *Development* y *Operations*. DevOps se establece como una práctica en la que se fomenta la colaboración entre diferentes disciplinas en el desarrollo de software [51] [52].

El origen de la palabra y las primeras ideas de DevOps se las puede adjudicar al desarrollador de software Patrick Debois quien es considerado como el padre de DevOps [53].

Patrick tenía inconvenientes debido a la división que existía entre los desarrolladores y el personal de operaciones, por lo que busco a personas que tenían el mismo problema, se unieron para crear un grupo y solventar este problema. No fue hasta el año 2009 cuando se presentó la conferencia de O' Reilly Velocity: "10+ *Deploys per Day: Dev and Ops*

[41] RESTful: Los servicios web RESTful son una forma de proporcionar interoperabilidad entre los sistemas informáticos en Internet. Los servicios Web compatibles con REST permiten a los sistemas solicitantes acceder y manipular representaciones textuales de recursos Web utilizando un conjunto uniforme y predefinido de operaciones sin estado.

Cooperation at Flickr", donde se presentan los inconvenientes que existen entre el departamento de desarrollo y operaciones [54].

DevOps tiene su origen y raíces en los principios de desarrollo ágil[42] de software. El extracto del manifiesto sobre las metodologías ágiles se presenta a continuación: “Individuos e interacciones sobre procesos y herramientas. Software funcionando por encima de la documentación. La colaboración con el cliente sobre la negociación de contratos. La respuesta al cambio por encima del seguimiento de un plan” [55]. De este manifiesto se puede concluir que existe una estrecha relación con DevOps, pero el que se encuentra más relacionado es el principio "Individuos e interacciones sobre procesos y herramientas".

DevOps tiende a resaltar las interacciones entre los individuos y que la tecnología facilita que las interacciones se produzcan fluidamente, eliminando los obstáculos existentes en una organización.

Un objetivo central de DevOps es la automatización y la entrega continua de procesos entre el departamento de operaciones y desarrollo. En otras palabras DevOps intenta la automatización de tareas repetitivas y tediosas para dejar más tiempo para la interacción humana como valor agregado.

En la Figura 1.27 se presenta a DevOps como la intersección entre el desarrollo de software, operaciones de tecnología y la garantía de calidad de procesos.

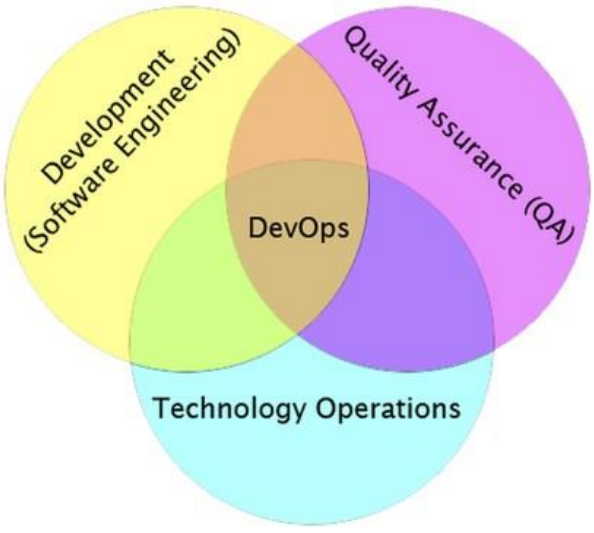

Figura 1.27 Interacción de DevOps **[56]**

[42] Desarrollo ágil: envuelve un enfoque para la toma de decisiones en los proyectos de software, que se refiere a métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto.

DevOps maneja cinco principios que se los describen a continuación:

- DevOps no es una metodología, es un movimiento que pretende cambiar el modo de funcionamiento del departamento de tecnología.
- DevOps intenta derrumbar los muros entre desarrollo y operaciones, con lo que se puede mejorar la generación de resultados.
- DevOps intenta implantar una cultura de compartición de conocimientos, para obtener una retroalimentación positiva.
- Los grupos aislados de conocimiento deben desaparecer.
- DevOps se genera dentro de cada empresa.

1.11.1. Razones para implementar DevOps

Los beneficios de utilizar DevOps son innegables cuando se los realiza correctamente. Algunos de los beneficios que se obtienen se listan a continuación [56]:

- La velocidad de respuesta ante cambios
- Tiempo medio para resolver problemas
- Tiempo de actividad mejorado
- Aumento del número de desarrollos
- Habilidades cruzadas entre equipos

Al implementar DevOps, se promueve la eliminación de tareas repetitividas, medición y automatización. La adopción de la automatización mejora la velocidad del cambio, aumenta el número de implementaciones que un equipo puede hacer en un día y mejora el tiempo de lanzamiento de una solución. La automatización del proceso de despliegue, permite a los equipos llevar las correcciones a producción rápidamente.

De la automatización se puede obtener otro producto, el tiempo promedio para resolver y solucionar los problemas de infraestructura de forma eficiente. Si los cambios de infraestructura o de red se automatizan, pueden aplicarse de manera eficiente en comparación a lo que se realizarían manualmente. Los cambios manuales dependen de la velocidad del ingeniero que implementa el cambio por otro lado, un script automatizado es mucho más rápido y los tiempos de implementación pueden ser medidos con mayor precisión.

Implementar DevOps también significa medir y monitorear los procesos eficientemente, por lo que tener un monitoreo efectivo es crucial en una infraestructura, ya que significa que el ritmo en el cual se puede llevar a cabo el análisis, se reduce y se puede disminuir el tiempo de operación. Tener un monitoreo efectivo ayuda resolver inconvenientes, por

lo que cuando se produce un problema de producción, el origen del error se puede encontrar más rápido de lo que se demorarían muchos ingenieros que tienen que conectarse a consolas y servidores intentando depurar problemas.

En esta Tesis se utilizará DevOps para la creación de la función de red que solventa el problema de *shared bottleneck* de MP-TCP, esto se lo describirá en el Capítulo 2

2. DISEÑO E IMPLEMENTACIÓN

2.1. Introducción

En este capítulo se presentará el diseño e implementación de una solución del problema del *shared bottleneck* del protocolo MP-TCP mediante la utilización de SDN, usando OpenDayLight como el controlador.

También se describirá el uso de una herramienta de DevOps, la cual es necesaria para la automatización del despliegue de la solución implementada para el problema *shared bottleneck* como una función de red virtualizada.

2.2. Funcionalidades de la aplicación

Los requerimientos funcionales expresan el funcionamiento de la aplicación, es decir, la interacción de la aplicación en el entorno que se encuentra desarrollada, a continuación se presentan los requerimientos de la aplicación desarrollada:

- Se determinará el tipo de los paquetes que llegan a cada nodo; si no pertenecen al protocolo MP-TCP, se realizará una conmutación de capa 2. Para los paquetes de tipo MP-TCP, se decodificará la cabecera del protocolo para discriminar el tipo de mensaje MP-TCP.
- La aplicación determinará la topología de red compuesta por los nodos.
- Se encontrará los nodos en los que se producen los problemas de *shared bottleneck*, tanto en el extremo de recepción como en el de transmisión.
- La aplicación determinará las diferentes rutas que pueden ser utilizadas por los subflujos de una conexión MP-TCP.
- Se instalará las reglas en los switches necesarias para que los subflujos que crean una conexión MP-TCP, tengan diferentes rutas.

2.3. Diseño de la aplicación

Para el diseño de la aplicación que resuelve el problema del *shared bottleneck* de MP-TCP, se modeló cierto tipo de acciones que se pueden implementar mediante el uso de la aplicación; las cuales se pueden observar en el diagrama UML[43] de casos de uso que se encuentra en la Figura 2.1.

En el diagrama se utilizó como actor al administrador de red que es la única persona que puede manejar la aplicación.

[43] UML (*Unified Modeling Language*) es el lenguaje de modelado de sistemas de software por el *Object Management Group* (OMG).

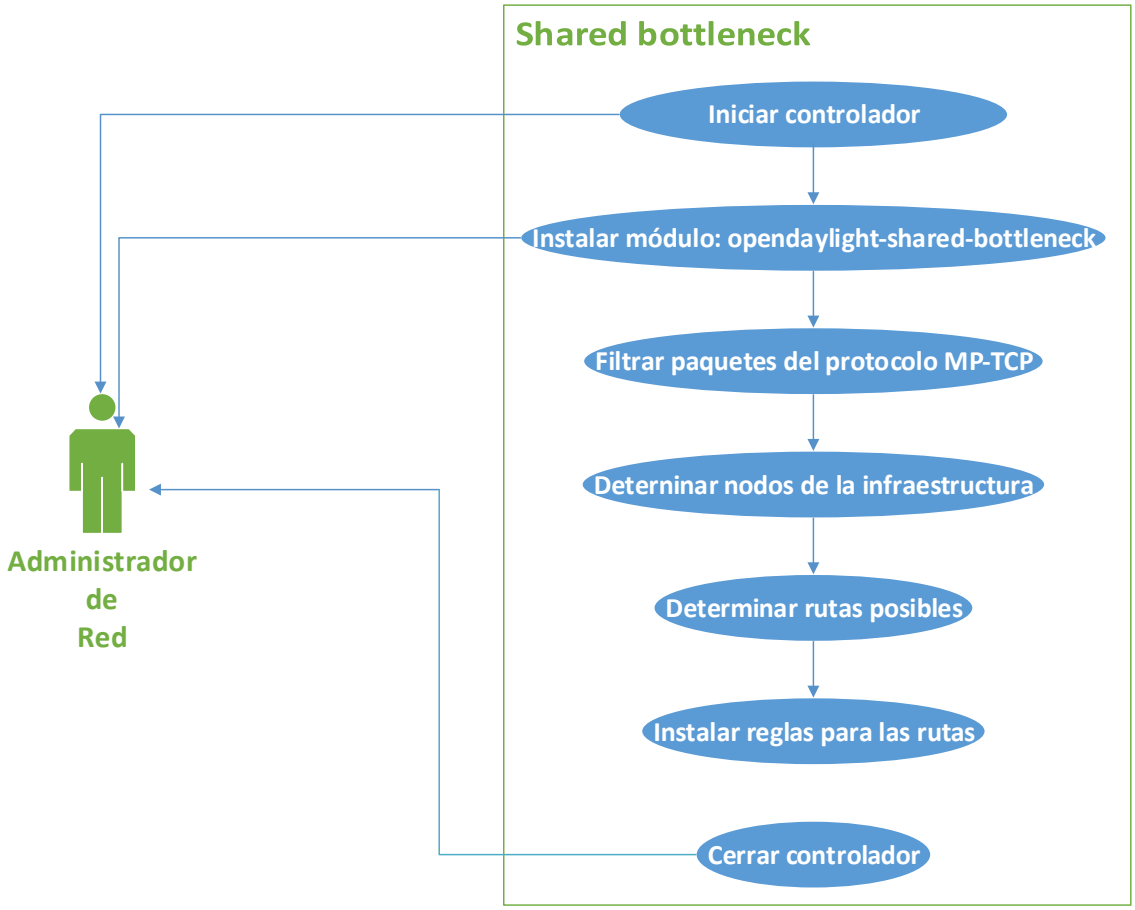

Figura 2.1 Diagrama de casos UML de la aplicación

La Tabla 2.1 presenta los casos de uso, con la respectiva explicación de las acciones que la aplicación realizará dependiendo de la tarea que ejecute el administrador de red.

Caso de uso	Aplicación *shared bottleneck* MP-TCP
Iniciar controlador	• Iniciar controlador • Iniciar Karaf
Instalar feature OpenDayLight-sharedbottleneck-mptcp	• Instalar el *feature* de la aplicación. • Inicia la aplicación OpenDayLight-sharedbottleneck-mptcp • Iniciar el *feature* L2Switch el cual se encarga de la conmutación de capa 2
Filtrar los mensajes MP-TCP decodificados	• Diferenciar los distintos tipos de mensajes de capa de transporte, sólo se utilizarán los mensajes MP-TCP en la aplicación. • El *feature* L2Switch se encargará de la

	conmutación de los mensajes que no sean MP-TCP.
Determinar todos los nodos que intervienen en la infraestructura de red	• Determinar los nodos y los *hosts* que se encuentran conectados entre sí y van a establecer la comunicación MP-TCP. • Determinar los nodos intermedios que componen la infraestructura de red; estos nodos son los que se presentan cuando los *hosts* no están directamente conectados al cuello de botella compartido.
Determinar las rutas posibles en la infraestructura de red	• Asignar pesos o distancias a cada enlace entre los nodos que intervienen en la infraestructura de red. • Realizar el Algoritmo de Dijkstra en la topología de red para obtener todas las rutas posibles.
Asignación de rutas	• Asignar todas las posibles rutas obtenidas del Algoritmo de Dijkstra a cada subflujo creado por el protocolo MP-TCP.
Instalación de reglas	• Instalar las reglas con el criterio de los 5 elementos de la tupla para crear la comunicación a través de los nodos de la infraestructura de red.
Cerrar controlador	• Cerrar la aplicación *shared bottleneck* MP-TCP. • Salir de Karaf.

Tabla 2.1 Casos de uso de la aplicación *shared bottleneck* MP-TCP

2.3.1. Solución propuesta para la aplicación que soluciona el problema de shared bottleneck del protocolo MP-TCP

La Figura 2.2 presenta el diagrama de flujo de la aplicación diseñada, la cual tiene como base la propuesta delineada en [5]. Lo que se añade a la solución propuesta en [5] fueron algunos cambios que a continuación se los describe:

- Solo una vez se utiliza el Algoritmo de Dijkstra en una conexión MP-TCP ya que cuando se lo aplica el algoritmo se conoce todos los posibles caminos que existe desde un *host* emisor a un *host* receptor que pueden ser utilizados los subflujos que crea MP-TCP.
- La aplicación utilizar múltiples conexiones MP-TCP a una infraestructura de red sin perder los beneficios de la aplicación es la velocidad de transmisión.
- La aplicación encuentra el lugar donde se produce el problema de *shared bottleneck*. Esta información es importante para solventar el problema de los nodos intermedios que más adelante se detalla.
- La conmutación de capa 2 se encuentra a cargo del *feature* L2switch con las reglas que instala en los switches

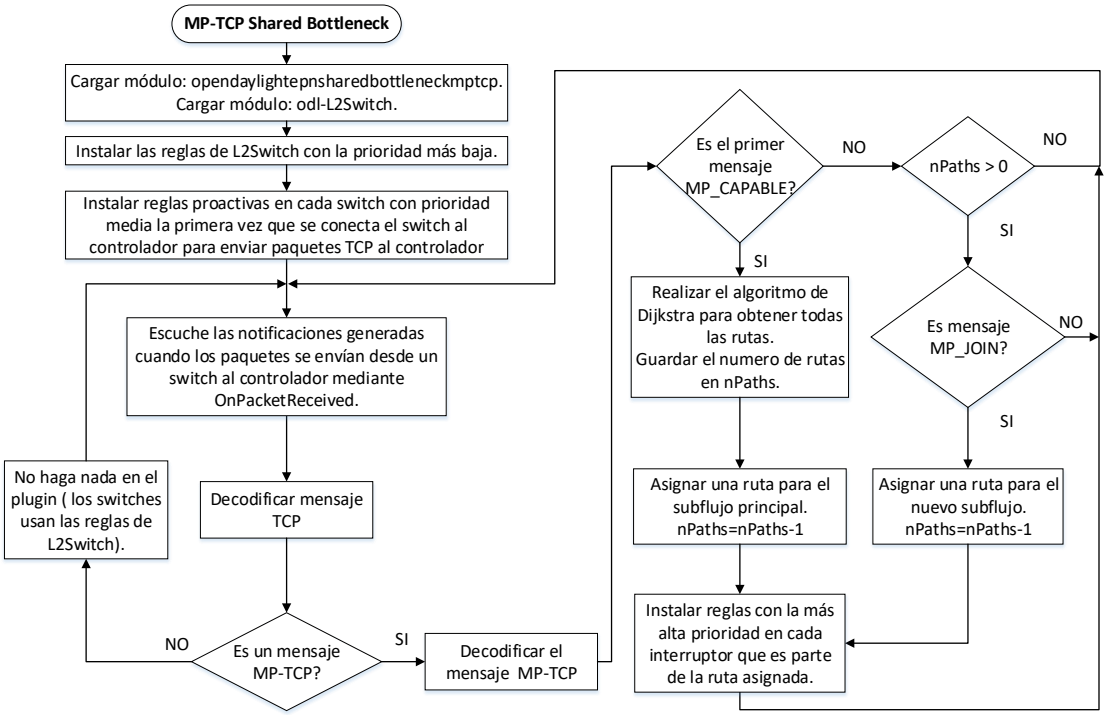

Figura 2.2 Diagrama de flujo de la aplicación

El primer paso de la aplicación es instalar el *feature* opendaylight-epn-sharedbottleneck-mptcp y odl-L2Switch en el controlador OpenDayLight. Al momento que los switches o nodos se conectan al controlador automáticamente se instalan reglas proactivas para que todo el tráfico de tipo TCP se dirija al controlador. Cuando ingresa un paquete TCP al switch, este lo envía al controlador para poder procesarlo mediante el módulo OnPacketReceived.

La siguiente acción que realizará el controlador es determinar si los paquetes TCP que ingresaron son del tipo MP-TCP. Para esto se tiene que decodificar y examinar el campo

de opciones de la cabecera TCP, el cual se encuentra establecido con el valor de 30 para paquetes MP-TCP, como fue descrito en el capítulo anterior.

En esta acción existen dos opciones: la primera es que el valor del campo opciones de TCP sea diferente a 30, para estos tipos de paquetes se debe realizar una conmutación de capa 2 mediante el *feature* L2Switch; por otro lado, si los paquetes que ingresan al nodo son de tipo MP-TCP (campo de opciones es 30), se debe realizar una nueva decodificación.

En este punto se decodifica el mensaje MP-TCP para determinar el tipo de mensaje que está ingresando al nodo, en este caso se pueden tener dos opciones ya que solo dos tipos de mensajes MP-TCP se utilizan en la aplicación desarrollada, MP_CAPABLE y MP_JOIN, que fueron descritos en el capítulo anterior.

La primera opción es cuando arriba un mensaje del tipo MP_CAPABLE, este mensaje es el primero que se envía para el establecimiento de la conexión del subflujo principal además se verifica que sea el primer mensaje enviado de este tipo.

La primera acción que se realiza cuando ingresa el primer mensaje MP_CAPABLE al controlador, es obtener toda la topología de red tanto de los switches, así como de los enlaces que los conectan, para tener una representación en forma de un grafo[44] donde los switches son representados por vértices o nodos.

Los switches son la unidad fundamental del grafo y las aristas corresponden a los enlaces ya que son la conexión entre dos vértices.

Con la infraestructura de red se puede aplicar el Algoritmo de Dijkstra[45] el cual determina la ruta más corta entre un nodo de origen y uno de destino, además de la ruta más corta se obtienen todas las posibles rutas entre estos dos nodos aunque no sean las más cortas.

En la Figura 2.3 se puede observar la transformación que se realizó de una infraestructura de red a la de un grafo.

Al aplicar el Algoritmo de Dijkstra a la infraestructura de red se necesita que cada arista tenga un peso o coste determinado; en este caso, a cada arista o enlace entre nodos se

[44] Grafo es un conjunto de objetos denominados vértices o nodos conectados por aristas, que representan a las relaciones que existen entre elementos de un conjunto.
[45] Algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de los vértices en un grafo con pesos en cada arista

le asigna un peso que está relacionado con la velocidad de transmisión, en la Figura 2.3 se presenta la asignación de pesos.

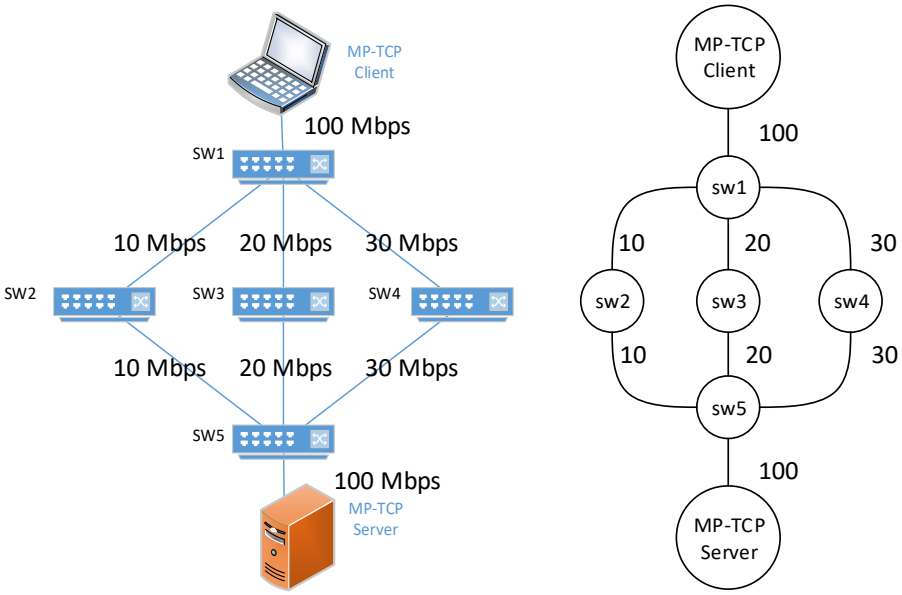

Figura 2.3 Representación de una infraestructura de red a un grafo

A la primera ruta que se obtiene al utilizar el Algoritmo de Dijkstra se la asigna al subflujo creado con los parámetros que tiene el mensaje MP_CAPABLE, para poder diferenciarlo de otros subflujos de una misma conexión.

Se utilizará el concepto de los 5 elementos de una tupla que se detalló en el capítulo anterior, solo se agrega una modificación en el protocolo de capa de transporte ya que TCP es constante para todos los subflujos de MP-TCP. Además, también se guarda el número total de rutas que se obtienen con el Algoritmo ya que con este valor se determina el número máximo de subflujos que se puede asignar a cada conexión MP-TCP.

Una vez que se asigna la primera ruta del Algoritmo de Dijkstra al subflujo MP-TCP, está ruta ya no se vuelve a utilizar para mejorar el *throughput* de la infraestructura de red, si se reutiliza esta ruta existe la posibilidad de saturar el enlace con una sola conexión MP-TCP.

Cuando al controlador llega un paquete MP_JOIN, corresponde a la segunda opción en la que se requiere la asignación de rutas, al igual que cuando se asigna la ruta principal, se tienen que utilizar los 5 elementos de la tupla para poder diferenciar una ruta de otra. Además de esta acción, se tiene que disminuir en una unidad al número de rutas totales cada vez que se asigna un camino a un subflujo, ya que la aplicación no permite asignar más rutas de las que permite la infraestructura de red a los subflujos.

Para que por las rutas, obtenidas por el Algoritmo de Dijkstra, puedan circular los flujos de datos, el controlador tiene que instalar las reglas en cada switch que intervenga en la comunicación. Los datos que va a atravesar por los switches, es un conjunto de mensajes MP_DSS, explicados en el capítulo anterior; estos paquetes son los encargados de transportar los datos que se desea transmitir entre un par de *hosts*.

Los mensajes MP_DSS de cada subflujo tienen asignados 5 elementos de una tupla; usualmente, en la transmisión de datos entre un par de *hosts,* lo que diferencia una tupla de otra es el puerto de transmisión y el de recepción; basado en eso se crean diferentes tipos de reglas para distinguir cada ruta que se establece con los mensajes MP_CAPABLE y MP_JOIN. Para diferenciar cada conexión MP-TCP, se deben utilizar las llaves que intercambian los mensajes MP_CAPABLE, las cuales son generadas por el *host* emisor y son números únicos.

2.4. Implementación de la aplicación

La sección de implementación de la aplicación se la dividió en cuatro partes para que la explicación sea más comprensible. Además, en la última sección se presentan y explican secciones de códigos importantes de la aplicación. La aplicación se desarrolló con el IDE[46] de Eclipse ya que en la página oficial de OpenDayLight se recomienda su uso.

2.4.1. Desarrollo de la aplicación en el controlador OpenDayLight

Para la desarrollo de la aplicación se utilizó como base la estructura del proyecto presentado en la página web de SDN Hub [57]. En la Figura 2.4 se observa la estructura de directorios que tiene el proyecto desarrollado por SDN Hub. A continuación se describen los archivos importantes para la implementación de una aplicación:

- **pom.xml:** el archivo pom de primer nivel especifica los poms de niveles inferiores.
- **acl**, **Netconf-exercise** y **Tapapp** se eliminaron y solo se modificó Learning-switch para la aplicación desarrollada
- .**Learning-switch:** es una aplicación que funciona como un L2switch o Hub.
- **Commons/utils:** esta carpeta contiene archivos de java los cuales están enfocados en clases y métodos del protocolo OpenFlow.
- **features/pom.xml:** en este archivo pom se especifican las aplicaciones que se desean implementar con todas las dependencias necesarias para cargar en Karaf.
- **distribution/opendaylight-karaf/pom.xml:** este archivo contiene instrucciones para la creación de la carpeta donde se ejecutará la consola de Karaf.

[46] IDE (*Integrated Development Environment* o entorno de desarrollo integrado) es una aplicación que proporciona servicios integrales para facilitarle al programador el desarrollo de software.

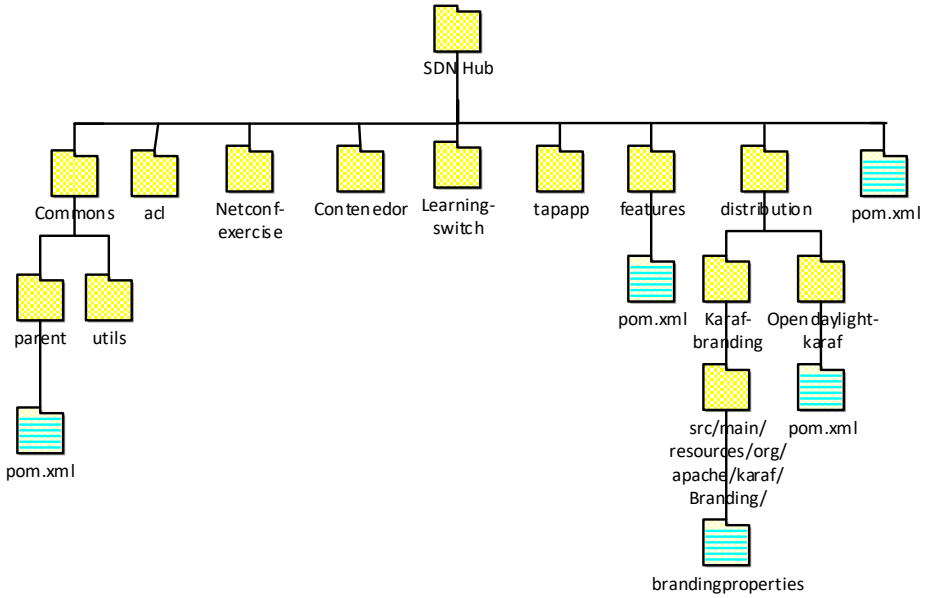

Figura 2.4 Estructura del proyecto SDN Hub

La Figura 2.5 presenta las modificaciones necesarias que se realizaron en la estructura del proyecto SDN Hub para implementar la aplicación *shared bottleneck* de MP-TCP. En dicha figura se pueden diferenciar tres colores los cuales indican si los archivos fueron o no modificados, o si fueron creados en su totalidad.

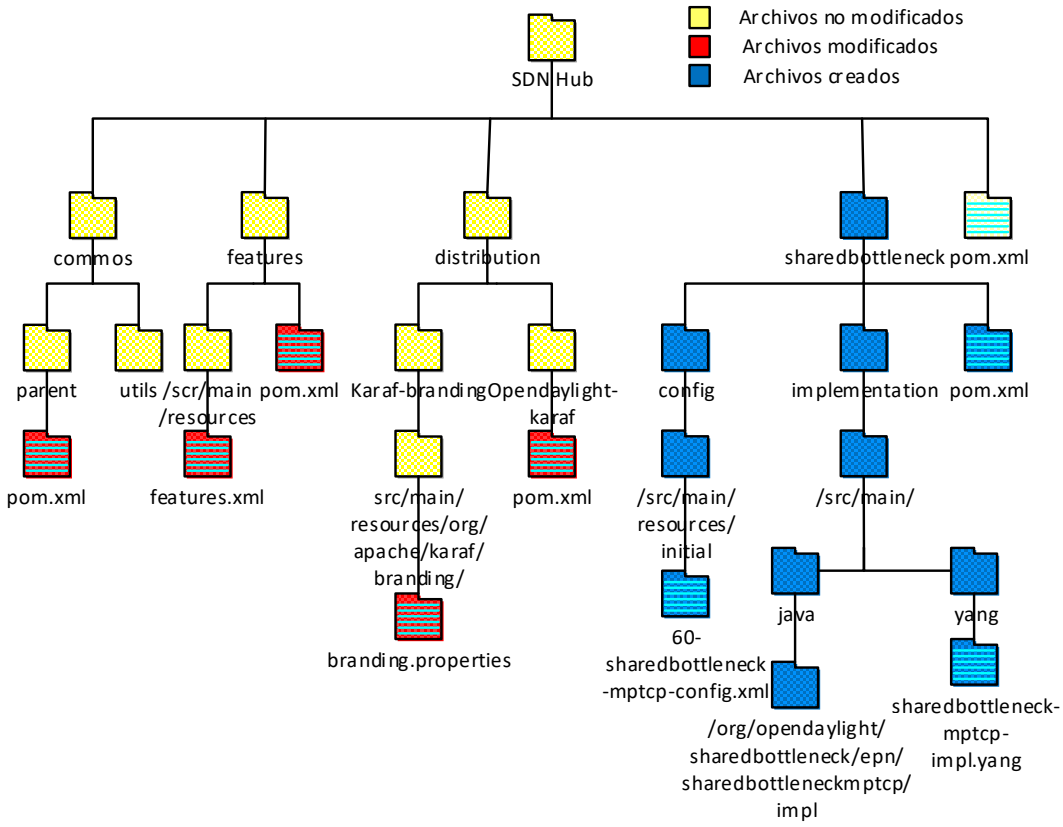

Figura 2.5 Estructura modificada del proyecto de SDN Hub

Como se mencionó, la estructura del proyecto SDN Hub se la dividió en tres secciones, los archivos y carpetas de color azul y color rojo se las van a describirá a continuación, las de color amarrillo ya se los describió anteriormente.

A la primera sección se la denominó “Archivos creados”, se encuentra representada en la Figura 2.5 con color azul.

Como la carpeta principal tiene el nombre de sharedbottleneck, en dicha carpeta se localizan archivos en formato: .java, .xml y yang. Para la creación de esta aplicación se tomó como base la aplicación de Learning-switch de SDN Hub.

En esta sección se explicarán dos archivos modificados que son: sharedbottleneck-mptcp-imp.yang y 60-sharedbottleneck-mptcp-config.xml.

El primer archivo, sharedbottleneck-mptcp-imp.yang, está desarrollado mediante YANG. Este archivo es importante al momento de compilar el proyecto ya que tomará información de todos los archivos pom.xml. Al compilar el proyecto se crearán dos archivos .java a partir del archivo sharedbottleneck-mptcp-imp.yang, en la dirección ../implementation/src/main/java/org/opendaylight/sharedbottleneckmptcp/impl /. Los archivos creados son: SharedBottleneckMptcpModule.java y SharedBottleneckMptcpModule Factory.java que se describen a continuación:

- SharedBottleneckMptcpModule.java: A este archivo se lo considera como un activador de la aplicación *shared bottleneck* MP-TCP. Además contiene servicios iniciales que se encuentran definidos en 60-sharedbottleneck-mptcp-config.xml, el cual se describirá más adelante; dichos servicios se encontrarán disponibles a través de MD-SAL. Existen varios métodos en este archivo .java que son utilizados en la aplicación, siendo uno de los más importantes createInstance, el cual pasa referencias a la aplicación desarrollada como argumentos en un constructor.
- SharedBottleneckMptcpModuleFactory.java: a este archivo java no se lo modifica. Dicho archivo contiene la clase que permite instanciar a SharedBottleneckMptcpModule.

En el archivo 60-sharedbottleneck-mptcp-config.xml, su nombre se encuentra iniciado por un número; en este caso se le asignó 60 considerando que los valores en el rango de 49 a 59 se encuentran asignados a diferentes *features* como DLUX, L2Switch, Openflow, entre otros, que son utilizados por la aplicación desarrollada. Además, esta

numeración especifica el orden en que se instalarán y ejecutarán los *features* en OpenDayLight.

En el archivo 60-sharedbottleneck-mptcp-config.xml se encuentran especificados los siguientes servicios MD-SAL que son utilizados en la aplicación desarrollada:

- NotificationProviderService: este servicio permite que la aplicación desarrollada se registre y escuche las notificaciones que se encuentran especificadas en el modelo Yang.
- DataBroker: este servicio es uno de los más importantes y utilizados en la aplicación ya que permite la escritura y lectura del *Data Store* de OpenDayLight.

La segunda sección de la estructura del proyecto SDN Hub se observa en la Figura 2.5 donde se muestran los archivos denominados Archivos modificados, cuyo color es rojo. En estos archivos solo se realizaron modificaciones para eliminar las aplicaciones del proyecto SDN Hub y agregar la aplicación desarrollada sharedbottleneck-mptcp.

La mayoría de los archivos modificados son pom.xml, pero existe un archivo adicional llamado features.xml que se lo describe a continuación.

En el archivo features.xml se encuentran definidos los *features* que necesita la aplicación para su funcionamiento; en este caso, los *features* son: openflow, md-sal, l2switch y dlux. Cada uno de estos *features* serán explicados y como se aplicaron en la aplicación a continuación.

2.4.1.1. *Features* utilizados en la aplicación

Para la aplicación desarrollada se utilizaron varios *features*, los cuales facilitaron su implementación. A continuación se describen las tareas en las que fueron utilizados cada uno de ellos:

Controller: Es uno de los *features* más importantes dentro del controlador OpenDayLight, ya que brinda un soporte para las aplicaciones *bundles* y *features* en cualquier módulo de la arquitectura del controlador.

OpenFlow: Es el encargado de utilizar y gestionar el protocolo OpenFlow entre el controlador OpenDayLight y los switches. Además, este *feature* tiene la función de guardar la información en MD-SAL que proporciona el protocolo OpenFlow como por ejemplo la versión o las reglas instaladas en los switches, entre otros.

L2Switch: Realiza la conmutación de paquetes; este *feature* es importante en la aplicación para el caso en que los paquetes no sean MP-TCP, ya que se utilizarán las reglas que el *feature* instala para la conmutación de capa 2.

HostTracker: Es el encargado de encontrar los *hosts* dentro de la infraestructura de red que se encuentra manejada por el controlador OpenDayLight. Además, asocia tanto la IP así como la MAC de cada *host*, con el nodo y puerto al que el *host* se encuentra conectado.

LoopRemover: Este *feature* trabaja de forma complementaria con L2Switch ya que elimina todos los posibles lazos que tiene la infraestructura de red. En la aplicación desarrollada este *feature* bloquea posibles rutas que podrían tomar los subflujos de MP-TCP para que no se creen lazos.

ArpHandler: Se encarga de decodificar los paquetes ARP para obtener las direcciones MAC y asociarlas con un puerto. En la aplicación se utiliza este *feature* para actualizar el servicio REST de *network topology* que se explicará más adelante.

DLUX: Este *feature* se utilizó para la administración de la infraestructura de red así como también del controlador OpenDayLight de forma gráfica e interactiva.

2.4.2. Diagrama de las clases de la aplicación

En la Figura 2.6 se pueden observar las clases desarrolladas para la aplicación, como se mencionó en la sección anterior, dos clases se generan automáticamente mediante el modelo Yang, y se crearon seis clases adicionales, las cuales tienen funciones específicas. Las tres primeras clases creadas son las que tienen relación con el Algoritmo de Dijkstra: Graph, Nodo y Dijkstra.

Se agregó la clase MensajesMP_TCP desarrollada en [1], la cual tiene como finalidad la decodificación de los paquetes MP-TCP. A dicha clase también se la modificó y agregó nuevos métodos para obtener parámetros necesarios del paquete MP-TCP para el desarrollo de la aplicación.

También se implementó la clase Rutas, la cual está encargada de almacenar las rutas creadas por el Algoritmo de Dijkstra. Finalmente, se creó la clase principal denominada SharedBottleneckMP_TCP; en esta clase interactúan las clases anteriormente mencionadas, además se crearon métodos específicos los cuales ayudan a la implementación de la aplicación.

Figura 2.6 Diagrama de clases de la aplicación

Esta clase se la tiene que instanciar en SharedBottleneckMptcpModule para que cuando se instale el *feature*, toda la funcionalidad de la aplicación desarrollada se implementó en el controlador OpenDayLight.

2.4.3. Codificación de la aplicación

Esta sección presenta la aplicación implementada y explica las clases desarrolladas y métodos usados para el desarrollo de la misma. Además se describirá el código más importante implementado en cada clase.

Los métodos más importantes de OpenDayLight que se utilizaron son: OnPacketReceived, OnNodeConnectorUpdated, OnNodeUpdated y OnIpv4PacketReceived.

De los métodos antes mencionados el que tiene más importancia en la aplicación es OnPacketReceived, ya que este método procesa todos los paquetes que son enviados desde un switch OpenFlow al controlador. Este método pertenece a la interfaz PacketProcessingService y tiene como parámetro de entrada un objeto de tipo PacketReceived, el cual contiene el *payload* de los paquetes que ingresan al conmutador OpenFlow. Además se utilizó el método OnNodeUpdated el cual se crea una notificación cuando se agrega o se actualiza una característica de un switch conectado al controlador OpenDayLight.

A parte de los métodos que tiene el controlador OpenDayLight se utilizaron métodos desarrollados en el proyecto de SDN Hub. De todos los métodos que fueron desarrollados por SDN Hub se utilizaron específicamente los que se describen a continuación:

- InventoryUtils: este método permite la decodificación de los paquetes que ingresan al conmutador, entre la información que se pueden obtener son la dirección IP tanto de origen como de destino, puertos de origen y destino, entre otros.
- GenericTransactionsUtils: es un conjunto de funciones que permiten realizar transacciones tanto de lectura así como de escritura en el *Data Store* de OpenDayLight.

En la siguiente sección se describirán las clases desarrolladas con los métodos más importantes implementados en cada una de ellas.

2.4.3.1. Desarrollo de las clases propias

En esta sección se han dividido las clases según la funcionalidad que cumplen cada una de ellas, por este motivo se las separo en tres partes: Algoritmo Dijkstra, rutas y aplicación principal.

2.4.3.1.1 Algoritmo de Dijkstra

El objetivo principal del algoritmo es determinar la trayectoria más corta entre un nodo inicial con el resto de nodos que conforman el grafo.

Para mantener actualizadas las rutas se utilizaron dos tipos de conjuntos de nodos: establecidos y no establecidos. Los nodos establecidos son aquellos que fueron evaluados, es decir que tienen una distancia o peso al nodo inicial.

Por otro lado los nodos no establecidos son aquellos que van a ser evaluados y a los cuales aún no se les ha asignado un peso al nodo inicial. A continuación, se enumeran los pasos a seguir para implementar el Algoritmo de Dijkstra:

El primer paso consiste en asignar el peso de 0 al nodo inicial, a los nodos restantes se los distinguirán por un nodo predecesor y un peso.

En este paso también se asignan a los nodos, excepto al inicial, un peso infinito y con un predecesor desconocido, debido a que los nodos aún no se encuentran procesados por el Algoritmo de Dijkstra. Los conjuntos de nodos establecidos como no establecidos se encuentran sin elementos al momento.

Como segundo paso se agrega el nodo inicial al conjunto de nodos no establecidos, en este conjunto solo existe un elemento y a este se lo evalúa.

Para evaluar un nodo se tiene que tomar en cuenta a los nodos que se encuentran directamente conectados a él, denominados nodos adjuntos; con la información de los nodos adjuntos se actualiza los valores de los nodos predecesores desconocidos por el nodo evaluado y el valor del enlace que se encuentra con infinito por el valor del enlace entre el nodo evaluado y el nodo adjunto.

Como tercer pasó se agregan a los nodos adyacentes del nodo inicial al conjunto de nodos no establecidos, el nodo inicial se retira de los nodos no establecidos y se lo agrega a los nodos establecidos ya que fue evaluado.

Del conjunto de nodos no establecidos se selecciona al nodo con menor peso y se procede a evaluarlo como en el paso anterior. Existe una condición adicional, si el peso disminuye se actualiza el valor como el nodo predecesor sino se mantiene el valor.

Como cuarto paso se realiza de forma iterativa el tercer paso hasta que todos los nodos se encuentran en el conjunto de nodos establecidos. Al realizar este último paso ya se puede calcular la ruta más corta desde el nodo inicial a un nodo en el grafo.

A la primera clase desarrollada se la llamo Nodo, ya que hace referencia a la unidad más pequeña dentro de un grafo. Esta clase contiene cuatro atributos de entrada como se puede observar en el Código 2.1.

```
private String nombre;
private LinkedList<Nodo> rutaCorta = new LinkedList<>();
private Integer peso = Integer.MAX_VALUE;
private Map<Nodo, Integer> nodosAdjuntos = new HashMap<>();
```

Código 2.1 Atributos de la clase Nodo

El primer atributo es nombre, de tipo String; en este parámetro se almacena el nombre del nodo dentro del grafo.

El segundo atributo es rutaCorta, de tipo LinkedList, esta lista es doblemente ligada con lo que se puede acceder a los componentes sucesores y predecesores de un elemento, para procesos de inserción y eliminación de los elementos de la lista sean en tiempos constantes mediante iteradores. La lista permite almacenar la ruta más corta a cada nodo.

El tercer atributo es peso, de tipo Integer, el cual se encuentra destinado para contener el peso de cada enlace que pertenece a un nodo. Por último, el atributo nodosAdjuntos, de tipo Map, que permite la presentación de una estructura de datos para almacenarlos en pares de clave/valor, en este parámetro se agregaran los nodos adjuntos.

Los siguientes métodos se implementaron en la clase Nodo: un constructor con un argumento de entrada para agregar el nombre al nodo.

Se desarrolló el método addDestino que tiene dos argumentos de entrada que son: destino de tipo Nodo donde se agrega al nodo que se desea alcanzar y peso de tipo int; estos parámetros se agregan a nodosAdjuntos de tipo Map.

Existen más métodos que son de tipo getter[47] y setter[48], para todos los atributos de la clase Nodo, un total de ocho métodos, la mitad son de tipo setter y la otra de tipo getter.

[47] Getter: este método permite recuperar o acceder a un atributo de una clase. Este tipo de métodos obligatoriamente retorna un valor.

La siguiente clase implementada fue Graph, que alberga la información relacionada con el grafo. Esta clase solo tiene un atributo de entrada de tipo Set, ya que se va a manejar un conjunto de elementos que en este caso son objetos de tipo Nodo.

En esta clase también se utilizan los métodos getter y setter para su único atributo, además, dispone de un método llamado addNodo que tiene como argumento de entrada un objeto de tipo Nodo, el cual se agregará al conjunto de nodos.

En el Código 2.2 se presenta la clase Graph antes descrita.

```
private Set<Nodo> nodos = new HashSet<>();
//metodo para agregar nodos
public void addNodo(Nodo nodoA) {
    nodos.add(nodoA);
}
//metodo para retornar un nodo
public Set<Nodo> getNodos() {
    return nodos;
}
// metodo para cambiar un nodo
public void setNodos(Set<Nodo> nodes) {
    this.nodos = nodes;
}
```

Código 2.2 Clase Graph

Finalmente, a la clase principal del Algoritmo de Dijkstra se le asignó el nombre de Dijkstra, esta clase no tiene atributos, solo contiene métodos que se describen a continuación, su implementación se puede observar en las secciones de código: Código 2.3, Código 2.4 y Código 2.5.

El primer método desarrollado fue calculoRutaCortaOrigen (Código 2.3), el cual se encarga de encontrar la ruta más corta a un nodo determinado.

Dentro de esta clase se tienen dos tipos de conjuntos de nodos, los establecidos y no establecidos que son variables de tipo Set<Nodo> y tienen asignados los siguientes nombres: nodosEstablecidos y nodosNoEstablecidos; dentro de estos conjuntos se encuentran objetos de tipo Nodo.

El método calculoRutaCortaOrigen tiene dos argumentos de entrada: a) grafo que es un objeto de tipo Graph, y en el que se encuentra almacenado el grafo para ser evaluado con todos sus nodos; y, b) el argumento inicial que es un objeto de tipo Nodo donde se indica el nodo inicial para realizar la evaluación del Algoritmo de Dijkstra.

[48] Setter: es un método de tipo void que permite asignar un valor a un atributo.

El nodo inicial se agrega al conjunto de nodos no establecidos y se procede a implementar el Algoritmo de Dijkstra. Al nodo inicial se asigna un peso igual a 0 mediante el método setPeso, luego se establecen dos arreglos nodosEstablecidos y nodosNoEstablecidos donde se agrega al nodo inicial, después ingresa a un lazo donde se verifica que nodosNoEstablecidos se encuentre con nodos, si se encuentra vacío se termina el Algoritmo de Dijkstra ya que recorrió todos los nodos del grafo obteniendo las rutas.

Si el arreglo nodosNoEstablecidos tiene elementos ingresa al lazo donde se obtiene el nodo con menor peso mediante el método getNodoMenorPeso de los nodosNoEstablecidos y se guarda el nodo obtenido en nodoActual el cual es removido de nodosNoEstablecidos.

Posteriormente se tiene que evaluar los nodos adjuntos del nodoActual mediante un lazo para recorrer todos elementos de los nodosEstablecidos y utilizar el método CalculoPesoMinimo para agregarlos si es nodo adjunto en nodosNoEstablecidos. Para recorrer todo el grafo es necesario que el nodo que fue evaluado que en este caso es nodoActual pase a nodosEstablecidos y se repita hasta que no exista elementos en nodosNoEstablecidos.

En el método se emplean dos funciones que son: a) calculoPesoMinimo que tiene tres argumentos de entrada, que permiten comparar los pesos de los enlaces con los que se calculan al explorar una ruta; en el Código 2.4 se puede observar lo antes mencionado; y**, b)** getNodoMenorPeso es un método que tiene un argumento de entrada, este método retorna el nodo con la distancia o peso más pequeño del conjunto de nodos sin establecer. En el Código 2.3 se puede observar todo el método calculoRutaCortaOrigen antes descrito.

```
public static Graph calculoRutaCortaOrigen(Graph grafo, Nodo inicial) {
    inicial.setPeso(0);//al nodo inicial se agrega con el peso de 0
    Set<Nodo> nodosEstablecidos = new HashSet<>();//creacion de nodosEstablecidos
    Set<Nodo> nodosNoEstablecidos = new HashSet<>();//creacion de nodosNoEstablecidos
    nodosNoEstablecidos.add(inicial);//el nodo inicial es agregado a nodosNoEstablecidos
    while (nodosNoEstablecidos.size() != 0) {//verificacion que los nodosNoEstablecidos no sea 0
        Nodo nodoActual = getNodoMenorPeso(nodosNoEstablecidos);//se obtiene nodo con menor peso
        nodosNoEstablecidos.remove(nodoActual);//se quita al nodo que se evalua de nodosNoEstablecidos
        for (Entry<Nodo, Integer> parAdyacentes : nodoActual.getNodosAdjuntos().entrySet()) {
            Nodo nodoAdyacente = parAdyacentes.getKey();
            Integer pesoBorde = parAdyacentes.getValue();
            // se evalua el nodo actual con los nodos establecidos mediante los nodos adyacentes
            if (!nodosEstablecidos.contains(nodoAdyacente)) {
                CalculoPesoMinimo(nodoAdyacente, pesoBorde, nodoActual);//se calcula los pesos minimos
                nodosNoEstablecidos.add(nodoAdyacente);//se agrega a nodosNoEstablecidos  los nodoAdyacente
            }
        }
        nodosEstablecidos.add(nodoActual);// se agrega el nodo evaluado a nodosEstablecidos
    }
    return grafo;// se retorna el elemento grafo
}
```

Código 2.3 Método calculoRutaCortaOrigen

```
private static void CalculoPesoMinimo(Nodo nodoEvaluado, Integer pesoBorde, Nodo nodoOrigen) {
    Integer pesoOrigen = nodoOrigen.getPeso();//Se obtiene el peso del nodo origen
    //Se agrega el pesoOrigen mas el pesoBorde con el peso del nodoEvaluado
    if (pesoOrigen + pesoBorde < nodoEvaluado.getPeso()) {
        nodoEvaluado.setPeso(pesoOrigen + pesoBorde);//si el peso es menor se agrega al nodoEvaluado
        LinkedList<Nodo> rutaCorta = new LinkedList<>(nodoOrigen.getRutaCorta());
        //Se crea la rutaCorta y se agrega el nodoOrigen con la ruta corta
        rutaCorta.add(nodoOrigen);//Se agrega a rutaCorta el nodoOrigen
        nodoEvaluado.setRutaCorta(rutaCorta);//se agrega la rutaCorta la nueva ruta calculada
    }
}
```

Código 2.4 Método calculoPesoMinimo

```
private static Nodo getNodoMenorPeso(Set<Nodo> nodosNoEstablecidos) {
    Nodo nodoMenorPeso = null;//Se crea un nodo nulo
    int menorPeso = Integer.MAX_VALUE;// se crea un menorPeso con el maximo valor
    //Se compara un nodo dentro del conjunto de nodosNoEstablecidos
    for (Nodo nodo : nodosNoEstablecidos) {
        int nodoDistancia = nodo.getPeso();
        //Se compara el peso del nodo con nodoDistancia
        if (nodoDistancia < menorPeso) {
            menorPeso = nodoDistancia;//se agrega un nodo a menor peso
            nodoMenorPeso = nodo;// el nodo con menor peso se agrega a nodoMenorPeso
        }
    }
    return nodoMenorPeso;//Se retorna el nodo con menor peso
```

Código 2.5 Método getNodoMenorPeso

2.4.3.1.2 Rutas

La clase Rutas se utilizó para almacenar todos los caminos o rutas que provee el Algoritmo de Dijkstra, además se combinó con la información del protocolo MP-TCP. La clase tiene cuatro atributos que se pueden observar en el Código 2.6.

```
public ArrayList<String> rutascortas=new ArrayList<>();
public String hash;
public int contador;
public int contadorJOIN;
```

Código 2.6 Atributos de la clase Rutas

La clase Rutas contiene un constructor, en el que se inicializan varios atributos, además incluye métodos de tipo getter y setter, tal como se puede observar en el Código 2.7.

```
public Rutas(ArrayList<String> rutascortas,String hash,int contador, int contadorJOIN) {
    //Este constructor tiene todos los argumentos de la clase Rutas
    this.rutascortas= rutascortas;
    this.hash=hash;
    this.contador = contador;
    this.contadorJOIN= contadorJOIN;
}
public ArrayList<String> getrutascortas(){//metodo getter para rutas cortas
    return this.rutascortas;
}
public String gethash() {//metodo getter para obtener el hash
    return this.hash;
}
public int getconador() {//metodo getter para obtener el contador
    return this.contador;
}
public void sethash(String j) {// metodo setter para agregar hash a la ruta
    this.hash=j;
}
public void setcontador(int i) {//metodo setter para agregar contador a la ruta
    this.contador=i;
}
public int getcontadorJOIN() {//metodo getter para obtener el contadorJOIN
    return this.contadorJOIN;
}
public void setcontadorJOIN(int j) {//metodo para agregar el contadorJOIN
    this.contadorJOIN=j;
}
```

Código 2.7 Métodos y atributos de la clase Rutas

2.4.3.1.3 Aplicación Principal

La clase principal de la aplicación desarrollada se denomina SharedBottleneckMP-TCP, en esta clase se incluye la funcionalidad de la aplicación así como también la interacción con el resto de clases que ya fueron descritas.

La clase SharedBottleneckMP-TCP utiliza el constructor que se encuentra en el Código 2.8, donde se puede observar tres argumentos de entrada: a) dataBroker, este parámetro permite la escritura y lectura del *Data Store*; y b) notificationService, este objeto permite escuchar las notificaciones del controlador.

```
public SharedBottleneckMP-TCP(final DataBroker dataBroker, NotificationProviderService notificationService,
        RpcProviderRegistry rpcProviderRegistry) {
    // Almacenar el data broker para leer/escribir en en inventory store
    this.dataBroker = dataBroker;
    // Obtener acceso al servicio de procesamiento de paquetes para realizar llamadas RPC
    this.packetProcessingService = rpcProviderRegistry.getRpcService(PacketProcessingService.class);
    // Lista utilizada para rastrear la notificación (tanto el cambio de datos como el definido por YANG)
    this.registrations = Lists.newArrayList();
    // Registro de objeto para recibir notificaciones cuando exista
    registrations.add(notificationService.registerNotificationListener(this));
```

Código 2.8 Constructor SharedBottleneckMP-TCP

El método onPacketReceived se puede observar en el Código 2.9, el cual tiene como argumento de entrada un objeto notification que es de tipo PacketReceived. Este método se invoca solo cuando un paquete es enviado desde el switch al controlador OpenDayLight. Lo que a continuación se presenta se encuentra dentro del método onPacketReceived.

```
@Override
public void onPacketReceived(PacketReceived notification) {
    //Metodo que se activa cuando un paquete es enviado al controlador
```

Código 2.9 Método onPacketReceived

2.4.3.1.3.1 Método onPacketReceived

En el Código 2.10 se observa la decodificación de un paquete TCP mediante el método getPayload, de la clase PacketReceived, se obtiene el *payload* del paquete TCP, el cual se encuentra en bytes y se guarda en la variable Paquete que es un arreglo de bytes.

El *payload* se almacena en un arreglo de bytes con el nombre de PROTOCOLCAPA4RAW.

Para extraer el tipo de protocolo de capa 4 del paquete se utiliza el método extraerProtoType de la clase PacketParsingUtils, el cual tiene como argumento de entrada un arreglo de bytes, que en este caso será la variable Paquete. Para poder

obtener el tipo de protocolo en formato String se emplea el método rawProtocolToString de la clase PacketParsingUtils, esta información se almacena en ID_Protocolo_Capa_4.

Para la aplicación solo se utilizan los paquetes TCP, por este motivo se tiene que comparar la variable ID_Protocolo_Capa_4 con el valor de "6" que es el asignado para el protocolo TCP.

Para la conmutación del resto de paquetes que no son TCP se utilizan las reglas que instala el *feature* L2Switch.

```
Paquete = notification.getPayload();
    try {
        byte[] PROTOCOLCAPA4Raw = PacketParsingUtils.extraerProtoType(Paquete);
        //se obtiene el protocolo de capa 3
        final String ID_Protocolo_Capa_4 = PacketParsingUtils.rawProtocolToString(PROTOCOLCAPA4Raw);
        // se obtiene el protocolo de capa 4
        if(Integer.parseInt(ID_Protocolo_Capa_4)==6){
            //se verifica que el paquete pertenece a TCP
            NodeConnectorRef ingressNodeConnectorRef = notification.getIngress();
            //se obtiene el identificador del conector
            final NodeId path_id_switch = InventoryUtils.getNodeId(ingressNodeConnectorRef);
            //se obtien el id del nodo del paquete que ingresa
            byte[] MacFuenteRawAnalizador = PacketParsingUtils.extraerSrcMac(Paquete);
            //se obtiene la MAC en bytes de la fuente
            final String MacFuenteAnalizador = PacketParsingUtils.rawMacToString(MacFuenteRawAnalizador);
            //se tansforma la MAC de la fuente de bytes a string
            byte[] MacDestinoRawAnalizador = PacketParsingUtils.extraerDstMac(Paquete);
            //se obtiene la MAC en bytes del destino
            final String MacDestinoAnalizador = PacketParsingUtils.rawMacToString(MacDestinoRawAnalizador);
            //se tansforma la MAC de la fuente de bytes a string
            byte[] VersionAndLongitudCabeceraIPRaw = PacketParsingUtils.extraerVersionAndLongCabeceraIP(
            Paquete);
            //Se obtiene la longitud de lacabecera en bytes
            byte LongitudCabeceraIP = VersionAndLongitudCabeceraIPRaw[0];
            //se obtiene la version de la longitud
            String LongitudCabeceraIPString = Integer.toHexString(LongitudCabeceraIP & 0xf);
            //se obtien la longitud IP de tipo String
            int Longitud_Cabecera_IP_Entero = 0;
            Longitud_Cabecera_IP_Entero = Integer.parseInt(LongitudCabeceraIPString);
            //Se transforma a entero la longitud IP
```

Código 2.10 Decodificación del paquete TCP

Una vez realizada la comparación de los paquetes TCP se extrae toda la información de los mismos. En el Código 2.10 se puede observar el empleo de dos clases para la extracción de la información de los paquetes TCP, estas clases son: InventoryUtils, la cual se encarga de extraer información sobre los switches y PacketParsingUtils, con esta clase se obtiene información sobre la conformación del paquete como su dirección MAC de origen y destino, longitud de la cabecera IP entre otros elementos.

En el Código 2.11 se presenta el filtro que se realizó para determinar si un paquete utiliza el protocolo MP-TCP, para realizar esto se utiliza un lazo donde se analiza todos los identificadores del mensaje en el campo de opciones de TCP llamado Campo_Opciones_TCP_Raw.

La variable Puntero_Mensajes_Campo _Opciones_TCP se incrementa dependiendo del campo de longitud del mensaje con lo que se determina el identificador de tipo el cual se compara con el valor de 30 para saber si efectivamente es un mensaje MP-TCP.

```
for (Puntero_Mensajes_Campo_Opciones_TCP = 0; Puntero_Mensajes_Campo_Opciones_TCP < Campo_Opciones_TCP_Raw
.length; Puntero_Mensajes_Campo_Opciones_TCP++) {
    //lazo en el que se verifica el campo de opciones se encuentra en 0
    if ((Campo_Opciones_TCP_Raw[Puntero_Mensajes_Campo_Opciones_TCP] != 1)
            && (Campo_Opciones_TCP_Raw[Puntero_Mensajes_Campo_Opciones_TCP] != 30)) {
        //se verifica que el campo de opciones sea diferente de uno y de 30
        Puntero_Mensajes_Campo_Opciones_TCP = Puntero_Mensajes_Campo_Opciones_TCP
                + Campo_Opciones_TCP_Raw[Puntero_Mensajes_Campo_Opciones_TCP + 1] - 1;
        //se almacena en Puntero_Mensajes_Campo_Opciones_TCP el puntero obtenido
    } else if (Campo_Opciones_TCP_Raw[Puntero_Mensajes_Campo_Opciones_TCP] == 30) {
        //si el campo de opciones es igual a 30 se sale del lazo
        break;
    }
}
```

Código 2.11 Filtro para mensajes MP-TCP

En el Código 2.12 se presenta la forma como se extrae tanto la dirección IP de origen como la de destino del paquete que ingreso al controlador. En primera instancia las direcciones IP que se obtienen son un arreglo de bytes los cuales tienen que ser transformados a tipo String para poder manejarlos. Las direcciones IP son importantes para la creación de las reglas de los 5 elementos de una tupla que se describirán más adelante. Además, se crea un objeto de la clase MensajesMP_TCP para la decodificación de los paquetes MP-TCP.

```
byte[] ipFuenteRaw = PacketParsingUtils.extraerIpSrc(Paquete);
//se obtiene la ip de la fuente en bytes
String ipFuenteAnalizador = PacketParsingUtils.rawIpToString(ipFuenteRaw);
//se convierte la ip de fuente a string
final MensajesMP_TCP mensajes = new MensajesMP_TCP();
//se crea un objeto de la clase MensajesMP_TCP
byte[] ipDestinoRaw = PacketParsingUtils.extraerIpDest(Paquete);
//se obtiene en bytes la IP de destino
String ipDestinoAnalizador = PacketParsingUtils.rawIpToString(ipDestinoRaw);
//se convierte la ip de destino a string
```

Código 2.12 Obtención de direcciones IP de origen y de destino

Una vez que la aplicación dispone solamente de mensajes tipo MP-TCP se realiza un nuevo filtro para poder diferenciar los tipos de mensajes que tiene el protocolo MP-TCP. Entre todos los tipos de mensajes MP-TCP de la aplicación solo se utilizarán dos tipos, estos son: MP_CAPABLE y MP_JOIN para mayor información sobre estos mensajes se encuentra en el capítulo anterior. Para poder filtrar los tipos de mensajes MP-TCP se utiliza el objeto mensajes con el método iniciar el cual tiene un conjunto de parámetros de entrada que son: el campo de opciones TCP de tipo bytes, el puntero del campo de opciones, el *payload* del mensaje, la longitud de la cabecera del mensaje, la dirección MAC tanto de origen como de destino, IP de origen y destino, y por último, el nodo que está transmitiendo la información.

En el Código 2.13 se observan los pasos para filtrar los mensajes MP-TCP de tipo MP_CAPABLE por medio de la variable subtipodemensajeMPTCP la cual debe tener el valor de “0”; también, se verifica que los mensajes que ingresan sean de tipo SYN, esto

se lo realiza comparando el valor obtenido en getLongitudCampoMPCapable() con el valor 12, porque son los primeros que se envían al establecer la comunicación de tres vías. Además en el Código 2.13 se pueden observar los pasos para la creación de la matriz que tiene cuatro columnas: a) en la primera columna se almacena el identificador del nodo; b) a la segunda columna se le asigna al valor numérico del nodo que le asigna OpenDayLight; c) en la tercera columna se almacena al identificador del nodo al que se encuentra conectado el nodo identificado en la primera columna; y, d) en la última columna se fija el valor numérico del identificador del nodo de la tercera columna.

```
if (subtipodemensajeMPTCP == 0 && mensajes.getLongitudCampoMP_Capable()==12) {
//Se verifica que el tipo de mensaje MPTCP es Capable y de este mensjae el primero
            String ip, linkhostswitch;//Se crea dos elementos IP y linkhostswitch
            String puertoswitch1, puertoswitch2;//Se crea puertoswitch2 puertoswitch1
            int numeronodos = 0, numeronodo1=0, numeronodo2=0;
            List<List<String>> switches = new ArrayList<List<String>>();
            //Se crea una matriz de switches donde se agrega cuatro columnas
            switches.add(new ArrayList<String>());
            //Se agrega el nombre completo del nodo
            switches.add(new ArrayList<String>());
            //Se agrega el path ID del nodo
            switches.add(new ArrayList<String>());
            //Se agrega el nombre completo del nodo al que se encuentra conectado
            switches.add(new ArrayList<String>());
            //Se agrega el path ID del nodo al que se encuentra conectado
```

Código 2.13 Filtro para de mensajes MP_CAPABLE

Para obtener la topología de la infraestructura de red con los enlaces, switches y *hosts* que se encuentran conectados entre sí, es necesario el manejo del servicio REST network-topology, el cual se encuentra en http://ipcontrolador:8181/restconf/operational/network-topology:network-topology. El servicio network-topology brinda dos tipos de respuestas: a) esta opción se puede observar en la Figura 2.7 y tiene un formato XML; y, b) esta opción se presenta en Figura 2.8, la cual es una respuesta en formato JSON[49].

```
<node>
   <node-id>host:00:00:00:00:00:01</node-id>
   <id>00:00:00:00:00:01</id>
   <addresses>
      <id>1</id>
      <mac>00:00:00:00:00:01</mac>
      <ip>10.0.0.1</ip>
      <first-seen>1490038075913</first-seen>
      <last-seen>1490265831658</last-seen>
   </addresses>
   <attachment-points>
      <tp-id>openflow:1:1</tp-id>
      <active>true</active>
      <corresponding-tp>host:00:00:00:00:00:01</corresponding-tp>
   </attachment-points>
   <termination-point>
      <tp-id>host:00:00:00:00:00:01</tp-id>
   </termination-point>
</node>
```

Figura 2.7 Ejemplo de la respuesta en XML del servicio network-topology

[49] JSON: JavaScript *Object Notation*, es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript o XML.

Figura 2.8 Ejemplo de la respuesta en JSON del servicio network-topology

Para el caso de la aplicación desarrollada se utilizó la respuesta de tipo JSON; en el Código 2.14 se presenta el proceso necesario para realizar consultas al servicio network-topology, que se encuentra en el controlador. Para obtener información del servicio REST network-topology se debe establecer como método de solicitud GET, aparte de esto también es necesario la autenticación del usuario ya que sin esto no se puede obtener la información del servicio. Para establecer el tipo de respuesta JSON del servicio REST, se tiene que utilizar el método setRequestProperty de la clase HttpURLConnection con los siguientes parámetros Accept y application/json, mediante el método getInputStream() obtiene el flujo de entrada del servicio network-topology.

```
String url = "http://127.0.0.1:8181/restconf/operational/network-topology:network-topology";
//Direccion donde se encuentra alojado el servicio network-topology
        URL obj = new URL(url);//se crea el objeto de tipo URL con la direccion del servicio
        HttpURLConnection conn = (HttpURLConnection) obj.openConnection();
        //Se crea el objeto HttpURLConnection para poder abrir la conexion al servicio
        String userPassword = "admin:admin";//Se crea el usuario y contraseña del servicio
        conn.setRequestMethod("GET");//Se especifica el tipo de metodo para el servicio
        String authEncBytes = org.apache.commons.codec.binary.Base64
                .encodeBase64String(userPassword.getBytes());
        //Se crea authEncBytes codificada para un string base 64 donde se agrega a userPassword
        conn.setRequestProperty("Authorization", "Basic " + authEncBytes);
        // se agrega al objeto conn el usuario y password para autentificacion
        conn.setRequestProperty("Accept", "application/json");
        //se agrega el tipo de respuesta que se desea en este caso json
        InputStream content = conn.getInputStream();
        //Se alamacena el stream para almacenarlo en content
```

Código 2.14 Conexión al servicio REST de network-topology

En el Código 2.15 se observa el procedimiento para almacenar objetos de tipo JSON, para esto se almacenará en el objeto jsonTopology, el procedimiento para obtener dicha información se presenta en el Código 2.14. Al tener el objeto JSON, lo primero que se realizará es obtener las IP tanto del *host* receptor como transmisor, esta información se la guarda en una matriz donde se asocia tanto la dirección IP con la interfaz del switch que se conecta a cada *host*. Los identificadores de las interfaces de switch tienen la siguiente estructura openflow:x:y, que se describe a continuación:

- openflow: el primer identificador se encuentra en todos los switches que tienen soporte del protocolo OpenFlow.
- x: este identificador es utilizado para emparejar el controlador mediante un número con el switch se denomina como *Data Path ID*.
- y: el último identificador es el número del puerto al que se encuentra conectado.

```
org.json.JSONObject jsonTopology = new org.json.JSONObject(result.toString());
//Se agrega el resultado que se obtiene desde el servicio de JSON de Network topology
for (int i = 0; i < jsonTopology.getJSONObject("network-topology").getJSONArray("topology").getJSONObject(0)
                        .getJSONArray("node").length(); i++) {
//Dentro del objeto jsonTopology se busca en topology todos los nodos en node
    if (jsonTopology.getJSONObject("network-topology").getJSONArray("topology").getJSONObject(0)
                            .getJSONArray("node").getJSONObject(i).get("node-id").toString().indexOf("host") == 0) {
    //En este condicional lo que se intenta es buscar los host e ip para asociarlos
        for (int j = 0; j < jsonTopology.getJSONObject("network-topology").getJSONArray("topology")
                                .getJSONObject(0).getJSONArray("node").getJSONObject(i)
                                .getJSONArray("host-tracker-service:addresses").length(); j++) {
        //en el servicio de host tracker se tiene que buscar a los host que se encuentran conectados
            ip = jsonTopology.getJSONObject("network-topology").getJSONArray("topology").getJSONObject(0)
                                    .getJSONArray("node").getJSONObject(i).getJSONArray(
                                    "host-tracker-service:addresses").getJSONObject(j).get("ip").toString();
            // se obtiene la ip como los host y se los guardan en la variable ip
            linkhostswitch = jsonTopology.getJSONObject("network-topology").getJSONArray("topology")
                                    .getJSONObject(0).getJSONArray("node").getJSONObject(i)
                                    .getJSONArray("host-tracker-service:attachment-points").getJSONObject(j).get(
                                    "tp-id").toString();
            //se obtiene a que puerto se encuentra conectado el host al nodo
            hostpip.put(ip, linkhostswitch);
            //Se agrega a la matriz hostpip la ip y los host que se encuentran en network topology
        }
    }
}
```

Código 2.15 Obtención de direcciones IP, host y nodos de la topología de red

En la aplicación se utiliza la matriz denominada switches, la misma se encuentra conformada por cuatro columnas las cuales ya fueron descritas con anterioridad, para almacenar la información que se obtiene del objeto.

En el Código 2.16 se puede observar el proceso de almacenamiento.

```
ipinicio= ipFuenteAnalizador;
//se almacena la ip fuente en ipinicio
ipfinal = ipDestinoAnalizador;
//se almacena la ip de destino en ipfinal
nodoinicio = Integer.parseInt(hostpip.get(ipinicio).split(":")[1]);
//Se obtiene el numero del identificador del pathID del nodo de inicio
nodofinal = Integer.parseInt(hostpip.get(ipfinal).split(":")[1]);
//Se obtiene el numero del identificador del pathID del nodo de final
for (int i = 0; i < jsonTopology.getJSONObject("network-topology").getJSONArray("topology").getJSONObject(0)
                        .getJSONArray("link").length(); i++) {
    //En este momento se desplaza por todos los enlaces que pertenecen a los nodos que se van agregando
    if (jsonTopology.getJSONObject("network-topology").getJSONArray("topology").getJSONObject(0)
                            .getJSONArray("link").getJSONObject(i).get("link-id").toString().indexOf("host") == -1)
                            {
        //Se verifica que los enlaces que se obtengan tengan el valor de -1 ya que estos son los que se utiliza
        puertoswitch1 = jsonTopology.getJSONObject("network-topology").getJSONArray("topology").getJSONObject(0)
                                .getJSONArray("link").getJSONObject(i).getJSONObject("source").get("source-tp").
                                toString();
        //Se obtiene los puertos de los switches y se los almacena en puertoswitch1
        puertoswitch2 = jsonTopology.getJSONObject("network-topology").getJSONArray("topology").getJSONObject(0)
                                .getJSONArray("link").getJSONObject(i).getJSONObject("destination").get("dest-tp")
                                .toString();
        //Se obtiene los puertos de los switches directamente conectados y se los almacena en puertoswitch2
        switches.get(0).add(puertoswitch1);
        //Se agrega la etiqueta completa de los puertos del switches de inicio
        switches.get(1).add(puertoswitch2);
        //Se agrega la etiqueta completa de los puertos del switches de final
        switches.get(2).add(puertoswitch1.split(":")[1]);
        //Se agrega el path id del switches de inicio
        switches.get(3).add(puertoswitch2.split(":")[1]);
        //Se agrega el path id del switches de final
    }
}
```

Código 2.16 Creación de la matriz con los nodos

Una vez que se tiene disponible la matriz con la información necesaria, se tiene que aplicar el Algoritmo de Dijkstra. Pero al aplicar el algoritmo se encontraron errores ya que la solución propuesta en [5] tenía limitantes en ciertas topologías de red que se describen a continuación.

En la Figura 2.9 se pueden observar dos topologías, en el diagrama (a), el algoritmo propuesto funcionaba correctamente ya que encontraba los tres caminos disponibles, pero en el diagrama (b), al aplicar el algoritmo, se encontró un solo camino, eliminando los dos restantes, esto sucede porque Dijkstra al encontrar la primera ruta ya no se vuelve a ocupar los switches que ya fueron utilizados dejando incomunicados a los switches restantes.

Para resolver el problema suscitado, y para que el algoritmo funcione con cualquier tipo de topología, lo que se implementó es la eliminación de nodos intermedios que se encuentren antes (sw7) y después (sw6) de que se produzca el *shared bottleneck* compartido.

Para que no se afecte la construcción de las rutas, los nodos que fueron eliminados son almacenados en un arreglo para utilizarlos después que se aplique el Algoritmo de Dijkstra y sean agregados a las rutas generadas, para obtener una ruta completa desde el *host* transmisor al *host* receptor.

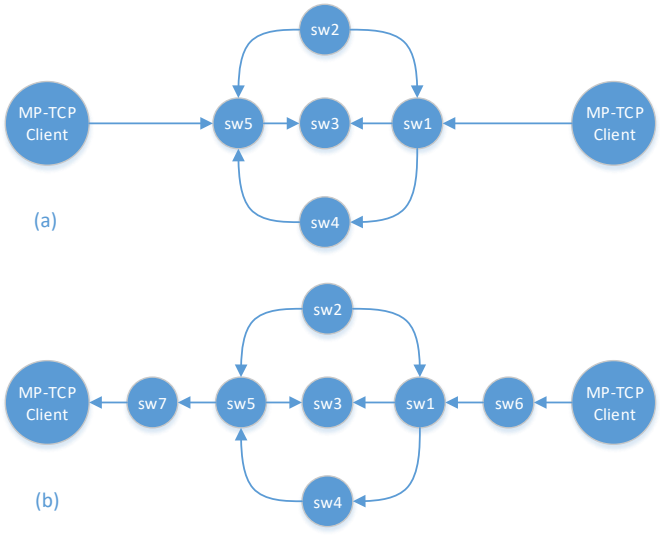

Figura 2.9 Diferentes tipos de topologías de red con cuellos de botella

En el Código 2.17 se puede observar el desarrollo necesario para eliminar los nodos a los que se les denominaron de inicio, estos nodos son los que se encuentran antes del cuello de botella compartido. La matriz que se utiliza es switches2 y la información de los nodos intermedios se almacena en variables para luego poder agregarlas a las rutas finales.

```
while (flag == 0) {
//con la bandera flag se verifica si existe o no nodos intermedios
    flagderutas = 0;
    //se crea una variable intermedia
    for (int i = 0; switches.get(0).size() > i; i++) {
    // se va a recorrer todos los elementos guardados en la matriz switches
        if (Integer.parseInt(switches.get(2).get(i)) == nodoiniciointer) {
        // se busca que exista coincidencia del data path con nodoiniciointer
            flagderutas++;//se agrega en una unidad a flagderutas
            if (flagderutas == 1) {//si es el primer elemento ingresa al condicional
                variableintermedia = i;//se agrega el valor de i a variableintermedia
            }
        }
    }
    if (contadord >= 1) {//Si el contadord es mayor o igual a una ingresa al condicional
        flagderutas = flagderutas - 1;//se resta en una unidad a flagderutas
    }
    if (flagderutas != 1) {//si flagderutas tiene el valor diferente a uno ingresa
        flag = 1;//se agrega el valor de una a flag
    } else {
        switches2.get(0).add(switches.get(0).get(variableintermedia));
        //se agrega a switches2 los nodos intermedios
        switches2.get(0).add(switches.get(1).get(variableintermedia));
        //se agrega a switches2 los nodos directamente conectados
        nodoiniciointer = Integer.parseInt(switches.get(3).get(variableintermedia));
        // se agrega a nodoiniciointer el valor del nodo inicial intermedio
        nodoinicio = nodoiniciointer;// se cambia el nodoinicio por nodoiniciointer
        nodosinter = "@" + nodoiniciointer + nodosinter;//se crea el nuevo token para la variable
        nodosintercompletosinicio = "@" + switches.get(1).get(variableintermedia) + "@"
        + switches.get(0).get(variableintermedia) + nodosintercompletosinicio;
        //se crea una nueva variable nodosintercompletosinicio para tener conexion extremo a extremo
    }
    contadord++;
}
```

Código 2.17 Encontrar nodos intermedios al inicio del shared bottleneck

En el Código 2.18 se puede observar el proceso para eliminar los nodos ubicados en la parte final del cuello de botella compartido; de igual manera se utiliza el arreglo llamado switches2 para obtener la información. Además, los nodos finales se almacenan en las variables nodofinal y nodosintermediofinal.

```
nodoiniciointer = nodofinal;//Se crea a nodo inicio a nodo final
contadord = 0;//se agrega el valor de 0 contadord
flag = 0;// se agrega el valor de 0 a flag
String nodosintermediofinal = "";//se inicializa en nulo el valor de nodosintermediofinal
while (flag == 0) {
//con la bandera flag se verifica si existe o no nodos intermedios
    flagderutas = 0;
    //se crea una variable intermedia
    for (int i = 0; switches.get(0).size() > i; i++) {
    // se va a recorrer todos los elementos guardados en la matriz switches
        if (Integer.parseInt(switches.get(2).get(i)) == nodoiniciointer) {
            // se busca que exista coincidencia del data path con nodoiniciointer
            flagderutas++;//se agrega en una unidad a flagderutas
            if (flagderutas == 1) {//si es el primer elemento ingresa al condicional
                variableintermedia = i;//se agrega el valor de i a variableintermedia
            }
        }
    }
    if (contadord >= 1) {//Si el contadord es mayor o igual a una ingresa al condicional
        flagderutas = flagderutas - 1;//se resta en una unidad a flagderutas
    }
    if (flagderutas != 1) {//si flagderutas tiene el valor diferente a uno ingresa
        flag = 1;//se agrega el valor de una a flag
    } else {
        switches2.get(0).add(switches.get(0).get(variableintermedia));
        //se agrega a switches2 los nodos intermedios
        switches2.get(0).add(switches.get(1).get(variableintermedia));
        //se agrega a switches2 los nodos directamente conectados
        nodoiniciointer = Integer.parseInt(switches.get(3).get(variableintermedia));
        // se agrega a nodoiniciointer el valor del nodo final intermedio
        nodofinal = nodoiniciointer;// se cambia el nodofinal por nodoiniciointer
        nodosintermediofinal = "@" + switches.get(0).get(variableintermedia) + "@"
                + switches.get(1).get(variableintermedia) + nodosintermediofinal;
        //se crea una nueva variable nodosintercompletosfinal para tener conexion extremo a extremo
    }
    contadord++;
}
```

Código 2.18 Encontrar nodos intermedios al final del cuello de botella

Para aplicar el Algoritmo de Dijkstra se tiene que asignar un peso o distancia administrativa a cada enlace que se encuentre asociado a cada nodo, para realizar esto se plantea el uso del servicio REST opendaylight-inventory. Este servicio REST se puede acceder a través de la URL http:// ipcontrolador:8181/restconf/ operational/opendaylight-inventory. A la respuesta JSON que se obtiene se la procesa de igual manera que se lo hizo para el servicio REST network-topology. En el Código 2.19 se puede observar el procedimiento que se sigue para obtener la velocidad de cada enlace de un nodo determinado el cual es utilizado para asignar el peso a cada arista del grafo a resolver.

```
String url1 = "http://127.0.0.1:8181/restconf/operational/opendaylight-inventory:nodes/node/"
        + switches.get(1).get(i).split(":")[0] + ":"
        + switches.get(1).get(i).split(":")[1] + "/node-connector/"
        + switches.get(1).get(i);
//Se agrega la direccion donde se encuentra alojado el servicio de inventory
URL obj1 = new URL(url1);//Se crea el objeto de tipo obj1 del servicio inventory
HttpURLConnection conn1 = (HttpURLConnection) obj1.openConnection();
//Se crea el objeto conn1 con los parametros del servicio inventory
String userPassword1 = "admin:admin";//Se crea el usuario y contraseña del servicio
conn1.setRequestMethod("GET");//Se especifica el tipo de metodo para el servicio
String authEncBytes1 = org.apache.commons.codec.binary.Base64
        .encodeBase64String(userPassword1.getBytes());
//Se crea authEncBytes codificada para un string base 64 donde se agrega a userPassword
conn1.setRequestProperty("Authorization", "Basic " + authEncBytes1);
// se agrega al objeto conn el usuario y password para autentificacion
conn1.setRequestProperty("Accept", "application/json");
//se agrega el tipo de respuesta que se desea en este caso json
InputStream content1 = conn1.getInputStream();
//Se alamacena el stream para almacenarlo en content1
BufferedReader in1 = new BufferedReader(new InputStreamReader(content1));
String line1 = "";
StringBuffer result1 = new StringBuffer();
while ((line1 = in1.readLine()) != null) {
    result1.append(line1);
}
line1 = in1.readLine();
//Se almacena toda la informacion que se obtiene en result1
org.json.JSONObject nodosno = new org.json.JSONObject(result1.toString());
//Se obtiene nodosno el cual almacena la informacion de tipo JSON
velocidad = nodosno.getJSONArray("node-connector").getJSONObject(0)
        .getInt("flow-node-inventory:current-speed");
//Se obtiene la velocidad de cada enlace para Dikjstra
```

Código 2.19 Procesamiento de las velocidades de transmisión de los Nodos mediante servicio network-topology

Al disponer de todos los parámetros requeridos para ejecutar el Algoritmo de Dijkstra, se debe proceder con los siguientes pasos. Lo primero es la creación del arreglo de nodos al cual se lo ha denominado arreglodenodos que contiene objetos de tipo nodo.

En segundo lugar se crea un objeto graph1 de tipo Graph donde se almacenarán todos los nodos que se encuentran en el arreglo arreglodenodos. Para obtener las rutas, se utiliza calculoRutaCortaOrigen, que es un método que pertenece a la clase Dijkstra. Como parámetros de entrada del método es necesario agregar al objeto graph1, el cual ya contiene todos los nodos de la topología de red y como segundo parámetro el nodo inicial.

En la variable rutas se almacenan todas las rutas que se van a obtener con el Algoritmo de Dijkstra; este es el primer paso para obtener la ruta completa; dentro de las rutas, para separar los nodos, de forma fácil e inequívoca, se utiliza la bandera "@". En el Código 2.20 se observa el proceso para realizar lo descrito.

```
graph1 = new Graph();//Se crea un objeto de tipo Graph
for (int i = 0; i < numeronodos; i++) {
    graph1.addNodo(arreglodenodos.get(i));//Se agrega todos los nodos a graph1
}
graph1 = Dijkstra.calculoRutaCortaOrigen(graph1, arreglodenodos.get(nodofinal - 1));
//se utiliza el metodo calculoRutaCortaOrigen para calcula la ruta mas corta
rutas = null;
for (Nodo node : graph1.getNodes()) {
//se recorre toda graph1 para obtener todos los nodos
    if (Integer.parseInt(node.getName()) == nodoinicio) {
        int contador1 = 0;//se distingue el nodo inicio
        for (Nodo nodes : node.getShortestPath()) {
            if (contador1 == 0) {
                rutas = nodes.getName();//se agrega el nodo a rutas
            } else {
                rutas = rutas + "@" + nodes.getName();//se agrega las rutas
            }
            contador1++;//se autmenta en contador en una unidad
        }
    }
}
```

Código 2.20 Inicio del proceso del Algoritmo de Dijkstra

Una vez que se tienen las rutas donde se producen los cuellos de botella compartidos, es necesario adjuntar los nodos que fueron eliminados anteriormente, tanto al inicio como al final del cuello de botella compartido ya que sin estos nodos no existiría una comunicación extremo a extremo. Para realizar lo indicado, es necesario ejecutar lo indicado en el Código 2.21, donde se agregan a todas las rutas intermedias los nodos previamente eliminados.

```
if (rutas != null) {//se verifica que rutas no sea un elemento nulo
    rutas = rutas + "@" + nodoinicio;//se agrega rutas con nodoinicio
    rutas1 = null;//se crea rutas1 de forma nula
    contadorrutas = 0;//contadorrutas se reinicia en 0
    for (int j = 0; j < rutas.split("@").length - 1; j++) {//se desplaza por el string rutas
        for (int i = 0; i < switches.get(0).size(); i++) {//se compara por cada switches que se encuentra
            if (Integer.parseInt(switches.get(2).get(i)) == Integer.parseInt(rutas.split("@")[j])) {
                //se compara el path id de switches con el path id del nodo en rutas
                if (Integer.parseInt(switches.get(3).get(i)) == Integer.parseInt(rutas.split("@")[j + 1])) {
                    // se compara el path id de los swites que se encuentran directamente conectados
                    if (contadorrutas == 0) {//si es la primera ruta que esta ingresando se realiza la accion
                        rutas1 = (switches.get(0).get(i)) + "@" + (switches.get(1).get(i));
                    } else {//si es desde la segunda ruta se realiza la siguiente accion
                        rutas1 = rutas1 + "@" + (switches.get(0).get(i)) + "@" + (switches.get(1).get(i));
                    }
                    switches.get(2).set(i, "0");//se elimina los nodos que fueron evaluados
                    switches.get(3).set(i, "0");//se elimina los nodos que fueron evaluados
                    contadorrutas++;//Se agrega en una unidad la variable contadorrutas
                }
            }
            if (Integer.parseInt(switches.get(3).get(i)) == Integer.parseInt(rutas.split("@")[j])) {
                //Se compara nuevamente la matriz switches con las rutas obtenidas para agregar el nodo
                if (Integer.parseInt(switches.get(2).get(i)) == Integer.parseInt(rutas.split("@")[j + 1])) {
                    switches.get(2).set(i, "0");//se elimina los nodos que ya fueron evaluados
                    switches.get(3).set(i, "0");//se elimina los nodos que ya fueron evaluados
                }
            }
        }
    }
```

Código 2.21 Creación y almacenamiento de las rutas obtenidas del Algoritmo de Dijkstra

Una vez que el arreglo dispone de todas las rutas completas posibles, es necesario conformar las reglas reactivas con los 5 elementos de la tupla, pero solo se utilizan 4 de ellos ya que no se toma en cuenta el protocolo de capa de transporte debido a que todos los paquetes de tipo MP-TCP utilizan el protocolo TCP.

También se instalaron reglas proactivas para que todos los paquetes de tipo TCP se envíen al controlador esto se lo realiza sobre el método onNodeUpdate.

En el Código 2.22 se puede observar la utilización de la función reglasrutasMPTCP para la creación de las reglas de los 5 elementos de la tupla.

Las reglas que se van a instalar son tanto para transmitir como para recibir la información, por esta razón esta función se utiliza dos veces en la creación de las reglas.

```
NodeConnectorId ncid = new NodeConnectorId(rutanodos[i]);//Id del conector al que se desea instalar la regla
        reglasrutasMPTCP(ncid, ipDestinoAnalizador, ipFuenteAnalizador, mensajes.getPuertoFuente(),
                mensajes.getPuertoDestino(), 11, n);
        //Intalacion de la regla por medio de la funcion reglasrutasMPTCP
```

Código 2.22 Creación de la regla con los 5 elementos de la tupla para la ruta principal

La función para instalar las reglas de los 5 elementos de una tupla se presenta en el Código 2.23 y en el Código 2.24.

Para la creación de cualquier tipo de regla es necesario considerar los siguientes pasos:

- **Creación del flujo:** Flujo es el conjunto de paquetes que tienen ciertas semejanzas, como ejemplos, pueden ser: igual dirección MAC tanto de origen como de destino; el mismo protocolo de transporte.
- **Creación del campo de *matching*:** este campo está relacionado con la coincidencia de parámetros como requisito para que se realice una acción determinada. Para poder procesar este campo es necesario la utilización de la clase MatchBuilder que es un *builder*[50].
- **Creación del campo de acciones:** En este campo se agregan las instrucciones necesarias cuando un flujo determinado realiza un *match* con los parámetros configurados en el campo de *matching*. Para acumular las instrucciones se utiliza la clase ActionBuilder.
- **Instalación de las reglas:** Para la instalación de las reglas es necesario la creación del flujo, para hacerlo es necesario utilizar un *builder* de tipo FlowBuilder, en el que se añadirá al Flow creado el matching y las acciones.

[50] *Builder*: es un patrón que se utiliza para la creación de objetos complejos compuestos de varias partes que son independientes

Para la función reglasrutasMPTCP se tienen ocho parámetros de entrada, tal como se puede observar en el Código 2.22, los cuales son:

- ncid, acepta objetos de tipo NodeConnectorId que contiene información sobre el identificador del switch en el cual se instalará la regla.
- Cuatro elementos que servirán para construir la tupla de 5 elementos. Se utilizan solo 4 argumentos ya que todos los paquetes son de tipo TCP; los dos primeros parámetros sirven para indicar la dirección IP de origen (ipFuente) y para la dirección IP de destino (ipDestino), estos parámetros son de tipo String; los dos últimos parámetros son de tipo entero y se utilizan para la asignación de los puertos de origen (puertodeentrada) y de destino (puertadesalida).
- Parámetro selector de tipo int, este elemento es utilizado para agregar un identificador único para poder diferenciar a las reglas creadas.
- Parámetro selectorcontrolador de tipo int, este elemento puede tomar cualquier valor, pero solo se activa cuando el valor es igual a cero ya que de esta manera determina que todo el tráfico pase por el controlador.
- El parámetro n tipo int se utiliza para crear un nuevo identificador para la instalación de las reglas de cada nodo el cual se une con el identificador del puerto de origen.

En la función reglasrutasMPTCP es necesario crear un arreglo para las acciones que se desean implementar en las reglas. Al crear el arreglo se tienen que agregar las acciones. Como primer paso se agrega el tamaño del paquete que se va a transmitir mediante la regla; en este caso se utilizará el tamaño completo porque se desea que el mensaje llegue de forma íntegra de extremo a extremo. Después se crea un flujo mediante FlowBuilder al que se le asigna un identificador único para poder procesarlo más adelante. Para la creación de los campos de matching se realizará un emparejamiento por cada elemento de la tupla, en este caso se obtienen un total de 5 condiciones de *matching* que se describen a continuación:

- **IpMatchBuilder**: Este *match builder* está destinado para elegir el tipo de protocolo sobre IP, para poder seleccionarlo se utiliza el método setIPProtocol con el valor de 6 que pertenece a TCP.
- **Ipv4MatchBuilder**: con este *match builder* se seleccionan todas las direcciones IPs coincidentes, para realizar este *matching* es necesario la utilización de dos métodos: setIpv4Destination que agrega la dirección IP de destino y setIpv4Source que agrega la dirección IP de origen.

- **TcpMatchBuilder**: este *match builder* tiene la finalidad de agregar los puertos de cada aplicación con los cuales se realizará el *matching*. Para establecer dichos puertos se utilizan dos métodos: setTCPSourcePort, para agregar el puerto de origen; y, SetTCPDestinationPort, para agregar el puerto de destino.

Una vez creados todos los elementos de *matching*, es momento de crear un objeto de tipo Match en el cual se almacenará todos los elementos de tipo *match builder*, para esta tarea se utilizan cuatro métodos, los cuales son: setEthernetMatch, setIpMatch, setLayer3Match y setLayer4Match.

Cada objeto de tipo *Match* se debe agregar al objeto de tipo FlowBuilder mediante el método setMatch, además se agrega información adicional como: el tiempo de instalación de la regla que se va a aplicar (setHardTimeOut), tiempo de vida de la regla dentro de un switch cuando ya no ingresen paquetes (setIdleTimeout), también se agrega la prioridad que tiene dentro de las reglas (setPriority), se agrega la cookie que es un identificador numérico de la regla en los switches OpenFlow (setCookie) y, por último, se agrega la sección de banderas (setFlags).

Al tener ya el flujo con las diferentes opciones que se describieron anteriormente, es necesario crear instrucciones para agregar al FlowBuilder. Para poder asociar instrucciones al flujo es necesario crear un objeto de tipo Instructions, donde se utilizará un método de flowbuilder llamado setInstructions que tiene como parámetro de entrada un objeto de tipo ApplyActionsCase.

Para la creación de ApplyActionsCase, el cual es necesario para la aplicación de acciones de la regla, es necesaria la creación de un objeto de tipo ApplyActions a través del método setAction agregando el arreglo de tipo Action con las acciones que ya se configuraron al inicio de la función reglasrutasMPTCP. El paso final para instalar las instrucciones es utilizar el método que pertenece a la clase FlowBilder llamado setInstruccion.

Para instalar una regla dentro de un switch, se debe crear un tipo InstanceIdentifier asociado al objeto Flow; para poder inicializarlo, se utiliza un método llamado child de Flow agregando el FlowId del FlowBuilder. Adicionalmente, se utiliza el objeto GenericTransactionUtils, el cual permite la interacción con el *Data Broker*, para poder instalar las reglas en los switches es necesario utilizar el método writeData que contiene cinco parámetros de entrada que se describen a continuación:

- El primer parámetro es un objeto de tipo DataBroker donde se almacenará la información de la regla.
- El segundo parámetro es un objeto de tipo LogicalDatastoreType.
- El tercer parámetro es un objeto InstanceIdentifier de tipo Flow el cual contendrá el flujo que se desea instalar.
- El cuarto parámetro agrega el FlowBuilder.
- Finalmente se agrega una variable de tipo boolean para instalar la regla.

La creación e instalación de la regla se la puede observar en el Código 2.23 y Código 2.24 dentro de la función reglasrutasMPTCP. Al instalar las reglas en los switches, se termina el procesamiento de los paquetes MP_CAPABLE, el siguiente paso es el procesamiento de los mensajes MP_JOIN.

```
private void reglasrutasMPTCP (NodeConnectorId ncid, String ipDestinoAnalizador, String ipFuenteAnalizador,
    int puertodeentrada, int puertodesalida, int selector, int tipoMPTCP) {
ArrayList<Action> accciones_salida = new ArrayList<Action>();//Se crea una lista con las acciones
accciones_salida.add(new ActionBuilder().setOrder(0).setKey(new ActionKey(1))
        .setAction(new OutputActionCaseBuilder().setOutputAction(new OutputActionBuilder().setMaxLength(0xffff)
                .setOutputNodeConnector(ncid).build()).build()).build());
                //se agrega las acciones para la regla
if (selector == 1) {
    accciones_salida.add(new ActionBuilder().setOrder(1).setKey(new ActionKey(2))
            .setAction(new OutputActionCaseBuilder().setOutputAction(new OutputActionBuilder().setMaxLength(0x64) // 100
                    .setOutputNodeConnector(new Uri(OutputPortValues.CONTROLLER.toString()))// PUERTO
                    .build()).build()).build());
            //si la bandera selector se encuentra en 1 se agrega la siguiente accion para que se envien los datos al
            //controlador como tambien al puerto de salida que se desee
}
FlowBuilder floodFlow = new FlowBuilder().setTableId(tabla_id).setFlowName("5e");//creacion de FlowBuilder
floodFlow.setId(new FlowId(selector + "regla" + tipoMPTCP));//se agrega un nombre unico a la clase floodFlow
IpMatchBuilder constructor_ipmatch = new IpMatchBuilder();//IpMatchBuilder se configura el matchin para TCP 6
constructor_ipmatch.setIpProtocol((short) 6).build();// se configura con el valor de 6 que pertenece a TCP
//EthernetMatchBuilder se configura el campotipo de ehternet para realizar matching con IP 800
EthernetMatchBuilder ethernetMatch_constructor = new EthernetMatchBuilder().setEthernetType(new EthernetTypeBuilder()
.setType(new EtherType((long) 0x800)).build());
contadorreglas++;
//con Ipv4MatchBuilder se crea el matching para las direcciones IPv4
Ipv4MatchBuilder ipv4matchdes = new Ipv4MatchBuilder();
ipv4matchdes.setIpv4Destination(new org.opendaylight.yang.gen.v1.urn.ietf.params.xml.ns.yang.ietf.inet.types.
rev100924.Ipv4Prefix(ipDestinoAnalizador + "/32"));//se agrega la direccion IP con su mascara
```

Código 2.23 Función reglasrutasMPTCP (1)

```
    ipv4matchdes.setIpv4Source( new org.opendaylight.yang.gen.v1.urn.ietf.params.xml.ns.yang.ietf.inet.
    types.rev100924.Ipv4Prefix(ipFuenteAnalizador + "/32"));//se agrega la direccion IP con su mascara origen
    TcpMatchBuilder tcpmatchPF = new TcpMatchBuilder();//se realiza match para el puerto de capa de transporte
    tcpmatchPF.setTcpSourcePort(new org.opendaylight.yang.gen.v1.urn.ietf.params.xml.ns.yang.ietf.inet.
    types.rev100924.PortNumber( puertodeentrada));//se agrega el puerto de origen para el match
    tcpmatchPF.setTcpDestinationPort(new org.opendaylight.yang.gen.v1.urn.ietf.params.xml.ns.yang.ietf.
    inet.types.rev100924.PortNumber(puertodesalida));//se agrega el puerto de destino para el match
    //se crea el objeto match y se agrega todas las opciones de matchin antes descritas
    Match match = new MatchBuilder().setEthernetMatch(ethernetMatch_constructor.build()).setIpMatch
    (constructor_ipmatch.build()).setLayer4Match(tcpmatchPF.build()).setLayer3Match(ipv4matchdes.build()).build();
    // SE CONFIGURA A floodFlow el campo match con prioridad 100
    floodFlow.setMatch(match).setPriority(prioridad_flujo_proactivo + 100).setCookie(new FlowCookie
    (BigInteger.valueOf(cookie_uno.getAndIncrement()))).setFlags(new FlowModFlags(false, false, false, false, false));
    ApplyActions acciones_aplicar = new ApplyActionsBuilder().setAction(ImmutableList.copyOf(accciones_salida)).build();
    // instrucciones_acciones_aplicar se configura en funcion de .setApplyActions(acciones_aplicar)
    Instruction instrucciones_acciones_aplicar = new InstructionBuilder().setOrder(0)
            .setInstruction(new ApplyActionsCaseBuilder().setApplyActions(acciones_aplicar).build()).build();
    // constructor_reglas_flood, resultado de la funcion creacion_campo_matching_reglas_proactivas que devuelve un objeto
    // FlowBuilder con su campo de matching se configura Instructions con ImmutableList.of(instrucciones_acciones_aplicar)
    floodFlow.setInstructions(new InstructionsBuilder().setInstruction(ImmutableList
    .of(instrucciones_acciones_aplicar)).build());
    //Creacion de la instancia_flujo para crear un nuevo camino
    NodeId id_switch = InventoryUtils.getNodeId(ncid);
    InstanceIdentifier<Flow> flujo = InstanceIdentifier.builder(Nodes.class)
            .child(Node.class, new NodeKey(id_switch)).augmentation(FlowCapableNode.class)
            .child(Table.class, new TableKey((short) 0))// Tabla de flujo 0
            .child(Flow.class, new FlowKey(new FlowId(floodFlow.build().getId().getValue()))).build();
    //Acciones para guardar la regla dentro de dataBroker del controlador
    GenericTransactionUtils.writeData(dataBroker, LogicalDatastoreType.CONFIGURATION, flujo, floodFlow.build(),true);
}
```

Código 2.24 Función reglasrutasMPTCP (2)

Antes de manejar los mensajes MP_JOIN, es necesario filtrar nuevamente los mensajes MP_CAPABLE que se envían en un ACK, esto se lo realiza para obtener las llaves tanto del *host* transmisor como del *host* receptor; estas llaves son utilizadas para diferenciar entre conexiones MP_TCP. Para poder realizar lo mencionado se presenta el Código 2.25.

```
if(ejemploint==0 && mensajes.getLongitudCampoMP_Capable()==20) {
//se filtran los mensajes Capable de tipo ACK
```

Código 2.25 Filtro para paquetes MP_CAPABLE que se envia en un ACK

La llave que se obtiene del receptor se encuentra cifrada mediante SHA-1, para obtener el valor numérico es necesario descifrarla. Además en los mensajes MP_JOIN solo se envían los 32 bits más significativos de la llave del receptor, para realizar lo antes descrito se utiliza el Código 2.26. En primer lugar se obtiene la llave del receptor para agregarla a un arreglo de tipo byte, el tamaño del arreglo es de 160 bits o 20 bytes. El siguiente paso es crear un objeto MessageDigest para obtener el descifrado SHA-1 de la llave del receptor por medio del método getInstance. De los 20 bytes de la llave del receptor solo se necesitan los 4 bytes más significativos, esto se lo realiza mediante un lazo para seleccionar la información importante para poder identificar los mensajes de una conexión MP-TCP. En el Código 2.26 se presentan los pasos que se describieron anteriormente.

```
BigInteger foo1 = new BigInteger(mensajes.getLlaveDelReceptor());
//se crea a foo1 de tipo BigInteger ya que la llave es muy grande
byte[] array = foo1.toByteArray();
//se transforma a un array de bytes la variable foo1
if (array[0] == 0) {
    byte[] tmp = new byte[array.length - 1];
    //se crea una variable intermedia para no perder informacion
    System.arraycopy(array, 1, tmp, 0, tmp.length);
    //se copia el arreglodenodos
    array = tmp;//se realiza un intercambio de variable
}
MessageDigest md1;//se crea una variable MessageDigest
md1 = MessageDigest.getInstance("SHA-1");
//Se agrega el tipo de sifrado en este caso SHA-1
md1.update(array, 0, array.length);
// se actualiza md1 con la informacion de array
byte[] bytes3 = md1.digest();
byte[] nj = new byte[4];
//solo se utiliza los 5 bytes mas significativos de la llave
for (int i = 0; i < 4; i++) {
    nj[i] = bytes3[i];
}
BigInteger keyB32 = new BigInteger(1, nj);
```

Código 2.26 Obtención de la llave del *host* receptor

Al obtener la llave del receptor, se tienen que asociar las rutas obtenidas ya que cada llave se encuentra ligada con una conexión MP-TCP en un HashMap donde la clave del elemento es la llave del receptor y el valor es un objeto Rutas asociado con los siguientes elementos: las rutas obtenidas mediante el Algoritmo de Dijkstra; los 32 bits de la llave del receptor; el número total de las rutas a la que se le resta una unidad ya que esta se utilizó para establecer el subflujo principal de datos; y, por último, el contador de mensajes MP_JOIN para poder utilizar las rutas generadas. En el Código 2.27 se puede observar lo anteriormente descrito.

```
llaverutas.put(keyB32.toString(),
new Rutas(rutascortas, keyB32.toString(), rutascortas.size() - 1, contadorJOIN));
// se agrega a llaverutas la key y la ruta para poder asociarlos
```

Código 2.27 Almacenamiento de las rutas con las llaves respectivas

Con el último paso descrito se termina el procesamiento de los mensajes MP_CAPABLE; en este momento es necesario el procesamiento de mensajes MP_JOIN de tipo SYN, para lo cual es necesario aplicar el Código 2.28 donde se usa un condicional el cual va a comparar y verificar que el mensaje MP_JOIN sea de tipo SYN, que puede asignar una ruta al subflujo.

```
if (ejemploint == 1 && mensajes.getLongitudDelCampoMP_Join() == 12) {
    //verificacion para que solo ingrese cuando sea mensaje tipo JOIN
    // y que sea el primer mensaje que se envia
```

Código 2.28 Filtro para mensajes MP_JOIN

Para poder asignar las rutas que se obtienen con el Algoritmo de Dijkstra es necesario utilizar el contador de rutas del objeto Rutas, ya que cada vez que se asigna un camino, el algoritmo resta en una unidad y se compara con 0 hasta que se distribuyen todos los caminos disponibles en la infraestructura de red. Lo antes mencionado se encuentra representado en el Código 2.29.

```
if (llaverutas.get(mensajes.getJoinTokenString()) != null) {
    //se verifica que las llaves no sean nulas
    if (llaverutas.get(mensajes.getJoinTokenString()).getconador() != 0) {
    //tambien se verifica que el contador de las rutas sea diferente de 0
```

Código 2.29 Filtro para mensajes de una misma conexión

En el Código 2.30 se observa cómo se obtienen las rutas encontradas con el Algoritmo de Dijkstra y como almacenarlas en una variable de tipo String, además es importante obtener el contador del total de rutas que se puede asignar para cada conexión MP-TCP, ya que es un recurso limitado.

```
int contadorJOIN1 = llaverutas.get(mensajes.getJoinTokenString()).getcontadorJOIN();
//Se crea el contador JOIN a partir del elemento que se ecnuentra almacenado
String rutanodos[] = llaverutas.get(mensajes.getJoinTokenString()).getrutascortas()
.get(contadorJOIN1).split("@");
//Se divide las rutas que se encuentran almacenadas y se asigna a una matriz rutanodos
```

Código 2.30 Diferenciación de rutas para los mensajes MP_JOIN

El paso final para el manejo de los paquetes MP_JOIN y la asignación de subflujos, es fijar las rutas a cada nodo (instalar las reglas en cada nodo), para realizar lo antes mencionado se utiliza la función reglasrutasMPTCP que ya se la describió anteriormente, esta función se aplica en dos ocasiones para construir los caminos para el envío y recepción de datos. Además, se adiciona en una unidad al contador Join y se almacena en la clase correspondiente, también se realiza la disminución en una unidad al contador que contiene el total de rutas asignadas, esto se realiza para llevar un control de los rutas, sin este control existiría la posibilidad de sobrelapamiento de caminos que ya fueron asignadas. Lo anteriormente descrito se encuentra presentado en el Código 2.31 de forma detallada.

```
for (int i = 0; i < rutanodos.length;) {
    if(selfr==1) {
        NodeConnectorId ncid = new NodeConnectorId(rutanodos[i]);
        //se obtiene el id del conectro asociado al nodo
        reglasrutasMPTCP (ncid, ipDestinoAnalizador, ipFuenteAnalizador, mensajes.getPuertoFuente(),
                mensajes.getPuertoDestino(), 13 + contadorJOIN1, 0, n);
        //se instala la regla en el nodo que se establece entrada
        i++;
        ncid = new NodeConnectorId(rutanodos[i]);
        reglasrutasMPTCP (ncid, ipFuenteAnalizador, ipDestinoAnalizador, mensajes.getPuertoDestino(),
                mensajes.getPuertoFuente(), 14 + contadorJOIN1, 0, n);
        //se instala la regla en el nodo que se establece salida
        i++;
    }
}
contadorJOIN1++;//se agrega en una unidad al contadorJOIN1
llaverutas.get(mensajes.getJoinTokenString()).setcontadorJOIN(contadorJOIN1);
//se almacena en una unidad al contador JOIN1
llaverutas.get(mensajes.getJoinTokenString())
        .setcontador(llaverutas.get(mensajes.getJoinTokenString()).getconador() - 1);
// se reduce en una unaidad el contador de rutas
```

Código 2.31 Asignación de reglas a las rutas obtenidas

2.5. Despliegue de la infraestructura para utilizar OpenStack

Para el despliegue de la infraestructura generada mediante OpenStack se utilizó la herramienta DevStack, la cual facilita la instalación. DevStack contiene varios *scripts* que facilitan la instalación y configuración del entorno de OpenStack [58]. Los *scripts* de DevStack se encuentran en el repositorio de GitHub[51], los cuales se encuentran actualizados con la última versión de OpenStack. DevStack ha evolucionado para admitir

[51] GitHub: es un repositorio para almacenar proyectos utilizando el sistema de control de versiones Git.

una gran cantidad de opciones de configuración de OpenStack así como también tiene servicio de soporte técnico. En los *scripts* de DevStack se encuentran incluidas pruebas unitarias y funcionales para comprobar el funcionamiento de OpenStack en el momento de la instalación así como también de su funcionamiento. Existe una advertencia sobre DevStack que se refiere a que solo debe ejecutarse sobre servidores o máquinas virtuales dedicadas para OpenStack, ya que DevStack realiza cambios en el sistema cuando ejecuta la instalación de OpenStack.

2.5.1. Instalación y Manejo de DevStack

Para la instalación de OpenStack se utilizó un entorno virtual con las características que se presenta en la Tabla 2.2.

Tabla 2.2 Características de la máquina virtual

Características	Especificación
Disco Duro	250 GB
Memoria RAM	6 GB
Procesadores	2

Para la instalación y buen funcionamiento de DevStack se utiliza el sistema operativo Ubuntu 16.04 ya que lo recomienda la página oficial [58].

En un inicio se estableció como sistema operativo Ubuntu 14.04, pero algunos de los repositorios que utiliza DevStack no fueron encontrados por los *scripts,* por lo que generó un error al momento de la instalación de OpenStack. El primer paso para la instalación y administración de DevStack es crear un usuario en Ubuntu denominado stack, como se indica en la Figura 2.10.

```
fellas@ubuntu:~$ sudo useradd -s /bin/bash -d /opt/stack -m stack
```

Figura 2.10 Creación del usuario stack

El usuario stack realizará muchos cambios en el sistema operativo, por este motivo el usuario debe tener permiso de súper usuario como se observa en la Figura 2.11.

```
fellas@ubuntu:~$ echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
fellas@ubuntu:~$ sudo su - stack
```

Figura 2.11 Creación del usuario stack

Para poder crear un repositorio dentro de la máquina virtual primero se tiene que instalar Git para poder clonar el repositorio de DevStack esto se realiza en la Figura 2.12.

```
fellas@ubuntu:~$ sudo apt-get install git
stack@ubuntu:~/devstack$ git clone https://git.openstack.org/openstack-dev/devstack
```

Figura 2.12 Clonación del repositorio de DevStack

Para la configuración de la instalación de DevStack se utiliza el archivo local.conf, para una instalación básica se puede observar el código de la Figura 2.13. Para encontrar la configuración completa de DevStack es necesario dirigirse a la dirección /devstack/examples/local.conf.

```
[[local|localrc]]
ADMIN_PASSWORD=fellas
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
```

Figura 2.13 Configuración básica de local.conf

Para realizar la instalación de OpenStack es necesario activar el script stack.sh. Hay que tomar en cuenta que para la instalación de los repositorios es necesario que el puerto 9418 de TCP que utiliza Git se encuentre desbloqueado, ya que sin este puerto no se instalan los repositorios que utiliza DevStack. La instalación de OpenStack mediante DevStack tarda alrededor de 30 minutos usando un enlace de 20 Mbps, este tiempo es relativo ya que depende de la conexión de Internet disponible.

Al finalizar la instalación de OpenStack se presentará la Figura 2.14 donde se encuentra el tiempo total que estuvo activo DevStack, además se especifican las acciones realizadas así como también la dirección IP en la cual se encuentra el servidor de OpenStack. En el caso de esta instalación la dirección IP fue asignada mediante DHCP.

```
DevStack Component Timing
 (times are in seconds)
=========================
run_process              62
test_with_retry           4
apt-get-update           17
pip_install             479
osc                     168
wait_for_service         33
git_timed               291
dbsync                   40
apt-get                 105
-------------------------
Unaccounted time        685
=========================
Total runtime          1884



This is your host IP address: 192.168.253.182
This is your host IPv6 address: ::1
Horizon is now available at http://192.168.253.182/dashboard
Keystone is serving at http://192.168.253.182/identity/
```

Figura 2.14 Finalización de instalación de OpenStack

Una vez finalizada la instalación de DevStack es posible utilizar OpenStack. DevStack instalará los siguientes elementos de OpenStack:

- Keystone [59]: es un servicio de OpenStack que brinda una API de autenticación y autorización de múltiples inquilinos (*tenants*).
- Glance [60]: es un servicio que provee imágenes ISO que los usuarios pueden cargar y descargar. Además, Glance permite descubrir, registrar y recuperar imágenes de máquinas virtuales.
- Nova [61]: es un servicio que proporciona una forma de aprovisionar instancias de cómputo. Este servicio permite la creación de máquinas virtuales y servidores *bare-metal* mediante Ironic[52].
- Cinder [62]: es un servicio de almacenamiento en bloque, está diseñado para presentar recursos de almacenamiento a los usuarios finales.
- Neutron [63]: es un servicio de OpenStack para proporcionar "redes como servicio" entre dispositivos que se encuentran en la plataforma.
- Horizon [64]: es el *dashboard* [53]que proporciona una interfaz web para el manejo y administración de los servicios de OpenStack para usuarios finales.

Además, DevStack proporciona y configura la dirección IP flotantes[54] para que tanto los administradores como los invitados puedan tener acceso al Internet o puedan enlazarse a otro tipo de nube de cómputo.

Para la administración de OpenStack existen dos métodos, el primero de ellos es mediante la interfaz web que proporciona Horizon, la cual es fácil de utilizar e intuitiva. La segunda forma de administrar es por comandos propios de OpenStack mediante el *Shell* del sistema operativo en el que se encuentra instalado OpenStack.

Para la configuración de las pruebas se tiene que ingresar a la dirección /opt/stack/tempest y utilizar el comando *tempestad* para realizar todas las pruebas que se configuren mediante DevStack.

En la Figura 2.15 se presenta la interfaz de Horizon de la instalación que se realizó. La instalación se encuentra sin ningún elemento de prueba ya que fue capturada la imagen inmediatamente luego de terminar su instalación.

[52] Ironic es un servicio de OpenStack que tiene como objetivo proporcionar máquinas virtuales de tipo *bare-metal* en lugar de máquinas virtuales de tipo host.
[53] *Dashboard:* es una Interfaz gráfica de usuario que utiliza en algunos sistemas operativos donde el usuario puede administrar el sistema operativo o el software.
[54] IP flotantes: son direcciones de IP públicas que se asocian a una instancia de máquina virtual para acceder a ellas desde afuera de OpenStack.

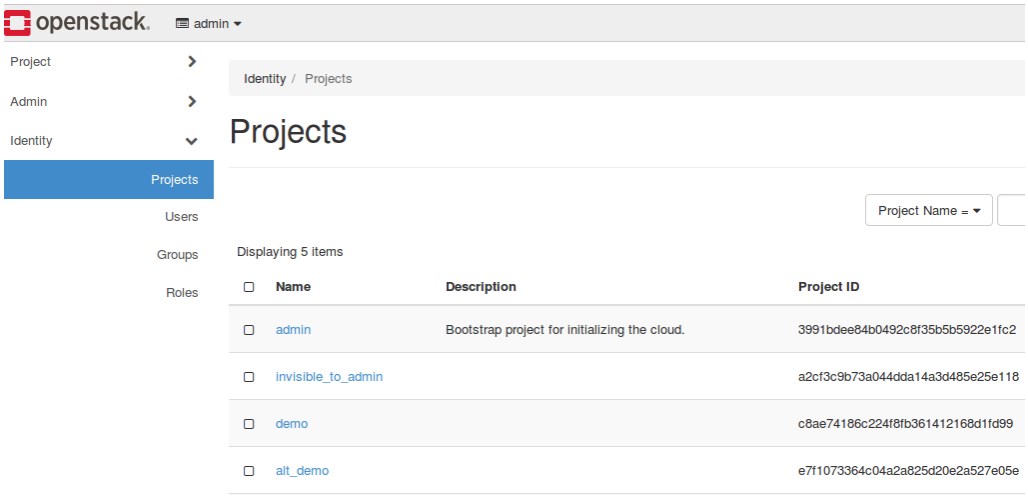

Figura 2.15 *Dashboard* de Horizon en OpenStack

2.6. Automatización con DevOps

Para el manejo de DevOps se utiliza la herramienta Ansible, la cual es una plataforma de software libre que permite la administración y configuración de *hosts* de forma remota; esta herramienta fue creada en el año 2012 por Michael DeHaan que basó su trabajo en la herramienta para DevOps Puppet. Se seleccionó esta herramienta por su versatilidad y fácil manejo del lenguaje de programación que permite interactuar con Ansible [65].

2.6.1. Ansible

Ansible tiene tres características que se diferencian de otras herramientas para DevOps, estas son: gestión de la configuración, despliegue de servidores y ejecución de tareas ad-Hoc. Además de lo anteriormente expuesto, Ansible tiene los siguientes objetivos [66] [67]:

- **Limpio:** Ansible utiliza el lenguaje YAML[55] el cual utiliza una sintaxis simple. Las APIs de Ansible son simples de utilizar.
- **Rápido:** Ansible tiene las siguientes características: rápido de aprender, configuración rápida ya que no necesita de agentes ni de daemons en los servidores para ejecutar una acción.
- **Completo:** Como ya se mencionó, la herramienta Ansible realiza tres operaciones de forma autónoma y automática (*plug & play*), esto quiere decir que Ansible es una herramienta completa.
- **Eficiente:** Ansible es una herramienta que no consume recursos adicionales de cómputo en los servidores. Los módulos de Ansible funcionan a través de JSON, además es extensibles mediante módulos escritos en YAML.

[55] YAML es un formato de serialización de datos basado en lenguajes como XML, C, Python, Perl.

- **Seguro:** Ansible utiliza el protocolo SSH por lo cual no necesita de puertos adicionales así como tampoco necesita de daemons, por lo que no crean huecos de seguridad en los servidores.

Ansible utiliza una conexión remota con otro *host* para realizar las configuraciones y acciones deseadas, de manera predeterminada utiliza una conexión SSH. Otra de las ventajas que tiene Ansible es conseguir un equilibrio en los entornos de desarrollo y producción, por lo que ayuda al aprovisionamiento y despliegue de ambientes de desarrollo.

2.6.1.1. Elementos de Ansible

A continuación se presentan algunos de los elementos más importantes que intervienen en la herramienta Ansible:

- **Servidor:** es la máquina donde se encuentra instalada la herramienta Ansible, desde esta máquina se ejecutarán las configuraciones y *tasks* en los *host* clientes.
- **Host clientes:** a estos elementos se los pueden definir en el archivo /etc/ansible/hosts del servidor Ansible. Los *hosts* clientes solo necesitan del servicio SSH y del lenguaje de programación Python para que el servidor Ansible se comunique con ellos y realice los cambios especificados.
- **Inventory:** es el archivo donde se registran los *host* clientes. En este archivo se puede agrupar a un conjunto de *hosts* que tienen una misma finalidad o similares necesidades.
- **Playbook:** En este archivo se encuentran especificadas las tareas que se desean realizar en los *hosts* clientes. El lenguaje que se utiliza *playbook* es YAML.
- **Tasks:** Este elemento se lo define dentro de los *playbook* como tareas que van a ser ejecutadas en el *host* clientes.
- **Module:** Este elemento permite ejecutar tareas de forma fácil sin la creación de un script o de un playbook.
- **Handler:** este elemento es similar a un *task*, pero solo se ejecuta si la tarea tiene la directiva *notify* que se presenta cuando se ha producido un cambio.

2.6.1.2. Instalación de Ansible

Para la instalación de Ansible se utilizó una máquina virtual con la distribución de Ubuntu 14.01, en este caso, a esta máquina se la denomina servidor. En la Figura 2.16 se observa los pasos para la instalación de Ansible en el servidor; como primer paso se tiene que instalar como requisito previo, la herramienta software-properties-common. Una vez instalados los complementos de Ansible, se tiene que agregar el repositorio que

contiene el software para que, finalmente, mediante el comando apt-get install ansible se pueda instalar Ansible.

```
root@ubuntu:/home/fellas# apt-get install software-properties-common
root@ubuntu:/home/fellas# apt-add-repository ppa:ansible/ansible
root@ubuntu:/home/fellas# apt-get install ansible
```

Figura 2.16 Instalación de Ansible en el *host* servidor

Para las primeras pruebas con Ansible, se utilizó el archivo hosts, descrito anteriormente; en dicho archivo se agregó un solo *host* cliente, el cual es utilizado para pruebas. En este ejemplo en particular, se desplego un conjunto de servidores a los que se les llamo webserver, como parte de este conjunto se encuentra el *host* de pruebas con la dirección IP 192.168.253.181. En la Figura 2.17 se puede observar lo descrito.

```
# Ex 2: A collection of hosts belonging to the 'webservers' group

## [webservers]
## alpha.example.org
## beta.example.org
## 192.168.1.100
## 192.168.1.110
[webserver]
192.168.253.181
```

Figura 2.17 Archivo hosts

Para establecer la conexión entre el servidor que contiene Ansible con los *hosts* clientes, se tienen que generar un par de llaves de tipo RSA[56]. Para la creación de estas llaves es necesaria la utilización del comando ssh-keygen, al ejecutarlo se solicita una frase para la generación de las llaves, esto se puede observar en la Figura 2.18.

```
root@ubuntu:/home/fellas# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
d2:ea:b3:5e:0e:2a:09:b1:8f:02:f1:d3:c1:cd:eb:bb root@ubuntu
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|   . o           |
|o   o o.         |
| = . ...S        |
|+ o . .o         |
|.+ o .o .        |
|o +  oo+         |
|.  ...E=.        |
+-----------------+
```

Figura 2.18 Generación de llaves RSA

[56] RSA: es un sistema criptográfico de clave pública desarrollado en 1977. Es el primer y más utilizado algoritmo de este tipo y es válido tanto para cifrar como para firmar digitalmente.

Una vez que se tienen las llaves RSA creadas en el servidor Ansible, deben ser enviadas al *host* cliente, para ello se utiliza el comando ssh-copy-id. Las llaves creadas se encuentran en /.ssh/id_rsa.pub. En la Figura 2.19 se presenta el intercambio de llaves utilizando el comando anteriormente mencionado.

```
root@ubuntu:/home/fellas# ssh-copy-id -i ~/.ssh/id_rsa.pub fellas@192.168.253.18
1
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompt
ed now it is to install the new keys
fellas@192.168.253.181's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'fellas@192.168.253.181'"
and check to make sure that only the key(s) you wanted were added.
```

Figura 2.19 Intercambio de llaves RSA con el *host* cliente

Para realizar pruebas de la conexión con los clientes, se utiliza el comando *ad-hoc* en el servidor, el cual ejecuta un *ping* al host cliente, el *host* cliente le responde de forma afirmativa con un *pong*, pero si no se encuentra configurado alguno de los complementos de Ansible como Python o SSH en el cliente, la respuesta es un *crash*. En la Figura 2.20 se puede observar la ejecución del comando *ping* mediante un comando *ad-hoc*, además al comando se agregó el usuario del *host* cliente mediante la opción –u que en este caso es fellas. Como resultado se puede observar las letras de color verde en el que se especifica la dirección IP a la que se está conectando Ansible, que en este caso es la 192.168.253.181, seguido de la palabra *success*, entre llaves se encuentra el resultado que se obtiene del comando ping, este comando como no realiza cambios en la configuración del *host* cliente tiene un resultado de *false* pero como si se ejecuta el comando se tiene una respuesta con *pong*.

```
root@ubuntu:/home/fellas# ansible all -m ping -u fellas
Enter passphrase for key '/root/.ssh/id_rsa':
192.168.253.181 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Figura 2.20 Prueba de funcionamiento de Ansible

2.7. Despliegue de la aplicación como función de red virtualizada

Para la creación de la función de red virtualizada se utilizaron los principios de DevOps que automatiza el despliegue del servicio que solventa el problema del *shared bottleneck* que se produce en las conexiones MP-TCP [68].

Lo primero que se realizó fue identificar los pasos necesarios para la creación de la función de red, estos son:

- Comprimir la carpeta que contiene OpenDayLight con el *feature* que solventa el problema de *shared bottleneck* en MP-TCP.
- Para el funcionamiento de OpenDayLight es necesario la instalación de Java versión 7 o superiores y de Maven 3.3.
- Es necesario que en el *host* cliente se instale el protocolo MP-TCP.
- Es necesario la instalación de Mininet para la creación de la infraestructura de red para realizar las pruebas.
- Reiniciar el *host* cliente para que se pueda utilizar el protocolo MP-TCP.

Para realizar las tareas anteriores se crea un *playbook* de Ansible, donde se especifica cada una de las tareas que anteriormente se describieron.

2.7.1. Creación del Playbook

Para la creación de un *playbook*, se crea un archivo, con la extensión yml, la cual utiliza el lenguaje de programación YAML. En la creación del playbook se tiene que especificar el *host* en el cual se desea ejecutar las tareas para el despliegue de la función de red. En el Código 2.32 se puede observar el proceso para declarar un *host* cliente especificando una dirección IP con la etiqueta de *hosts*; en este caso se realiza la prueba con el cliente con dirección IP 192.168.253.181.

```
- hosts: 192.168.253.181
  become: true
  tasks:
```

Código 2.32 Inicio de la aplicación en Ansible

Como segundo paso se tiene que comprimir la carpeta que contiene el controlador OpenDayLight con el *feature* que solventa el problema del *shared bottleneck* de MP-TCP. Esta tarea consta de dos pasos, ya que el proyecto completo tiene un peso de 615 MB el cual se demoraría mucho tiempo en el transporte de información desde el servidor Ansible al *host* cliente. Por otro lado, si al proyecto de Maven se le realiza un *clean*, el peso total del proyecto se reduce notablemente, a 1,42 MB. La opción con menos peso es la que se escogió ya que el tiempo que se tardaría en el traspaso de información es pequeño. Al utilizar esta opción se tiene que realizar una tarea adicional, la limpieza y compilación del proyecto.

Una vez que ya se tiene el proyecto comprimido, la siguiente tarea es transferir el archivo comprimido, primero se creará una carpeta llamada MPTCP dentro del directorio /root/.

En el Código 2.33 se presenta el proceso para la creación de la carpeta MPTCP, con la etiqueta name se especifica el mensaje que se va a presentar en el *bash*, y a continuación se especifica, mediante la etiqueta file, tanto el path y los permisos que va a tener el directorio.

```
- name: Crea el directorio /root/MPTCP
  file: path=/root/MPTCP state=directory owner=root group=root
```

Código 2.33 Creación del directorio MPTCP con YAML

Con la carpeta creada en el host cliente, el siguiente paso es copiarla desde el servidor. En YAML existe una etiqueta unarchive que permite el traspaso de la información y desempaquetado a un *host* cliente, los parámetros son dos: el primero es la fuente o src, para especificar la ruta en la que se encuentra la carpeta comprimida, que en este caso es /home/fellas/Desktop/bottleneck-Mptcpclean.tar.gz, y, como segundo argumento, se usa dest, para especificar la carpeta donde se va a descomprimir. En el Código 2.34 se puede observar lo mencionado.

```
- name: copiar archivo que contiene el codigo desarrollado y descomprimirlo
  unarchive:
    src=/home/fellas/Desktop/bottleneck-Mptcpclean.tar.gz
    dest=/root/MPTCP/
```

Código 2.34 Transferencia del archivo comprimido en YAML hacia el cliente

Para el funcionamiento del controlador OpenDayLight se tiene que instalar Maven 3.3.0 o versiones superiores, Java versión 8 y el JDK de Java 7. Si alguno de los elementos antes mencionados no se encuentra instalado o no se tiene una versión adecuada, no se puede ejecutar OpenDayLight.

Los paquetes de Maven en la versión 3.3 y de Java en la versión 8 no existen en los repositorios oficiales de Ubuntu 14.04, por esta razón se tienen que agregar repositorios externos para poder instalar las versiones requeridas. Para agregar el repositorio de Maven se utiliza la dirección ppa:andrei-pozolotin/maven3 y para el paquete de Java 8 se utiliza la dirección ppa:webupd8team/java. Para realizar esta acción desde Ansible, se utilizaron los comandos de *bash* mediante la etiqueta shell. En el Código 2.35 se presenta la codificación para realizar las tareas antes descritas.

```
- name: Agregar Repositorio de Maven
  shell: apt-add-repository ppa:andrei-pozolotin/maven3

- name: Agregar Repositorio de Java 8
  shell: add-apt-repository ppa:webupd8team/java -y
```

Código 2.35 Agregar los repositorios de Maven y Java en YAML

El siguiente paso es agregar los repositorios del protocolo MP-TCP, pero no se lo realiza directamente, debido a que primero se deben agregar las llaves mediante un id determinado, que en esta es 379CE192D401AB61 de acuerdo a [69] para poder obtener el paquete de MP-TCP, además se especifica el servidor que contiene las llaves el cual es keys.gnupg.net. Para realizar lo antes mencionado en YAML, se utiliza la etiqueta apt_key donde se determinan los parámetros mediante keyserver e id. En el Código 2.36 se presentan los pasos necesarios para agregar la llave del paquete que contiene el protocolo MP-TCP.

```
- name: Agregar las llaves del servidor que se encuentra los repositorios de MPTCP keyserver
  apt_key:
    keyserver: keys.gnupg.net
    id: 379CE192D401AB61
    state: present
```

Código 2.36 Agregar llaves del paquete del protocolo MP-TCP mediante Ansible

Al tener las llaves ya instaladas en el *host* cliente, es necesario agregar el repositorio, en este caso particular se tiene que manejar directamente el archivo sources.list, en este archivo se encuentra alojados los repositorios de los paquetes de software que pueden ser instalados, actualizados, borrado, etc. El cual se encuentra en /etc/apt/; al final del archivo se agrega la siguiente línea: deb https://dl.bintray.com/cpaasch/deb jessie main. Para agregar la última línea al archivo sources.list se tiene que verificar siempre si la línea ya está presente, para no duplicarla, ya que al momento de actualizar los repositorios la duplicación va a provocar un error y no se podrá actualizarlos.

Por la razón antes expuesta, mediante un condicional se verifica que no exista la línea deb https://dl.bintray.com/cpaasch/deb jessie main, si existe la línea antes mencionada no se realiza ningún cambio, caso contrario, solo se escribe un mensaje en el *bash*: No fue necesario modificar sources.list. En el Código 2.37 se presentan las tareas antes descritas, esto se lo realiza mediante la etiqueta shell en YAM además se utiliza el condicional if con las condiciones anteriormente señaladas.

```
- name: Instalar el repositorio de MPTCP
  shell: if [ `cat /etc/apt/sources.list | grep -c "dl.bintray.com/cpaasch/deb"` == 1 ];
  then echo "No fue necesario modificar sources.list";
  else echo "deb https://dl.bintray.com/cpaasch/deb jessie main" >> cat /etc/apt/sources.list ; fi
```

Código 2.37 Agregar repositorio de MP-TCP en YAML

En este punto ya se encuentran agregados todos los repositorios en el *host* cliente, la siguiente tarea es actualizarlos, esto se lo realiza en YAML mediante la etiqueta apt y agregando el parámetro update_cache: yes. En el Código 2.38 se presentan las tareas para actualizar los repositorios en YAML.

```
- name: Actualizar a apt packages
  become: true
  apt:
    update_cache: yes
```

Código 2.38 Actualización de los repositorios en YAML

Al tener los repositorios ya actualizados en el *host* cliente, es momento de instalar todos los paquetes necesarios para el funcionamiento del controlador así como también de la infraestructura, para esto se tienen que instalar los siguientes paquetes:

- Open JDK 7
- Java 8
- Maven 3.3.0
- Mininet
- Protocolo MP-TCP

Para realizar esta tarea en YAML, se tiene que utilizar la etiqueta apt utilizando el parámetro name y agregando cada uno de los paquetes que se desea instalar; además, se tiene que agregar un parámetro state: present para que se instale cada paquete de forma secuencial. En el Código 2.39 se presentan todas las tareas para la instalación de los paquetes necesarios para el funcionamiento de OpenDayLight.

```
- name: Instalacion Jdk 7
  apt:
    name: openjdk-7-jdk
    state: present

- name: Instalacion Java 8
  apt:
    name: oracle-java8-installer
    state: present

- name: Instalacion Maven3
  apt:
    name: maven3
    state: present

- name: Instalacion mininet
  apt:
    name: mininet
    state: present

- name: Instalacion del paquete MPTCP
  apt:
    name: linux-mptcp
    state: present
```

Código 2.39 Instalación de paquetes en YAML

Para poder utilizar Maven dentro de proyectos de OpenDayLight, es necesario realizar la copia del archivo settings.xml el cual se encuentra alojado en un repositorio externo. Sin el archivo settings.xml, al compilar el proyecto de OpenDayLight, no se van a encontrar los repositorios necesarios y se presentarán errores en la ejecución de Maven. Para realizar la tarea antes mencionada en YAML, es necesario utilizar la etiqueta shell como se indica en el Código 2.40.

```
- name: Agregar archivo settings.xml en .m2
  shell: cp -n ~/.m2/settings.xml{,.orig} ; wget -q -O -
  https://raw.githubusercontent.com/opendaylight/odlparent/master/settings.xml
  > ~/.m2/settings.xml
```

Código 2.40 Agregar el archivo settings.xml mediante YAML

En este punto es necesario ejecutar los comandos clean e install de Maven con el proyecto de OpenDayLight. En este caso, para realizar la tarea en YAML, lo primero que se tiene que realizar es cambiar el directorio por defecto, esto se realiza mediante la etiqueta args con el argumento de chdir, para especificar la nueva ruta: /root/MPTCP/bottleneck-Mptcp/, después de esto se utiliza la etiqueta shell con el comando mvn clean install, esto tomará un tiempo para realizar los cambios necesarios como también se descargarán los repositorios requeridos para el funcionamiento de OpenDayLight. En el Código 2.41 se presentan las tareas en Ansible para generar lo antes descrito.

```
- name: ejecutar maven clean install en la carpeta bottleneck-Mptcp
  shell: mvn clean install
  args:
    chdir: /root/MPTCP/bottleneck-Mptcp/
```

Código 2.41 Limpieza y compilación del proyecto en OpenDayLight con Maven en YAML

Como tarea final es necesario el reinicio del *host* cliente ya que cuando se instala el protocolo MP-TCP, el último paso obligatorio es el reinicio del *host* anfitrión, para que los cambios que se establecieron en el protocolo sean efectivos. En el Código 2.42 se puede observar la tarea descrita en YAML para reiniciar el *host* cliente.

```
- name: Reiniciar el Host Cliente
  command: /sbin/shutdown -r +1
  async: 0
  poll: 0
  ignore_errors: true
```

Código 2.42 Reinicio del servidor mediante YAML

El proceso del *playbook* anteriormente descrito se puede considerar como una función de red ya que facilita la instalación de forma automática de los paquetes necesarios para el

funcionamiento de la aplicación que solventa el problema de *shared bottleneck* de MP-TCP. Además, con el *playbook* de Ansible, no interviene el administrador de red para la instalación de ningún complemento para instalación y ejecución del controlador OpenDayLight, con esto se reducen los problemas que pueden ser introducidos por los usuarios.

Finalmente, el *playbook* de Ansible no necesita de una instalación adicional en los *host* clientes como demonios que se puedan comunicar con el servidor, solo necesita que se encuentre instalada la aplicación SSH y el lenguaje de programación Python 2.7 que viene por defecto en Ubuntu 14.4

3. ANÁLISIS DE RESULTADOS

En este capítulo se presentan las diferentes pruebas que se realizaron con la aplicación desarrollada empleando redes virtualizadas con distintas topologías. Además, se prueba el despliegue de la función de red mediante Ansible; se prueba la aplicación desarrollada con equipos físicos; y, finalmente se utiliza OpenStack para realizar pruebas con una infraestructura virtualizada en la nube.

3.1. Pruebas

La sección de pruebas se dividió en cinco partes, las cuales son: pruebas de la aplicación desarrollada; prueba de funcionamiento de la función de red virtualizada con Ansible; prueba de funcionamiento de OpenStack; pruebas con equipos físicos; y, pruebas de la aplicación sobre OpenStack.

3.1.1. Pruebas de la aplicación desarrollada para solventar el problema de shared bottleneck de MP-TCP

Para realizar las pruebas se utilizó un servidor Dell ProLiant DL320e Gen8 v2 en el que se instaló un hipervisor de tipo *bare-metal* llamado ESXI 6.0 de VMware. Dentro del hipervisor se instaló una máquina virtual con las características que se presentan en la Tabla 3.1.

Tabla 3.1 Características de la máquina virtual de pruebas

Característica	Especificación
Disco Duro	200 GB
Memoria RAM	32 GB
Numero de sockets virtuales	4

En la primera sección de pruebas se utilizaron cuatro escenarios en los que se configuró el protocolo MP-TCP para que cree diferente número de subflujos en una conexión MP-TCP, a continuación se especifican:

- Primer escenario: 3 subflujos (Figura 3.1),
- Segundo escenario: 4 subflujos (Figura 3.5),
- Tercer escenario: 7 subflujos (Figura 3.9); y,
- Cuarto escenario: 10 subflujos (Figura 3.13).
- Pruebas con nodos intermedios (Figura 3.18)
- Prueba con nodos aleatorios (Figura 3.22)

Para realizar las pruebas se utilizó un servidor FTP ya que la transferencia de información es constante en comparación con un servidor HTTP en donde la transferencia de información es de tipo ráfagas; los archivos utilizados en estas pruebas fueron de 10 MB, 50 MB, 500 MB y 2GB.

Para los distintos escenarios de pruebas se tomó en cuenta que los enlaces desde el *host* al switch tienen una velocidad de 200 Mbps, para el resto de enlaces la velocidad es de 20 Mbps.

En los cuatro escenarios propuestos se realizaron pruebas con el controlador OpenDayLight con el *feature* L2Switch y con la aplicación desarrollada que usa internamente el *feature* L2Switch.

3.1.1.1. Primer escenario: 3 subflujos

Para la prueba con el *feature* L2Switch se utilizó el escenario que se presenta en la Figura 3.1, donde se observa que existen un total de cinco switches OpenFlow conectados a OpenDayLight, creándose de esta manera tres posibles caminos (openflow:1-openflow:4-openflow:5; openflow:1-openflow:3-openflow:5 y openflow:1-openflow:2-openflow:5).

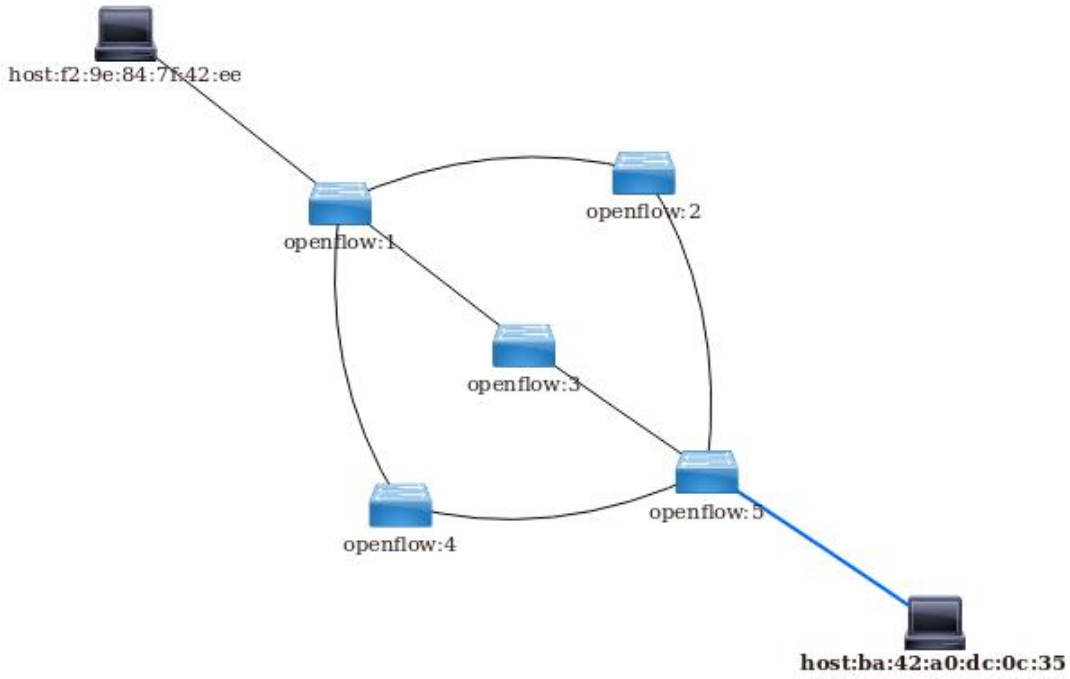

Figura 3.1 Primer escenario de prueba

En primera instancia se utilizó solamente el *feature* L2Switch, en este caso todos los paquetes tienden a transmitirse solo por un camino. Los resultados se los puede observar en la Figura 3.2.

Figura 3.2 Resultado de las pruebas primer escenario con L2Switch

En la Figura 3.2 se ve el resultado de las velocidades que se obtiene al utilizar una sola ruta ya que L2Switch elimina las posibles rutas al utilizar el *feature* loopRemover para evitar que se presenten lazos en la red.

En segunda instancia se utilizó la aplicación desarrollada cuyos resultados se pueden observar en la Figura 3.3, en donde se puede percibir que se aprovechan de mejor manera los recursos de red ya que las velocidades de transmisión de datos aumentan.

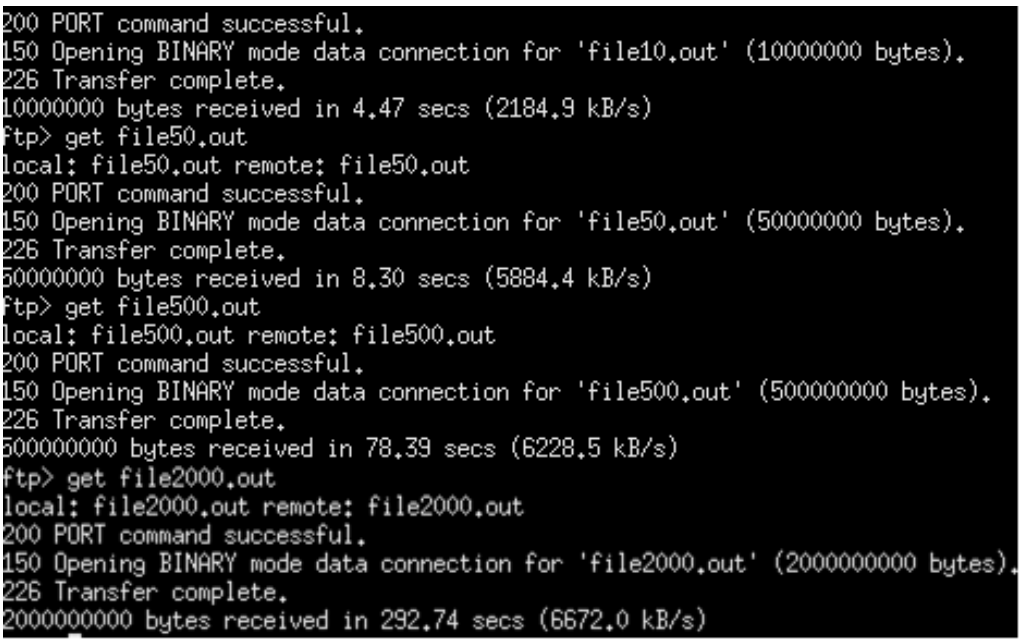

Figura 3.3 Resultados de las pruebas del primer escenario utilizando la aplicación desarrollada

En la Figura 3.4 se detallan los resultados obtenidos al utilizar la aplicación desarrollada en comparación con los valores obtenidos con L2Swtich.

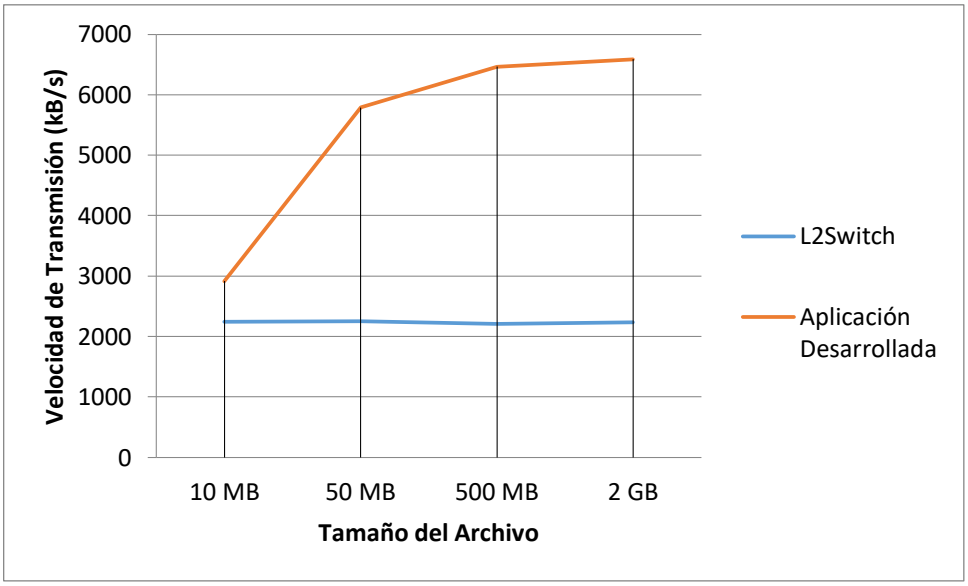

Figura 3.4 Resultados obtenidos en el primer escenario usando L2Switch y usando la aplicación desarrollada

De los resultados que se presenta en la Tabla 3.2 se puede observar que cuando los archivos que se transfieren mediante FTP son pequeños tanto la aplicación desarrollada como el *feature* L2Switch tienen la misma velocidad de transferencia como también el mismo tiempo, esto sucede ya que no se crean nuevos subflujos porque el tiempo es muy pequeño y solo se envía la información por una sola ruta.

Cuando se transfieren archivos mayores a 10 MB se puede observar que la velocidad y los tiempos de transferencia mejora cuando se utiliza la aplicación desarrollada a comparación de los resultados que se obtiene al utilizar L2Switch.

Tabla 3.2 Resultados de tiempos y velocidades obtenidas en las pruebas del *feature* L2Switch y la aplicación desarrollada con tres posibles rutas

	10MB		**50MB**		**500MB**		**2GB**	
	Velocidad (kB/s)	Tiempo (s)	Velocidad (kB/s)	Tiempo (s)	Velocidad (kB/s)	Tiempo (s)	Velocidad (kB/s)	Tiempo (s)
L2Switch	2244,6	4,35	2255,2	21,6	2204,9	221,4	2232,1	882,3
Aplicación desarrollada	2918,1	3,486	5789,4	8,476	6463,2	75,576	6587,2	296,84

3.1.1.2. Segundo escenario: 4 subflujos

En la Figura 3.5 se puede observar el segundo escenario de pruebas donde se agrega un camino adicional (openflow:5-openflow:6-openflow:1) en comparación del primer escenario.

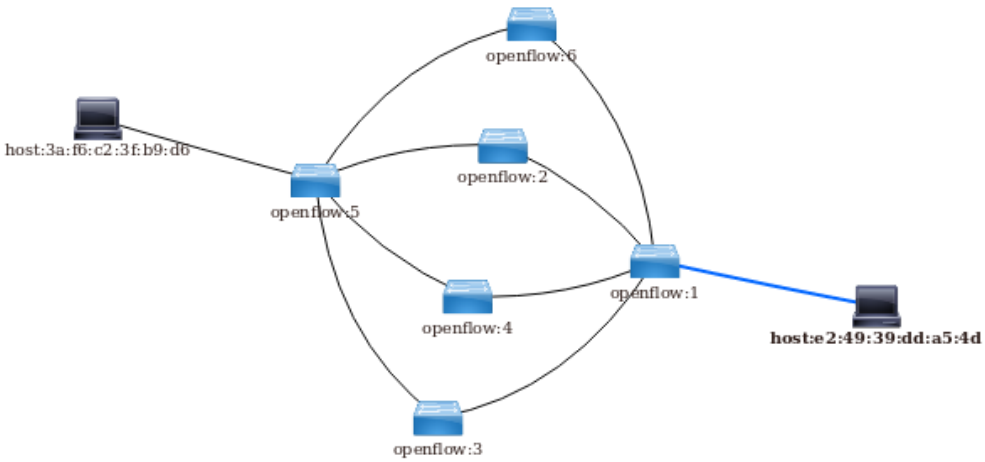

Figura 3.5 Segundo escenario de prueba

Como resultado de la transmisión de los diferentes archivos al servidor FTP en el segundo escenario, utilizando el *feature* L2Switch, se obtuvieron los resultados de tiempos y velocidades de transmisión que se detallan en la Figura 3.6.

```
ftp> get file10.out
local: file10.out remote: file10.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file10.out' (10000000 bytes).
226 Transfer complete.
10000000 bytes received in 4.32 secs (2263.1 kB/s)
ftp> get file50.out
local: file50.out remote: file50.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file50.out' (50000000 bytes).
226 Transfer complete.
50000000 bytes received in 21.62 secs (2258.6 kB/s)
ftp> get file500.out
local: file500.out remote: file500.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file500.out' (500000000 bytes).
226 Transfer complete.
500000000 bytes received in 214.03 secs (2281.4 kB/s)
ftp> get file2000.out
local: file2000.out remote: file2000.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file2000.out' (2000000000 bytes).
226 Transfer complete.
2000000000 bytes received in 880.91 secs (2217.2 kB/s)
```

Figura 3.6 Resultados de las pruebas del segundo escenario utilizando L2Switch

En la Figura 3.7 se pueden observar los resultados obtenidos al utilizar la aplicación desarrollada en el segundo escenario, se puede observar que tanto las velocidades de transmisión así como también los tiempos mejoran notablemente ya que se incrementa un camino adicional a la infraestructura de red.

```
ftp> get file10.out
local: file10.out remote: file10.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file10.out' (10000000 bytes).
226 Transfer complete.
10000000 bytes received in 3.45 secs (2827.4 kB/s)
ftp> get file500.out
local: file500.out remote: file500.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file500.out' (500000000 bytes).
226 Transfer complete.
500000000 bytes received in 56.68 secs (8614.0 kB/s)
ftp> get file50.out
local: file50.out remote: file50.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file50.out' (50000000 bytes).
226 Transfer complete.
50000000 bytes received in 7.29 secs (6696.1 kB/s)
ftp> get file2000.out
local: file2000.out remote: file2000.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file2000.out' (2000000000 bytes).
226 Transfer complete.
2000000000 bytes received in 215.14 secs (9078.5 kB/s)
```

Figura 3.7 Resultados de las pruebas del segundo escenario utilizando la aplicación desarrollada

En la Figura 3.8 se pueden observar los resultados que se obtuvieron tanto con la aplicación desarrollada como con L2Switch.

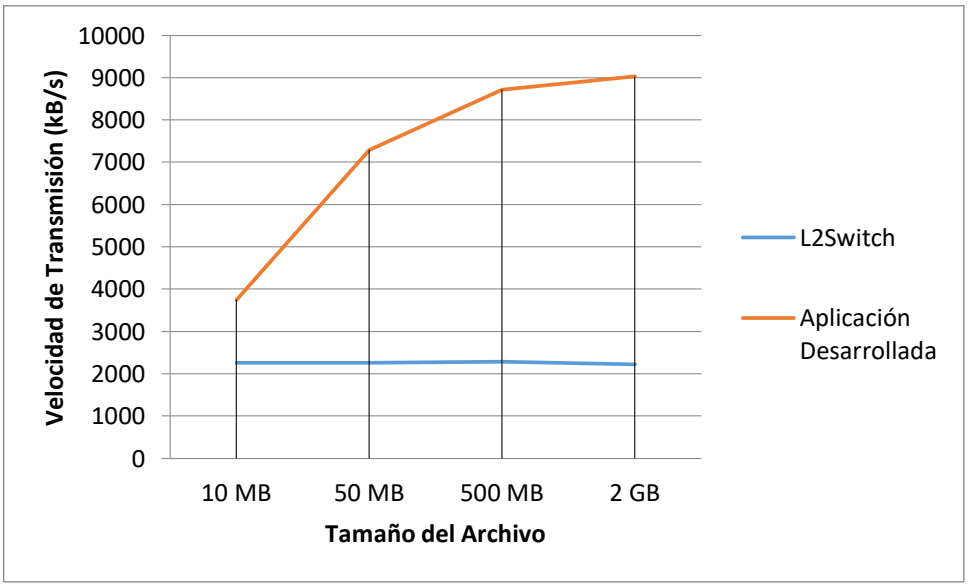

Figura 3.8 Resultados obtenidos por L2Switch y la aplicación desarrollada en el segundo escenario

En la Tabla 3.3 se observan los resultados que se obtienen al utilizar tanto la aplicación desarrollada como el *feature* L2Switch. De los resultados que se obtienen se puede observar que en todos los archivos que se transfieren con FTP mediante la utilización de la aplicación desarrollada se consiguen un mejores tiempos de transferencia como también mejores velocidades a comparación de los resultados que se obtiene con el *feature* L2Switch. Los resultados que se obtienen en la Tabla 3.3 se los puede comparar

con los resultados de la Tabla 3.2 en el que se tenía solo 3 posibles rutas, en esta nueva prueba se agregó una nueva ruta y en los resultados que se lograron se pudo observar que se mejoró las velocidades y los tiempos de transferencia excepto en la prueba de 10 MB el cual tiene resultados similares.

Tabla 3.3 Resultados de tiempos y velocidades obtenidas en las pruebas del *feature* L2Switch y la aplicación desarrollada con cuatro posibles rutas

	10MB		**50MB**		**500MB**		**2GB**	
	Velocidad (kB/s)	Tiempo (s)	Velocidad (kB/s)	Tiempo (s)	Velocidad (kB/s)	Tiempo (s)	Velocidad (kB/s)	Tiempo (s)
L2Switch	2263,1	4,32	2258,6	21,62	2281,4	214,03	2217,2	880,91
Aplicación desarrollada	3748,3	2,82	7294,5	6,706	8720,4	55,99	9025,9	216,4

3.1.1.3. Tercer escenario: 7 subflujos

En la Figura 3.9 se describe el tercer escenario de prueba en el que se añaden cuatro posibles caminos, tomando como base el primer escenario (openflow:5-openflow:6-openflow:1, openflow:5-openflow:7-openflow:1, openflow:5-openflow:8-openflow:1 y openflow:5-openflow:9-openflow:1).

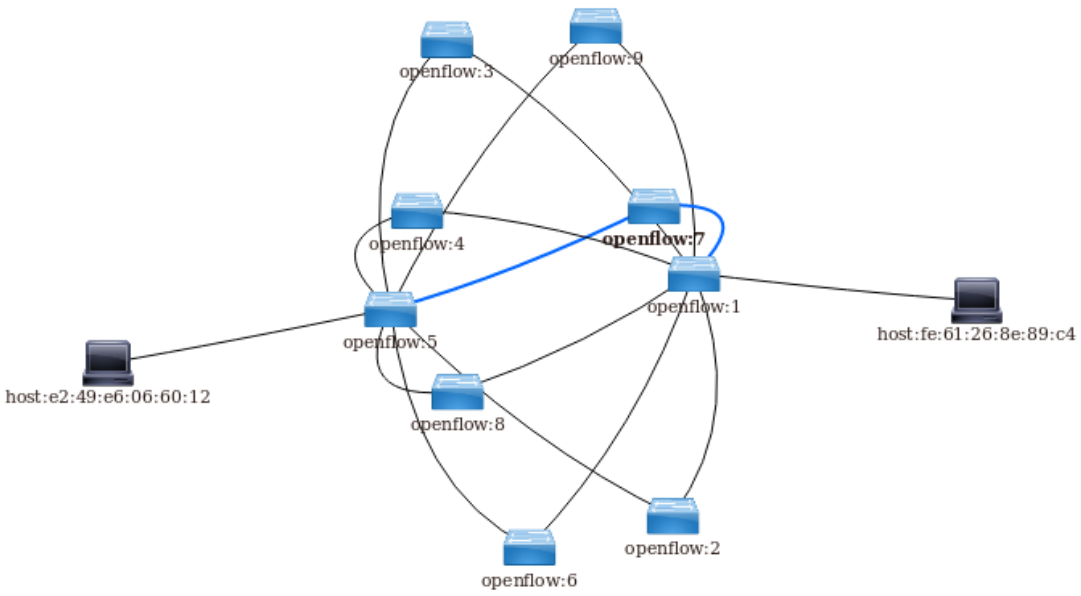

Figura 3.9 Tercer escenario de prueba

En la Figura 3.10 se puede observar el resultado de la prueba del tercer escenario utilizando el *feature* L2Switch.

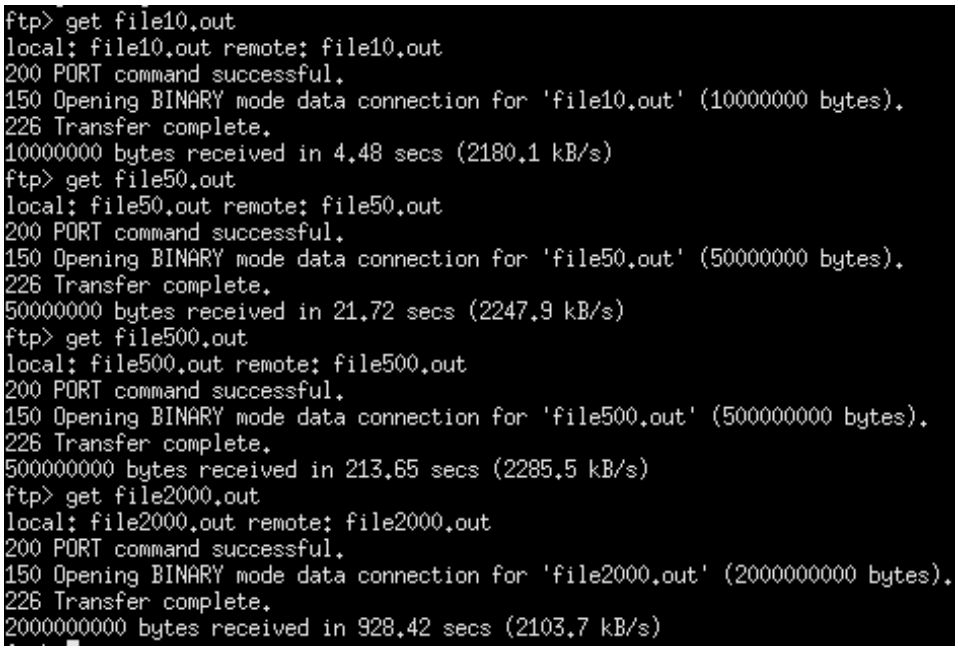

Figura 3.10 Resultados de las pruebas del tercer escenario con L2Switch

En la Figura 3.11 se pueden observar los resultados de las pruebas realizadas con la aplicación desarrollada, donde se puede apreciar un aumento en la velocidad de transmisión de datos.

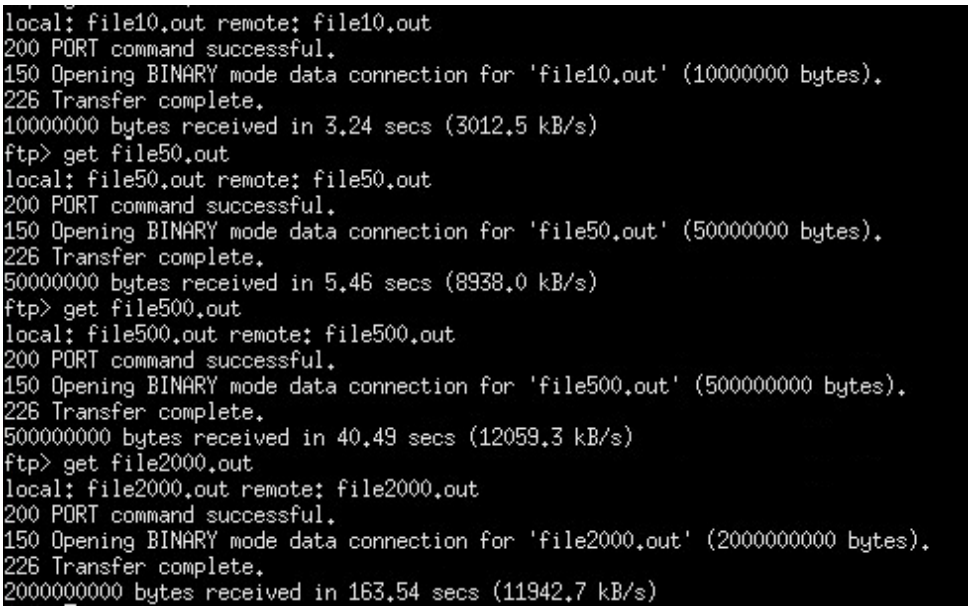

Figura 3.11 Resultados de las pruebas del tercer escenario con la aplicación desarrollada

En la Figura 3.12 se puede observar dos curvas en la que se compara los resultados obtenidos con L2Switch y la aplicación desarrollada.

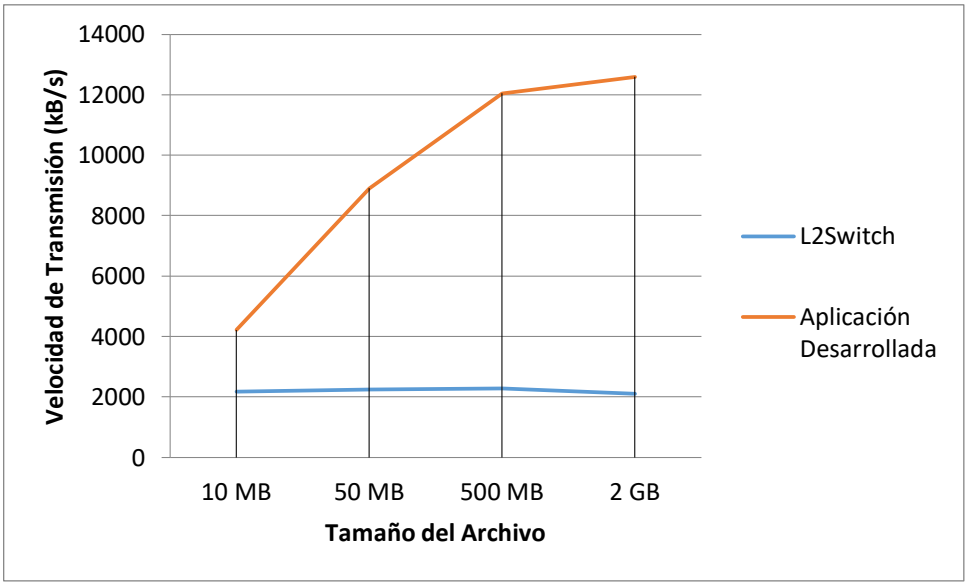

Figura 3.12 Resultados obtenidos en el tercer escenario por L2Switch y la aplicación desarrollada

En la Tabla 3.4 se presentan los resultados que se obtienen al transferir mediante FTP cuatro archivos de diferentes tamaños, esto se lo realiza mediante el feature L2Switch y también con la aplicación desarrollada para poder comparar el uno con el otro. De los resultados obtenidos se puede observar que la aplicación desarrollada tiene un mejor performance tanto en los tiempos como en las velocidades de transferencia a comparación del *feature* L2Switch. Esta prueba se realizó con siete posibles rutas, a comparación de las anteriores pruebas se puede observar que tanto las velocidades como los tiempos de transferencia mejoran notablemente.

Tabla 3.4 Resultados de tiempos y velocidades obtenidas en las pruebas del *feature* L2Switch y la aplicación desarrollada con siete posibles rutas

	10MB		**50MB**		**500MB**		**2GB**	
	Velocidad (kB/s)	Tiempo (s)	Velocidad (kB/s)	Tiempo (s)	Velocidad (kB/s)	Tiempo (s)	Velocidad (kB/s)	Tiempo (s)
L2Switch	2180,1	4,48	2247,9	21,72	2285,5	213,65	2103,7	928,42
Aplicación desarrollada	4222,1	2,472	8900,7	5,49	12045	40,454	12589	155,39

Existe una excepción en la prueba realizada, cuando se transfiere 2GB la velocidad y el tiempo no mejora a comparación de la transferencia del archivo de 500MB lo cual no sigue con la tendencia de velocidades y tiempos de transferencias.

Este problema se puede presentar cuando uno de los subflujos que se establecen mediante los mensajes MP_JOIN se cierra inesperadamente, con este evento lo que sucede es que se reduce el performance tanto de la infraestructura de red como también con la aplicación desarrollada y se obtienen los tiempos y velocidades de transmisión de la Tabla 3.4.

3.1.1.4. Cuarto escenario: 10 subflujos

En la Figura 3.13 se presenta el cuarto escenario de pruebas donde existe un total de diez rutas posibles, tres adicionales rutas adicionales que el escenario anterior (openflow:5-openflow:18-openflow:1, openflow:5-openflow:17-openflow:1 y openflow:5-openflow:16-openflow:1).

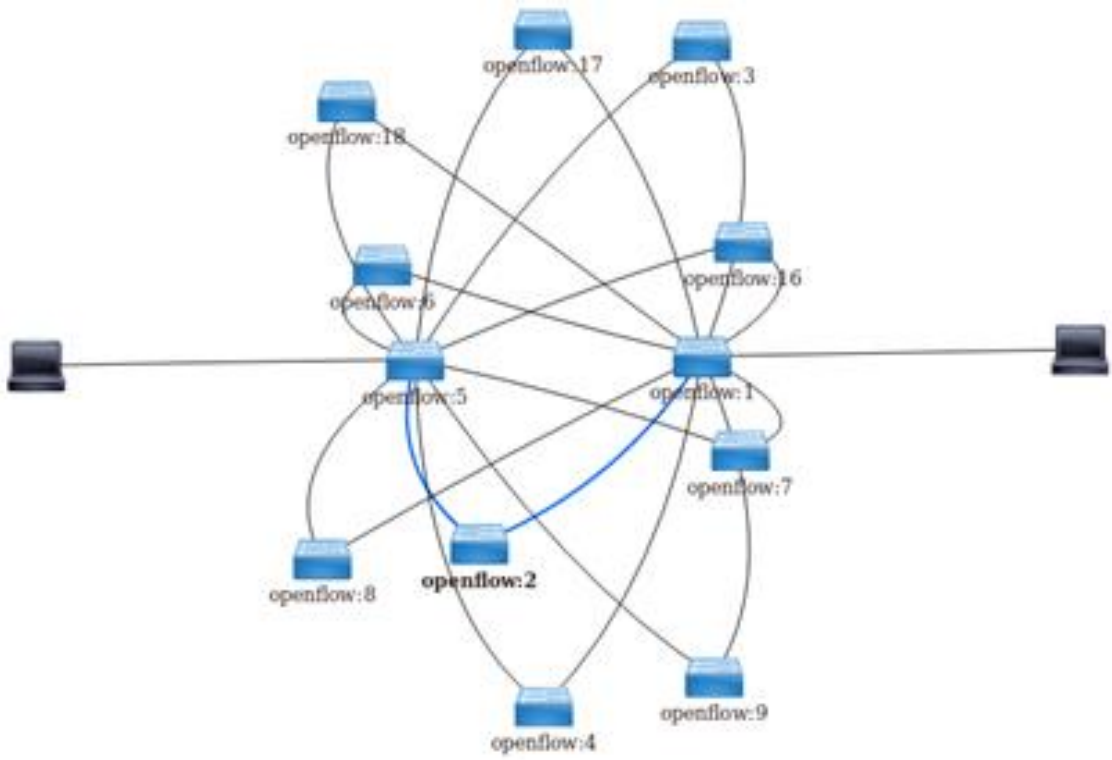

Figura 3.13 Cuarto escenario de prueba

En la Figura 3.14 se detallan las pruebas realizadas sobre el cuarto escenario mediante L2Switch, donde se observa que la velocidad de transmisión de datos se mantiene constante.

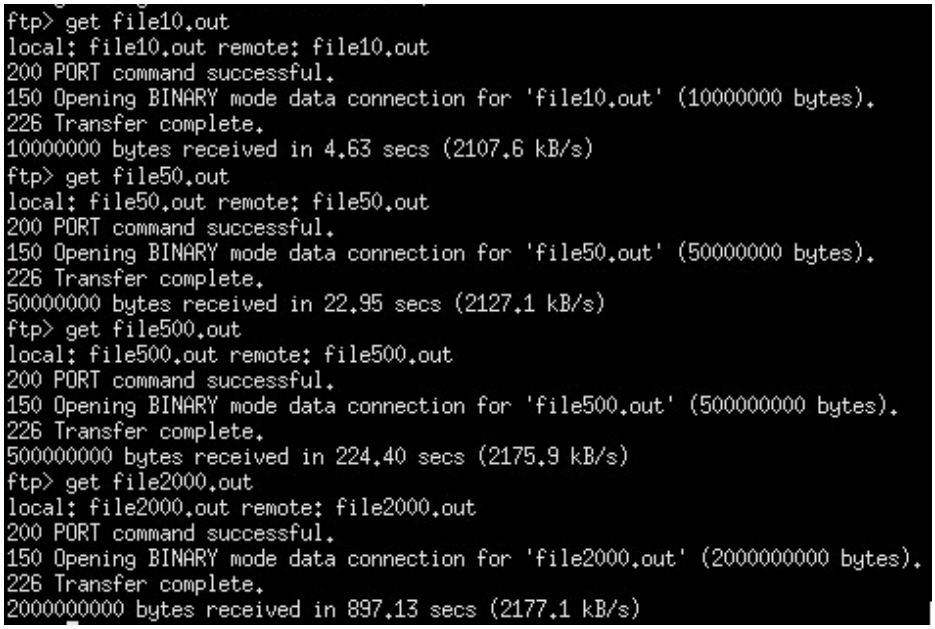

Figura 3.14 Resultados de las pruebas del cuarto escenario con L2Switch

En la Figura 3.15 se pueden observar las pruebas realizadas al cuarto escenario con la aplicación desarrollada, con lo que se puede concluir que la velocidad de transmisión se incrementa sustancialmente en comparación de las pruebas realizadas con L2Switch.

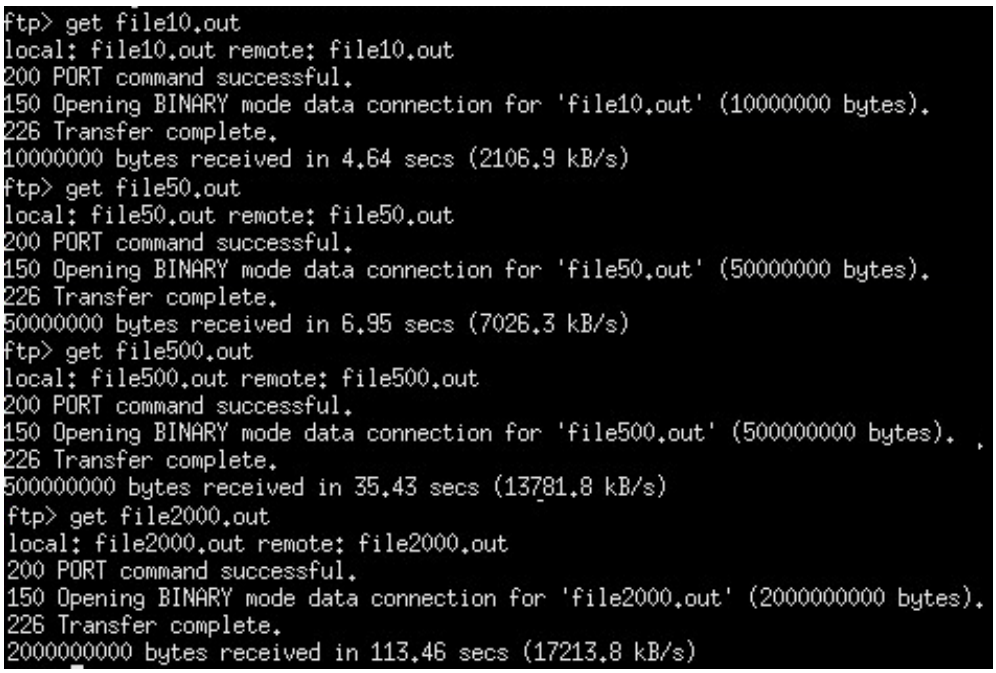

Figura 3.15 Resultados de las pruebas del cuarto escenario con la aplicación desarrollada

En la Figura 3.16 se presenta la gráfica obtenida del último escenario que se implementó donde se comparan las velocidades obtenidas tanto con L2Switch así como con la aplicación desarrollada.

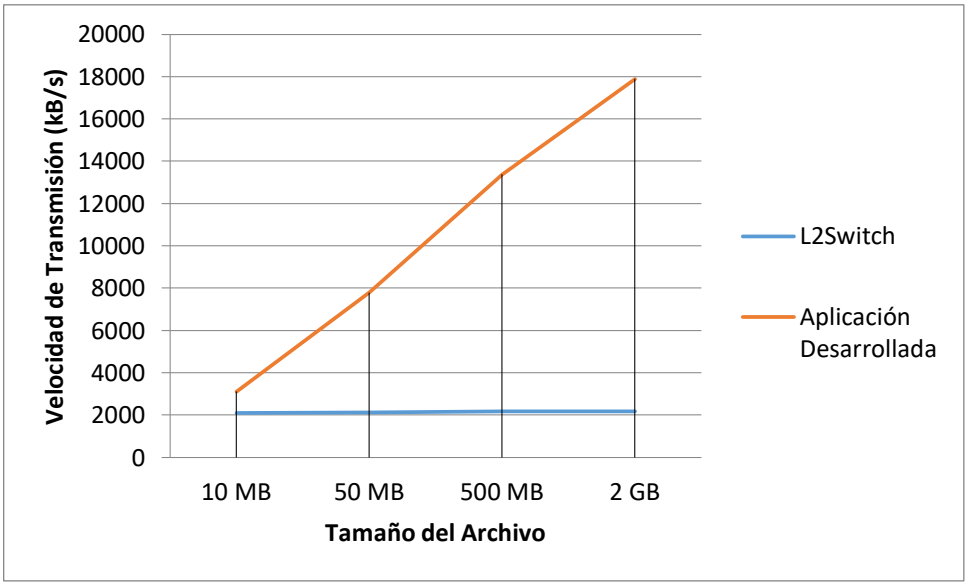

Figura 3.16 Resultados obtenidos en el cuarto escenario por L2Switch y la aplicación desarrollada

En la Tabla 3.5 se puede observar los resultados que se obtuvieron cuando se utilizan 10 posibles caminos para la transferencia de cuatro archivos de diferente tamaño mediante FTP utilizando primero el *feature* L2Switch y después la aplicación desarrollada. Con los resultados que se encuentra en la Tabla 3.5 se puede comparar con los resultados que se obtuvieron en las anteriores pruebas con tres, cuatro y siete posibles rutas donde se observa que la aplicación desarrollada mejora cada vez que se agrega una posible ruta aumentando el performance de la aplicación. También de la Tabla 3.5 se puede observar que cuando el archivo es pequeño (10MB) tanto la aplicación como el *feature* L2Switch tiene las mismas velocidades como también el mismo tiempo.

Tabla 3.5 Resultados de tiempos y velocidades obtenidas en las pruebas del *feature* L2Switch y la aplicación desarrollada con diez posibles rutas

	10MB		**50MB**		**500MB**		**2GB**	
	Velocidad (kB/s)	Tiempo (s)	Velocidad (kB/s)	Tiempo (s)	Velocidad (kB/s)	Tiempo (s)	Velocidad (kB/s)	Tiempo (s)
L2Switch	2107,6	4,63	2127,1	22,95	2175,9	224,4	2177,1	897,13
Aplicación desarrollada	3107,3	4,5	7787,3	6,9	13377	35,5	17889	114,3

En la Tabla 3.6 y en la Tabla 3.7 se pueden observar los resultados obtenidos de las pruebas realizadas con el *feature* L2Switch y con la aplicación desarrollada, respectivamente. Se puede observar en la Tabla 3.6, que usando solo L2Switch, las velocidades de transmisión de datos son prácticamente constantes, al variar el número de caminos y el tamaño de los archivos; por otro lado, en la Tabla 3.7 se observa que las velocidades de transmisión se incrementan con el incremento del número de posibles caminos por los que se pueden transmitir los datos. También se observa que cuando se varia el tamaño de los archivos y se utiliza la aplicación desarrollada entre más grande es el archivo que se va a transferir mejor *performance* tiene la aplicación. Existe una excepción si se utiliza archivos de tamaño pequeño, como es el caso de 10 MB, en la cual se tiene resultados similares tanto en la aplicación desarrollada como con el *feature* L2Switch, esto sucede ya que el tiempo es muy pequeño para establecer posibles rutas mediante los mensajes MP_JOIN.

Tabla 3.6 Resultados obtenidos en los diferentes escenarios de pruebas empleando L2Switch

Escenario	**10MB**	**50MB**	**500MB**	**2GB**
Escenario 1 (3 caminos)	2244,6 kB/s	2255,2 kB/s	2204,9 kB/s	2252,1 kB/s
Escenario 2 (4 caminos)	2263,1 kB/s	2258,6 kB/s	2281,4 kB/s	2217,2 kB/s
Escenario 3 (7 caminos)	2180,1 kB/s	2247,9 kB/s	2285,5 kB/s	2103,7 kB/s
Escenario 4 (10 caminos)	2107,6 kB/s	2127,1 kB/s	2175,9 kB/s	2177,1 kB/s

Tabla 3.7 Resultados obtenidos en los diferentes escenarios de pruebas con la aplicación desarrollada

Escenario	**10MB**	**50MB**	**500MB**	**2GB**
Escenario 1 (3 caminos)	2918,1 kB/s	5789,38 kB/s	6463,22 kB/s	6587,21 kB/s
Escenario 2 (4 caminos)	3748,3 kB/s	7294,5 kB/s	8720,44 kB/s	9025,9 kB/s
Escenario 3 (7 caminos)	4222,1 kB/s	8900,7 kB/s	12044,7 kB/s	12589,1 kB/s
Escenario 4 (10 caminos)	3107,3 kB/s	7787,3 kB/s	13377,4 kB/s	17889,5 kB/s

En la Figura 3.17 se pueden observar las gráficas que se obtienen utilizando la aplicación desarrollada con los diferentes escenarios de pruebas, además se observa como mejora la velocidad cuando se aumenta el número de caminos en la topología de red.

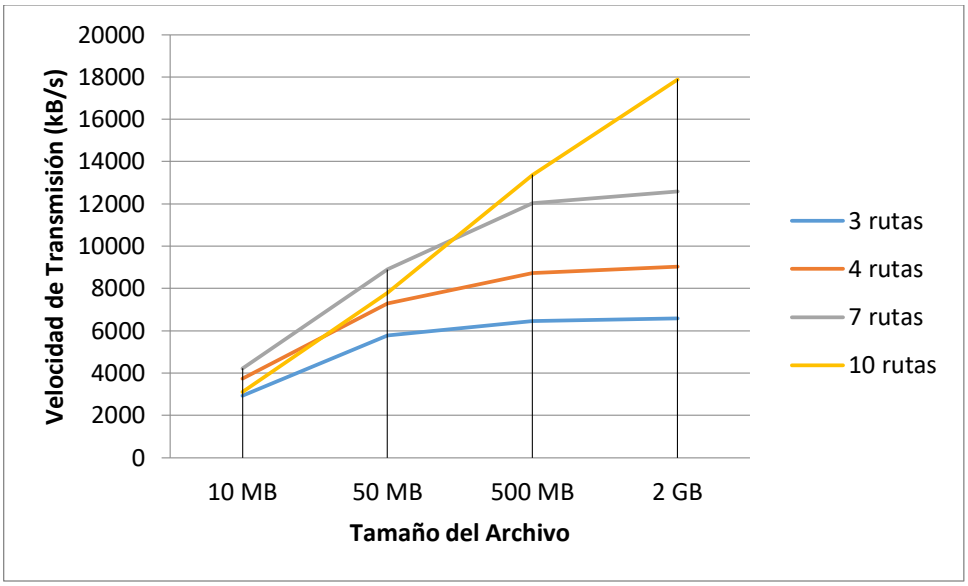

Figura 3.17 Resultados obtenidos de todas las pruebas realizadas en los diferentes escenarios por la aplicación desarrollada

Además en la Figura 3.17 se puede observar que existe una inconsistencia cuando se transmite un archivo de 50MB cuando se tienen siete y diez posibles rutas se puede observar que la velocidad con siete rutas es mayor a la que se obtiene cuando se tiene diez rutas esto se puede explicar ya que el tiempo que dura la transferencia de los archivos es pequeño y no se establecen las diez posibles rutas, pero si se envían los mensajes MP_JOIN disminuyendo el *performance* de la aplicación desarrollada.

3.1.1.5. Prueba con nodos intermedios

Como una prueba adicional se empleó una topología en la que se agregaron nodos intermedios ubicados entre el cuello de botella compartido y el *host* tanto transmisor como receptor y se configuró el protocolo MP-TCP para que cree 3 subflujos en cada conexión. En la Figura 3.18 se observan los nodos intermedios que, en este caso, son openflow:6 y openflow:7.

En la Figura 3.19 se pueden observar las pruebas que se realizaron con el *feature* L2Switch se obtiene una conexión extremo a extremo. En la Figura 3.20 se realizaron pruebas con la aplicación desarrollada donde se puede observar que primero existe conectividad extremo a extremo y, adicionalmente, se observa que se obtienen las ventajas de la aplicación desarrollada con el aumento en la velocidad de transmisión.

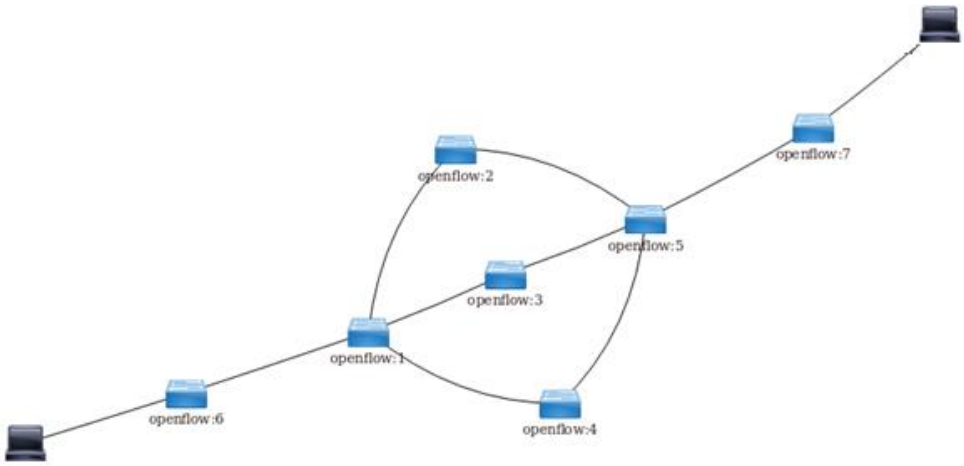

Figura 3.18 Quinto escenario de prueba

```
ftp> get file10.out
local: file10.out remote: file10.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file10.out' (10000000 bytes).
226 Transfer complete.
10000000 bytes received in 4.45 secs (2196.9 kB/s)
ftp> get file50.out
local: file50.out remote: file50.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file50.out' (50000000 bytes).
226 Transfer complete.
50000000 bytes received in 22.19 secs (2200.5 kB/s)
ftp> get file500.out
local: file500.out remote: file500.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file500.out' (500000000 bytes).
226 Transfer complete.
500000000 bytes received in 221.08 secs (2208.6 kB/s)
```

Figura 3.19 Resultados de las pruebas del quinto escenario con L2Switch

```
ftp> get file10.out
local: file10.out remote: file10.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file10.out' (10000000 bytes).
226 Transfer complete.
10000000 bytes received in 2.93 secs (3330.5 kB/s)
ftp> get file50.out
local: file50.out remote: file50.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file50.out' (50000000 bytes).
226 Transfer complete.
50000000 bytes received in 10.22 secs (4779.3 kB/s)
ftp> get file500.out
local: file500.out remote: file500.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file500.out' (500000000 bytes).
226 Transfer complete.
500000000 bytes received in 78.07 secs (6254.7 kB/s)
```

Figura 3.20 Resultados de las pruebas del quinto escenario con la aplicación desarrollada

En la Figura 3.21 se puede observar los resultados que se obtienen con la transferencia de los diferentes archivos de prueba. Los resultados del *feature* L2Switch se puede

observar que es una línea constante pero con la aplicación desarrollada los resultados mejoran constantemente.

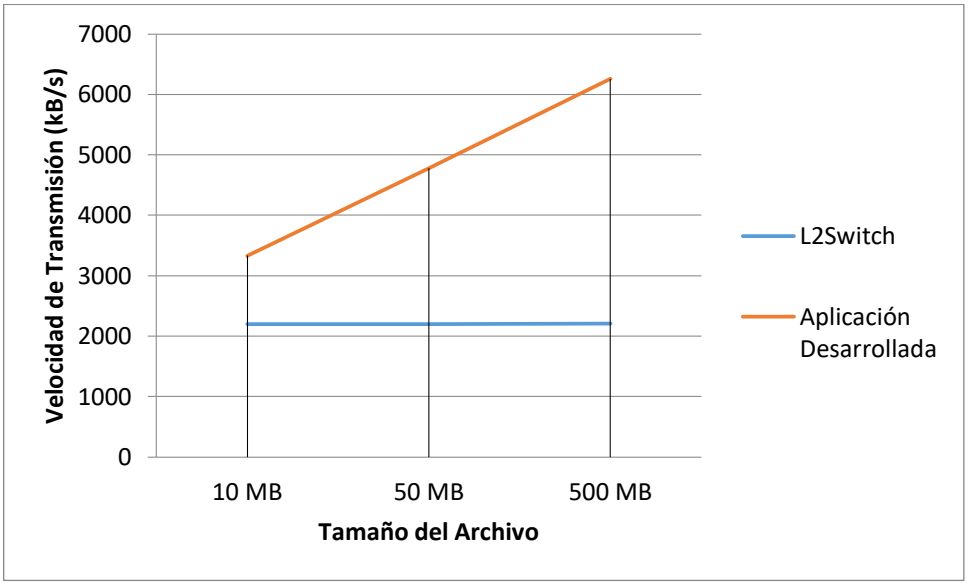

Figura 3.21 Resultados obtenidos en el quinto escenario con nodos intermedios por L2Switch y la aplicación desarrollada

3.1.1.6. Prueba aleatoria con varios nodos

En la Figura 3.22 se presenta un escenario donde se crearon múltiples caminos para monitorear el comportamiento de la aplicación desarrollada. Para este escenario se configuró MP-TCP para que se pueda establecer tres y siete subflujos tanto en el *host* cliente así como en el *host* servidor y los tamaños de archivos fueron de 10MB, 50MB y 500MB.

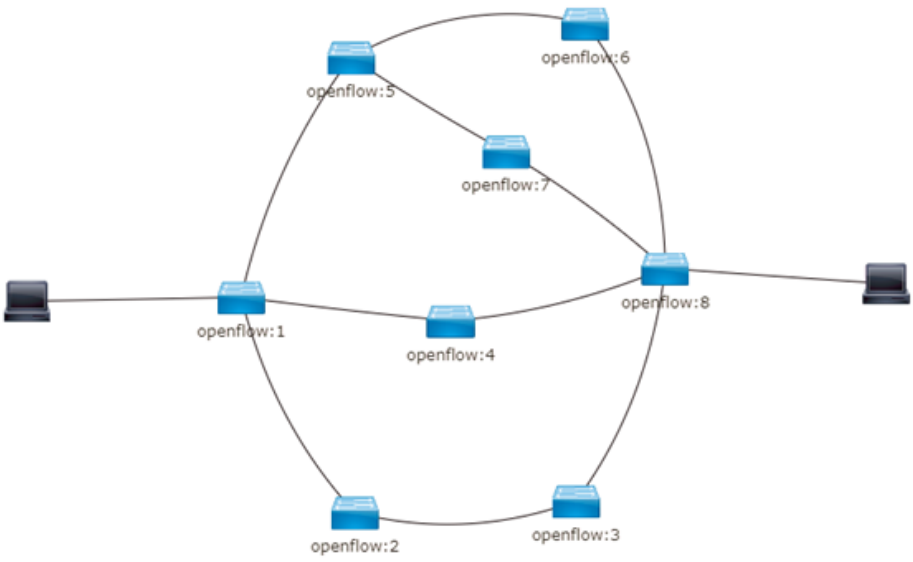

Figura 3.22 Sexto escenario de prueba

Cuando en las pruebas se configuran tres subflujos entre el *host* cliente y servidor, al utilizar el *feature* L2Switch se obtienen los resultados que se presentan en la Figura 3.23. Los resultados que se obtienen al utilizar la aplicación desarrollada se presentan en la Figura 3.24, se puede observar que cuando se utiliza la aplicación desarrollada aumenta la velocidad de transmisión ya que se aprovechan los tres caminos que dispone la red y fueron configurados tanto en el cliente como en el servidor.

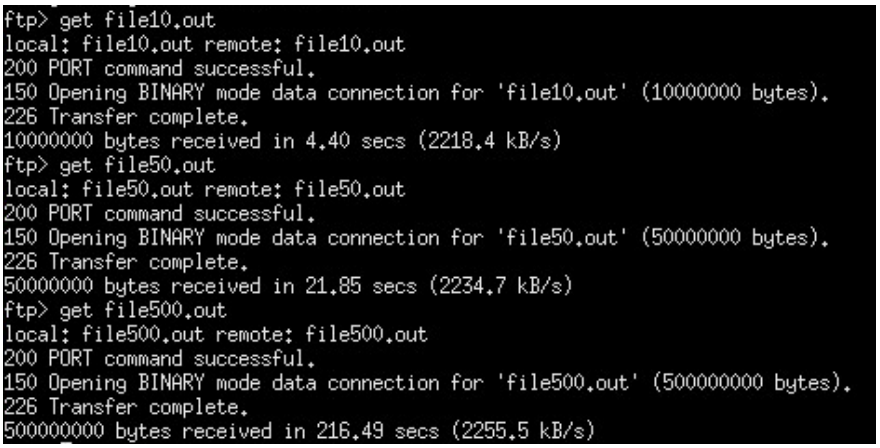

Figura 3.23 Resultados de las pruebas del sexto escenario con el *feature* L2Switch configurando los *hosts* para crear 3 subflujos

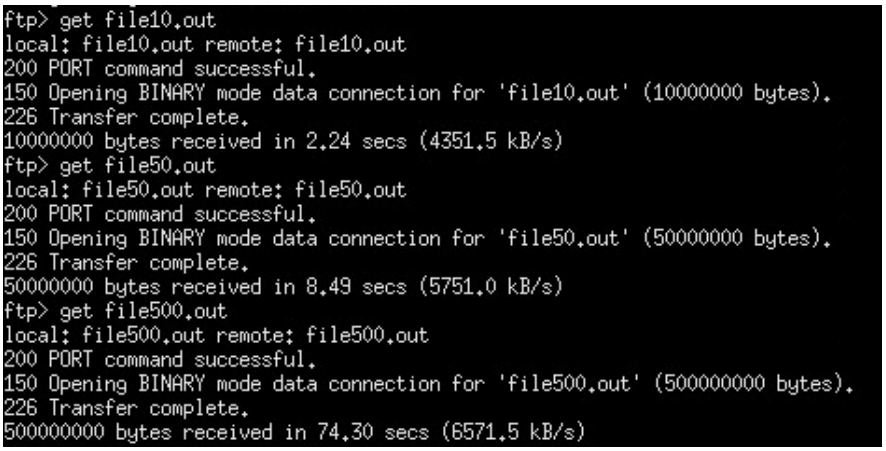

Figura 3.24 Resultados de las pruebas del sexto escenario con la aplicación desarrollada configurado en los *hosts* para crear 3 subflujos

En la Figura 3.25 se puede observar los resultados que se obtuvieron al utilizar tanto la aplicación como también la aplicación desarrollada para la transferencia de archivos con diferente tamaño utilizando FTP. Se puede observar que de los resultados obtenidos que la aplicación desarrollada tiene un mejor performance que el *feature* L2Switch ya que este se mantiene constante en todas las pruebas a diferencia de la aplicación desarrollada que aumenta la velocidad y disminuye los tiempos de trasferencia con lo que se mejora el performance de la red.

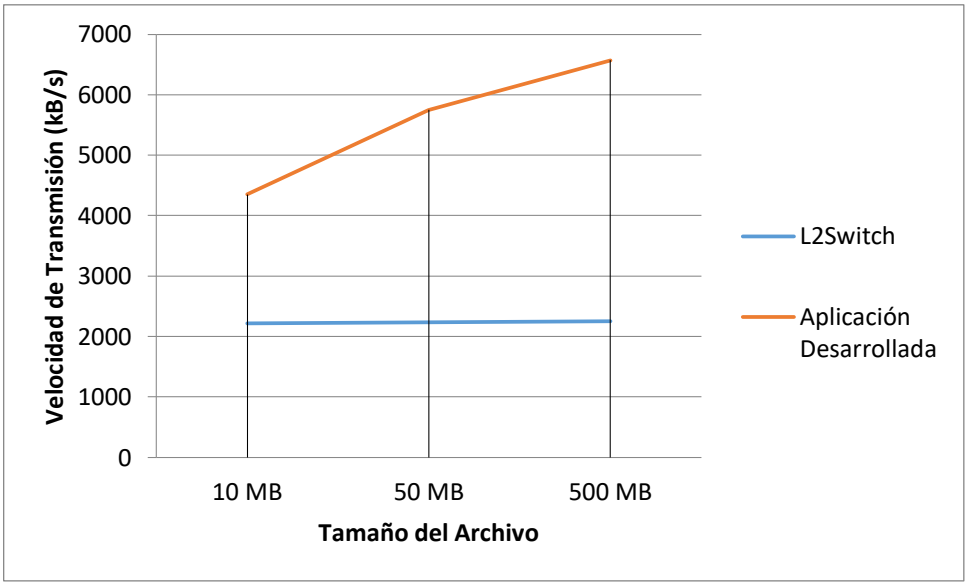

Figura 3.25 Resultados obtenidos en el sexto escenario configurado con tres subflujos por L2Switch y la aplicación desarrollada

Al configurar MP-TCP en los *hosts* para que se establezcan siete subflujos los resultados que se obtuvieron al utilizar la aplicación desarrollada se pueden observar en la Figura 3.27, mientras que los resultados que se obtuvieron al utilizar el *feature* L2Switch se presentan en la Figura 3.26.

En el sexto escenario que se presentó en la Figura 3.22 en el que existen tres posibles caminos, como primer resultado se obtuvo que solo se pueda asignar el máximo de rutas que la topología de red dispone, como sucedió con la configuración de MP-TCP con siete subflujos de los cuales cuatro no se utilizaron (ultimo escenario descrito).

```
ftp> get file10.out
local: file10.out remote: file10.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file10.out' (10000000 bytes).
226 Transfer complete.
10000000 bytes received in 4.36 secs (2240.1 kB/s)
ftp> get file50.out
local: file50.out remote: file50.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file50.out' (50000000 bytes).
226 Transfer complete.
50000000 bytes received in 21.60 secs (2260.9 kB/s)
ftp> get file500.out
local: file500.out remote: file500.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file500.out' (500000000 bytes).
226 Transfer complete.
500000000 bytes received in 216.76 secs (2252.7 kB/s)
```

Figura 3.26 Resultados de las pruebas del sexto escenario con L2Switch configurado en los *hosts* para crear 7 subflujos

```
ftp> get file10.out
local: file10.out remote: file10.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file10.out' (10000000 bytes).
226 Transfer complete.
10000000 bytes received in 2.32 secs (4215.7 kB/s)
ftp> get file50.out
local: file50.out remote: file50.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file50.out' (50000000 bytes).
226 Transfer complete.
50000000 bytes received in 8.30 secs (5883.7 kB/s)
ftp> get file500.out
local: file500.out remote: file500.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file500.out' (500000000 bytes).
226 Transfer complete.
500000000 bytes received in 75.12 secs (6500.2 kB/s)
```

Figura 3.27 Resultados de las pruebas del sexto escenario con la aplicación desarrollada configurando los *hosts* para crear 7 subflujos

En la Figura 3.28 se observan los resultados que se consiguieron al utilizar la infraestructura de red del escenario seis, cuando se configura tanto en el *host* cliente como en el servidor para que se establezcan siete posibles caminos.

Con este escenario se utilizó la aplicación desarrollada como también el *feature* L2Switch, donde mejores resultados se obtiene con la aplicación desarrollada.

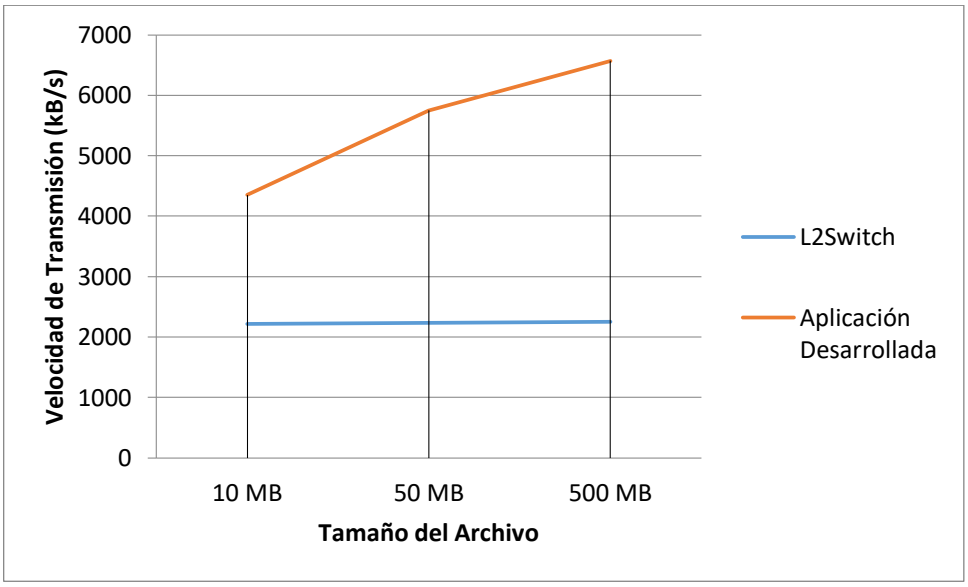

Figura 3.28 Resultados obtenidos en el sexto escenario configurado con siete subflujos por L2Switch y la aplicación desarrollada

Como una prueba final se presenta la topología de red de la Figura 3.29 la cual se encuentra en funcionamiento [70]. La topología original de red consta con veintiún

elementos de red, para la topología de pruebas se va a utilizar solo quince switch mediante Mininet.

Figura 3.29 Topología de red de AboveNet [70]

En la Figura 3.30 se puede observar la topología de red creada mediante Mininet en la que intervienen un total de quince switches y dos *hosts* denominados H1 y H2.

De la topología que se presenta en la Figura 3.30 se puede observar que existe una topología con varias posibles rutas que puede seleccionar..

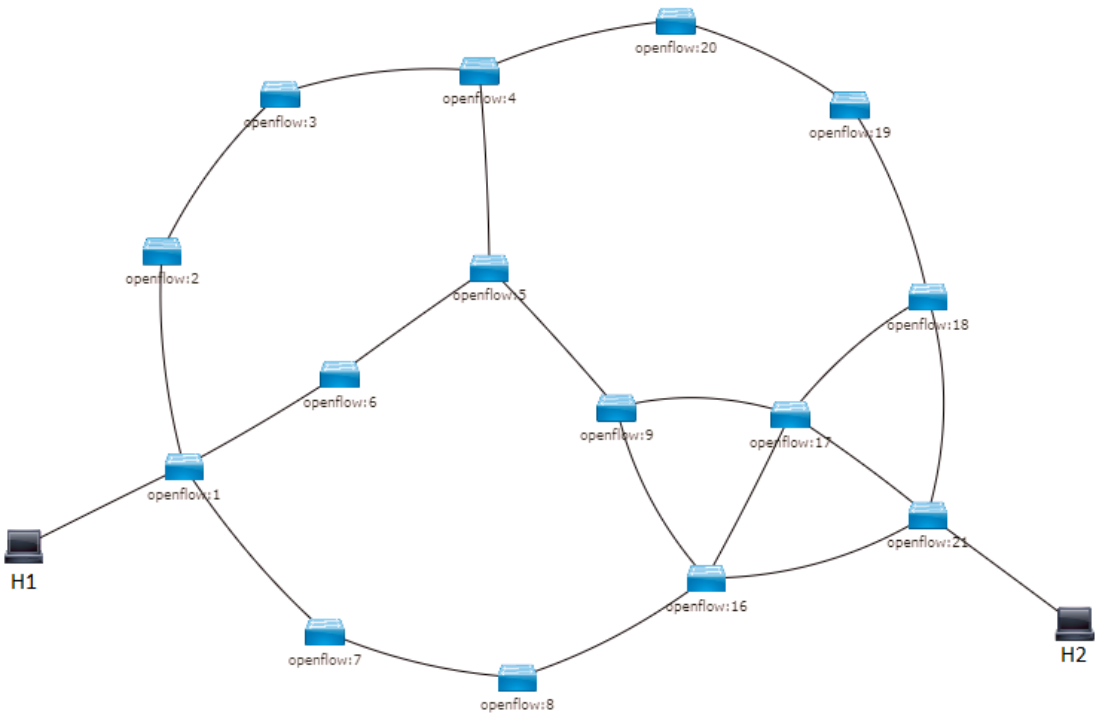

Figura 3.30 Topología de AboveNet con OpenVSwitch

En la Tabla 3.8 se observan los resultados que se obtuvieron mediante la transferencia de cuatro archivos con diferentes tamaños mediante el protocolo FTP.

También se puede observar de los resultados obtenidos que la aplicación desarrollada tiene mejores resultados en velocidad de transferencia que el *feature* L2Switch ya que este último solo utiliza un posible camino a diferencia que la aplicación desarrollada utiliza todas las posibles rutas de la topología.

Tabla 3.8 Resultados de velocidad obtenidos en la topología de AboveNet con el *feature* L2Switch y con la aplicación desarrollada

	10MB	**50MB**	**500MB**	**2GB**
L2Switch	2286,6 kB/s	2115,7 kB/s	2191,7 kB/s	2209,9 kB/s
Aplicación Desarrollada	2513,3 kB/s	4436,7 kB/s	5848,6 kB/s	6003,6 kB/s

En Figura 3.31 se puede observar los resultados que se obtienen en la topología AboveNet, cuando se transfiere un paquete de tamaño de 10MB el tiempo que se demora en transferir el mismo es pequeño y no se establece los subflujos, por esta razón la velocidad que se obtiene tanto con el *feature* L2Switch como la aplicación desarrollada es la misma.

Para los archivos restantes las velocidades que se obtienen con la aplicación desarrollada es superior a comparación del *feature* L2Switch.

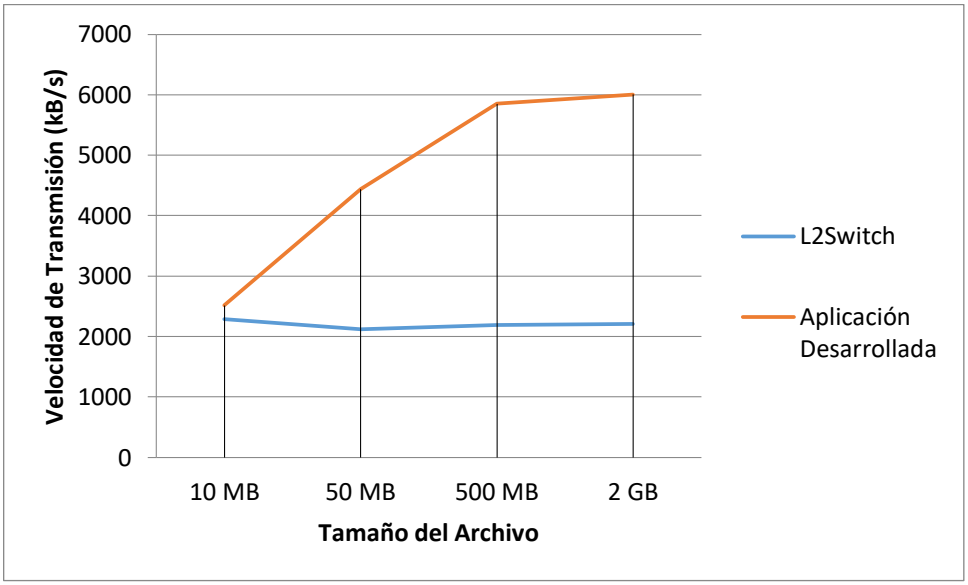

Figura 3.31 Resultados obtenidos en la topología AboveNet mediante L2Switch y la aplicación desarrollada

3.1.2. Pruebas de la función de red virtualizada con Ansible

Para realizar las pruebas de la función de red virtualizada mediante Ansible, se utilizaron dos máquinas virtuales, una de ellas es el *host* cliente y la otra es el servidor como se observa en la Figura 3.32.

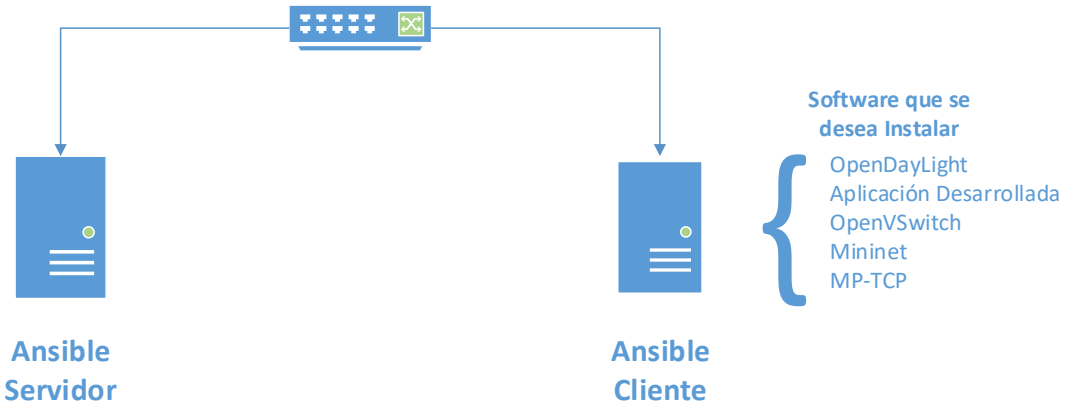

Figura 3.32 Ejemplo de la topología de red para la instalación de la función de red

Toda la información de la función de red virtualizada se encuentra en *playbook* funcionvirtualizada.yml, en el que internamente se dividen la aplicación en varias tareas (TASK). Al ejecutar *playbook* en el *host* servidor se presentan los resultados presentados en la Figura 3.33, la Figura 3.34, la Figura 3.35 y la Figura 3.36.

```
root@odl-virtual-machine:/home/odl/Desktop/mp-tcp# ansible-playbook funcionvirtu
alizada.yml -u odl -K
SUDO password:

PLAY [192.168.100.178] ***************************************************

TASK [Gathering Facts] ***************************************************
Enter passphrase for key '/root/.ssh/id_rsa':
ok: [192.168.100.178]

TASK [Instalacion de vim] ************************************************
changed: [192.168.100.178]

TASK [Agregar Repositorio de Maven] **************************************
changed: [192.168.100.178]

TASK [Agregar Repositorio de Java 8] *************************************
changed: [192.168.100.178]

TASK [Crea el directorio /root/MPTCP] ************************************
changed: [192.168.100.178]
```

Figura 3.33 *Playbook* ejecutado en el *host* servidor (1)

Figura 3.34 *Playbook* ejecutado en el *host* servidor (2)

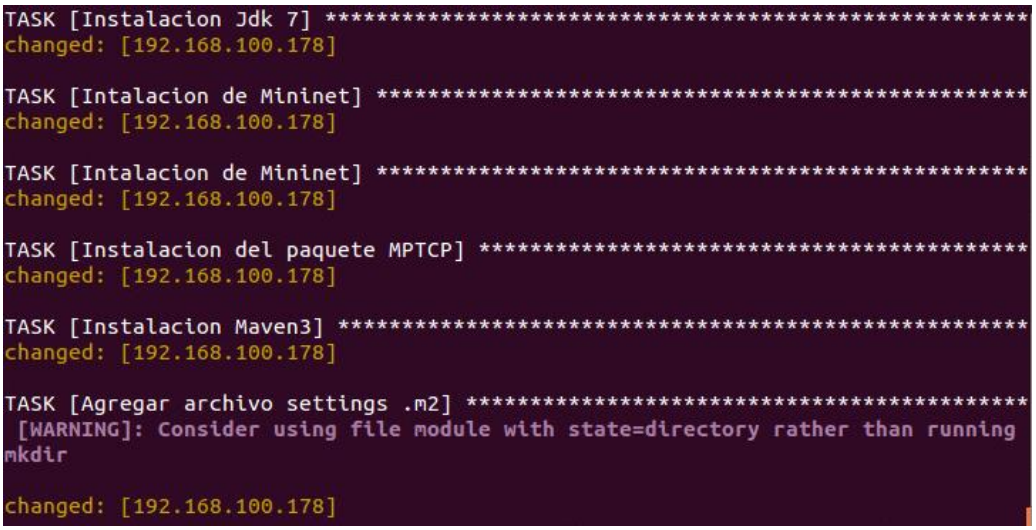

Figura 3.35 *Playbook* ejecutado en el *host* servidor (3)

Figura 3.36 *Playbook* ejecutado en el *host* servidor (4)

Al momento de ejecutar el *playbook,* si el mensaje que aparece después de cada TASK es de color amarrillo o verde significa que este ha sido ejecutado exitosamente por el *host*

cliente; por otro lado, si el mensaje es de color rojo significa que se ha producido un error y el *playbook* se detiene. En la Figura 3.36, que es la última captura de la ejecución de *playbook*, se puede observar que no existe ningún error al automatizar la instalación de la función de red virtualizada en una máquina virtual. Se puede observar que en la Figura 3.35 existe un warning al momento de agregar el archivo setting a .m2 ya que se encuentra en ejecución.

El software que se utilizó en la máquina virtual en la sección de las pruebas que se realizaron a la aplicación desarrollada, fue implementado mediante la función de red virtualizada por Ansible. En el *host* cliente se verificarón el funcionamiento de los cambios como también de las instalaciones realizadas por Ansible.

De los resultados que se obtienen de la prueba de la función de red virtualizada utilizando Ansible se puede observar que en la primera tarea que se realiza es verificar que las llaves instaladas tanto en el *host* cliente como en el servidor son correctas.

Después se instalan todos los requerimientos para el funcionamiento de Open vSwitch y de OpenDayLight, al finalizar esta etapa se agrega un repositorio necesario para el funcionamiento de OpenDayLight y se utilizó Maven para limpiar e instalar el proyecto que contiene la función de red virtualizada y además se reinicia el *host* cliente para que se ejecuten los cambios que se realizaron.

3.1.3. Prueba de funcionamiento de la función de red virtualizada con OpenStack

Para la instalación de OpenStack se utilizó una máquina virtual con la configuración que se presenta en la Tabla 3.9.

Tabla 3.9 Características de la máquina virtual de OpenStack

Características	Especificación
Disco Duro	100 GB
Memoria RAM	110 GB
Número de sockets virtuales	48
Sistema Operativo	Ubuntu 16.04

En la Figura 3.37 se observan las características del bare metal 48 CPU virtuales, 110 GB de memoria RAM y 100 GB de disco local. Todas las características expuestas anteriormente son las disponibles por OpenStack para crear máquinas virtuales o infraestructura de red virtualizada.

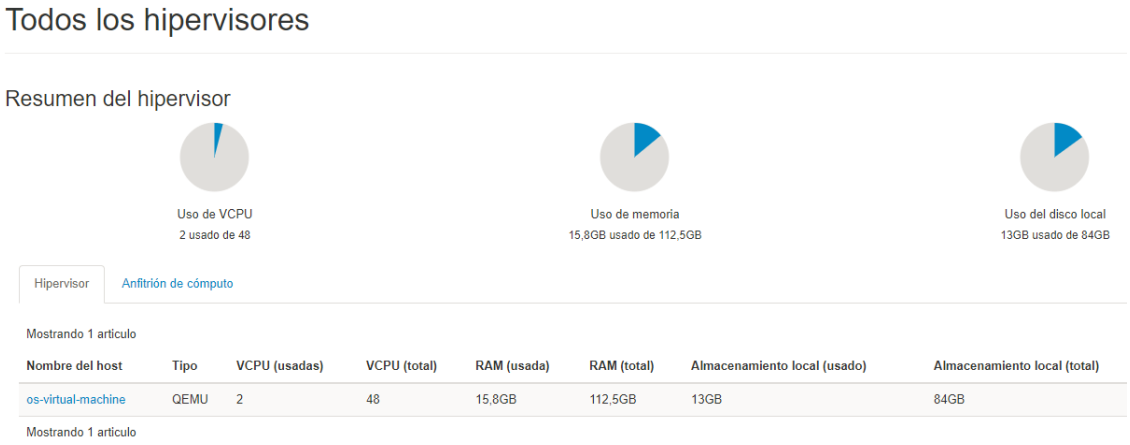

Figura 3.37 Características de Nova en OpenStack instalado

En la Figura 3.38 se observan las imágenes disponibles dentro de OpenStack, por defecto solo se encuentra imágenes con Cirros[57], pero para las pruebas se cargó Ubuntu 14.04.

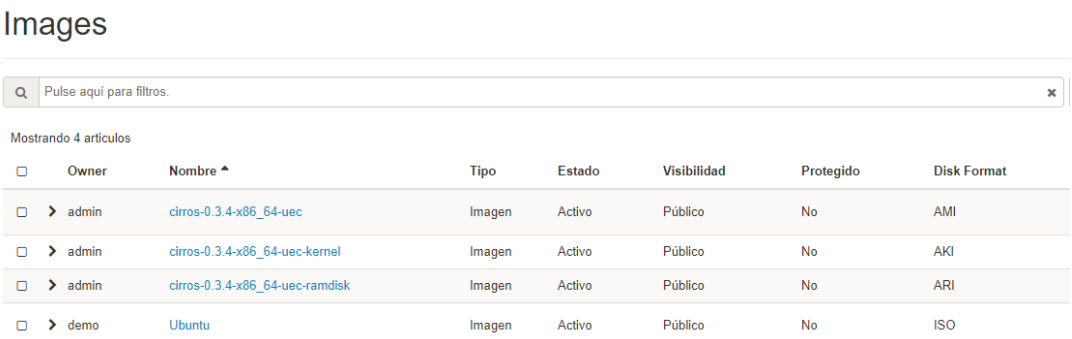

Figura 3.38 Imágenes de Sistemas Operativos en OpenStack

En la Figura 3.39 se presentan las instancias de máquinas virtuales de preubas implementadas en OpenStack, el nombre de estas máquinas virtuales son: ODL y UbuntuServer, en este caso solo tienen una interfaz de red que les permite conectarse a Internet. Además se observa que las máquinas virtuales se encuentran activas y en la zona de Nova.En la Figura 3.40 se presenta la topología de red implementada con las instancias de máquinas virtuales como es la infraestructura de red para tener conexión de Internet en cada máquina.

De la Figura 3.40 se pude observar que existe elementos ya creados, el primer elemento es public el cual se encuentra enlazado con una interfaz creada en la maquina física denominada br-ex la que permite la salida de los paquetes a redes externas o a Internet. El elemento router1 el cual se encarga del enrutamiento de paquetes, en este caso se

[57] Cirros es un pequeño sistema operativo que se especializa en correr en OpenStack.

utiliza este elemento para que las redes que se conecten a él tengan una salida a Internet.

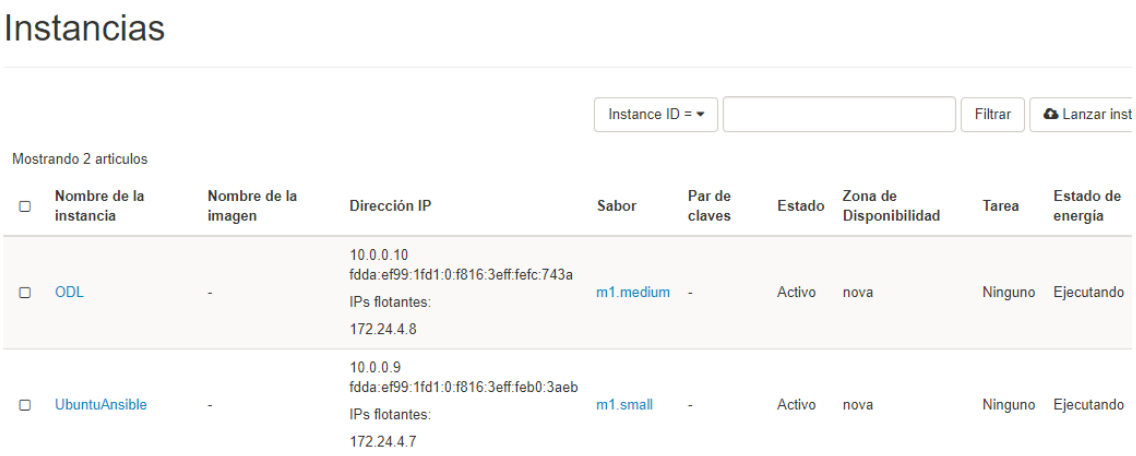

Figura 3.39 Instancias de máquinas virtuales en OpenStack

Finalmente private es la red que se encuentra configurada para que todas las máquinas virtuales que se encuentran conectas a esta red dispongan de una salida a internet, en este caso se encuentran conectadas las máquinas virtuales de pruebas que son: ODL y UbuntuAnsible, a esta red se encuentra asignada un rango de direcciones IP de 10.0.0.1/24 y 10.0.0.254/24.

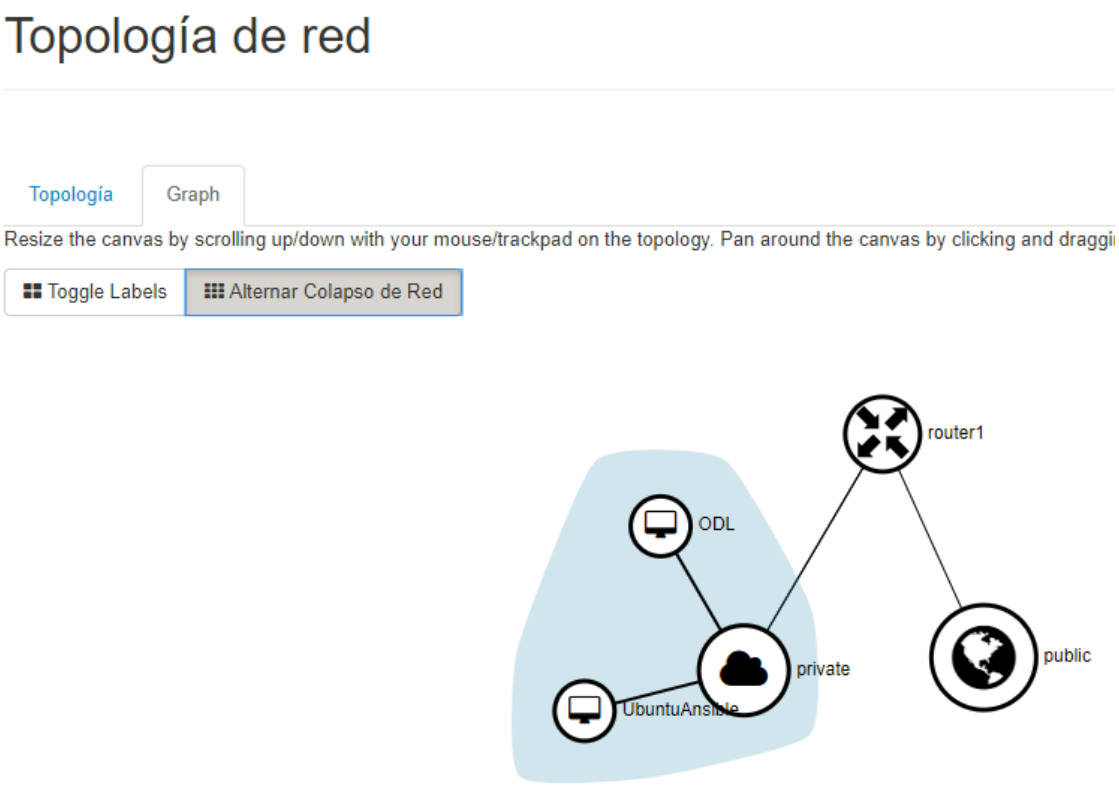

Figura 3.40 Topología de red implementada en OpenStack

3.1.4. Pruebas con equipos físicos

Para esta sección de pruebas con equipos físicos se utilizó switches HP 3500 Procurve, los cuales tienen soporte para el protocolo OpenFlow.

Los equipos físicos van a remplazar a los switches OpenVSwitch que fueron creados mediante Mininet, para los *hosts* se utilizaron dos computadoras con el sistema operativo Linux Ubuntu que además fueron configurados mediante MP-TCP.

La configuración de los equipos HP 3500 Procurve se encuentra descrita en el Anexo III, en el laboratorio solo se cuenta con dos switches HP pero solo con uno de ellos se puede instanciar un total de cinco switches OpenFlow.

En la Figura 3.41 se presenta la topología de red básica en la que se realizaron las pruebas con la aplicación desarrollada.

El controlador OpenDayLight se encuentra instalado en una maquina física con Ubuntu 14, los switches OpenFlow deben asociar a la dirección IP del controlador.

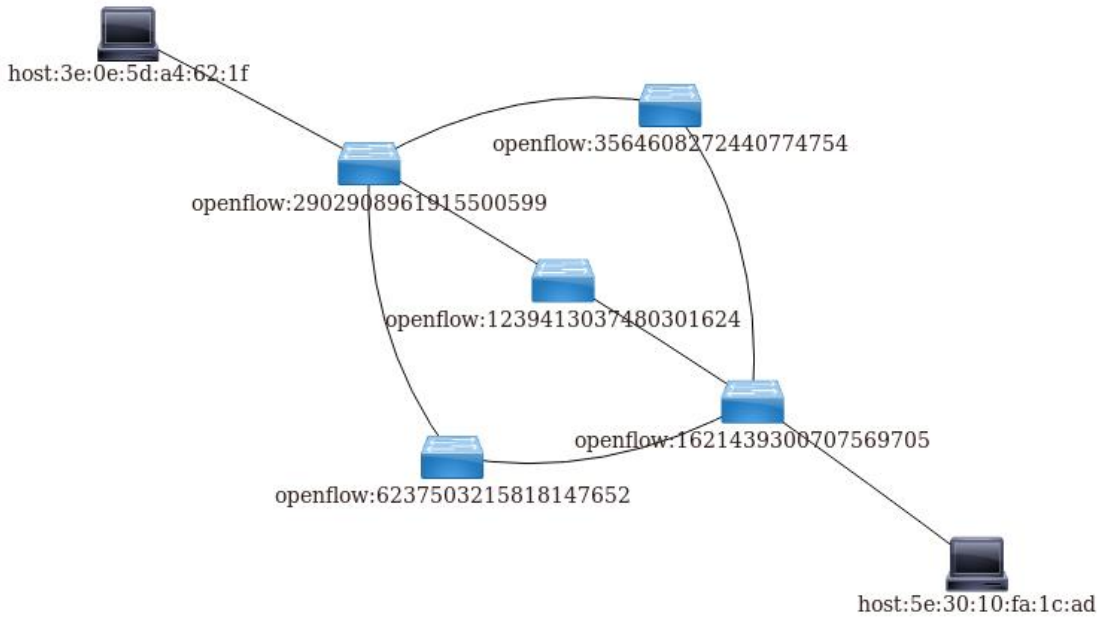

Figura 3.41 Topología de red con equipos físicos

Para realizar las pruebas se utilizaron cuatro archivos de 10MB, 50MB, 500MB y 2GB. Estos archivos fueron transferidos mediante el protocolo FTP.

Los resultados que se obtienen al utilizar L2Switch se los pueden observar en la Figura 3.42, también se realizaron pruebas con la aplicación desarrollada donde se obtuvieron los resultados presentados en la Figura 3.43.

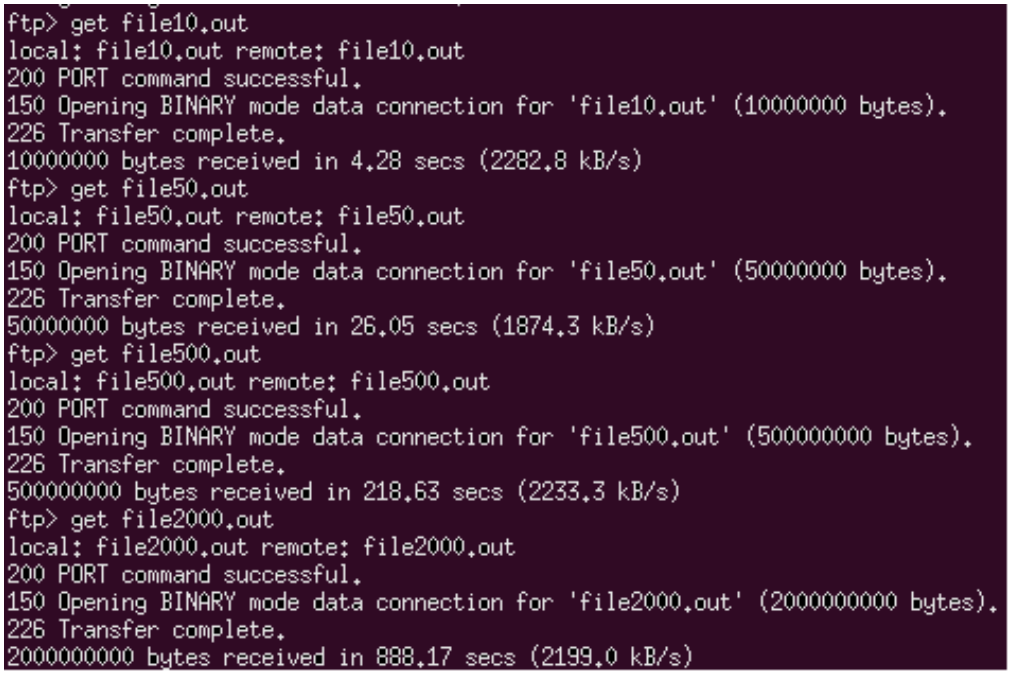

Figura 3.42 Resultados de las pruebas con equipos físicos con L2Switch

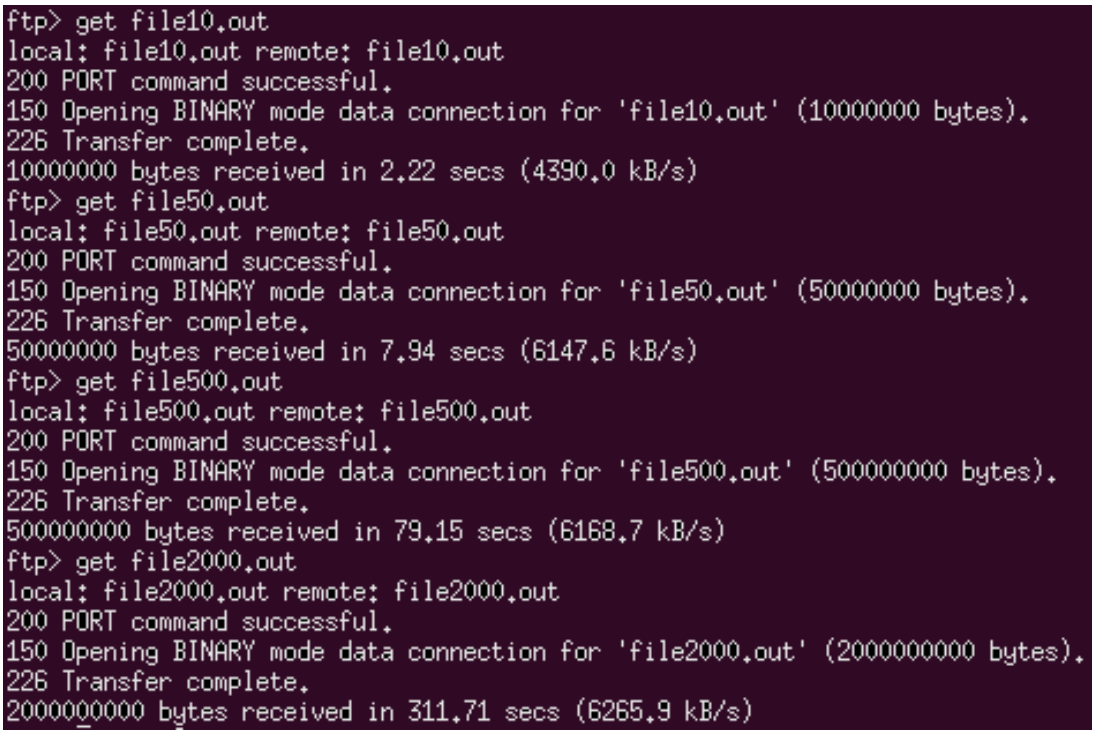

Figura 3.43 Resultados de las pruebas con equipos físicos con la aplicación desarrollada

En la Figura 3.44 se presenta la comparativa entre los resultados obtenidos en las pruebas sobre equipos físicos tanto con L2Switch como con la aplicación desarrollada. Se puede observar que el *performance* usando la aplicación desarrollada supera a L2Switch.

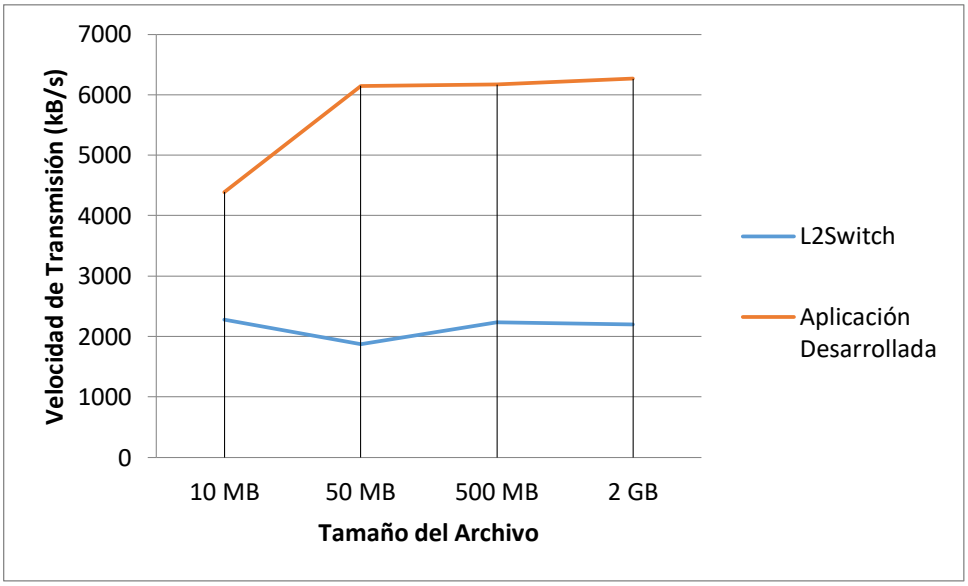

Figura 3.44 Resultados obtenidos de las pruebas en equipos físicos

3.1.5. Pruebas sobre OpenStack con la aplicación desarrollada mediante la función de red virtualizada

En esta sección se realizaron pruebas sobre OpenStack donde se crearon un total de cinco máquinas virtuales las cuales tenían el sistema operativo Ubuntu Cloud 14.04, este tipo de imagen se encuentra especializada para trabajar en ambientes OpenStack. Los nombres de las máquinas virtuales son las siguientes: Mininet, Prueba1, Prueba2 y OpenDayLight1 y OpenDayLight2.

En la Figura 3.45 se puede observar la topología de red que se utilizó para las pruebas. La máquina virtual Mininet contiene tres interfaces de red, las cuales se encuentran conectadas a *private* con el rango de direcciones IP 10.0.0.0/26, red3 y red4 que tiene el rango de direcciones IP 20.0.0./24; en la máquina Mininet se va a crear la infraestructura para conectar a Prueba1 y Prueba2, estos últimos son *hosts*.

La máquina virtual OpenDayLight1 solo contiene una interfaz de red, la cual se encuentra conectada a la red *private* ya que esta máquina contiene al controlador ya que solo se conecta a Mininet (OpenVSwitch). En esta máquina virtual se encuentra configurada con la función de red virtualizada y contiene la aplicación desarrollada que solvente el problema de *shared bottleneck* de MP-TCP.

La máquina virtual OpenDayLight2 solo contiene una interfaz de red, al igual que la OpenDayLight1 se conecta a la red *private* ya que esta máquina solo se va a conectar a Mininet y solo contendrá al controlador OpenDayLight instalado con el *feature* L2Switch para realizar las pruebas.

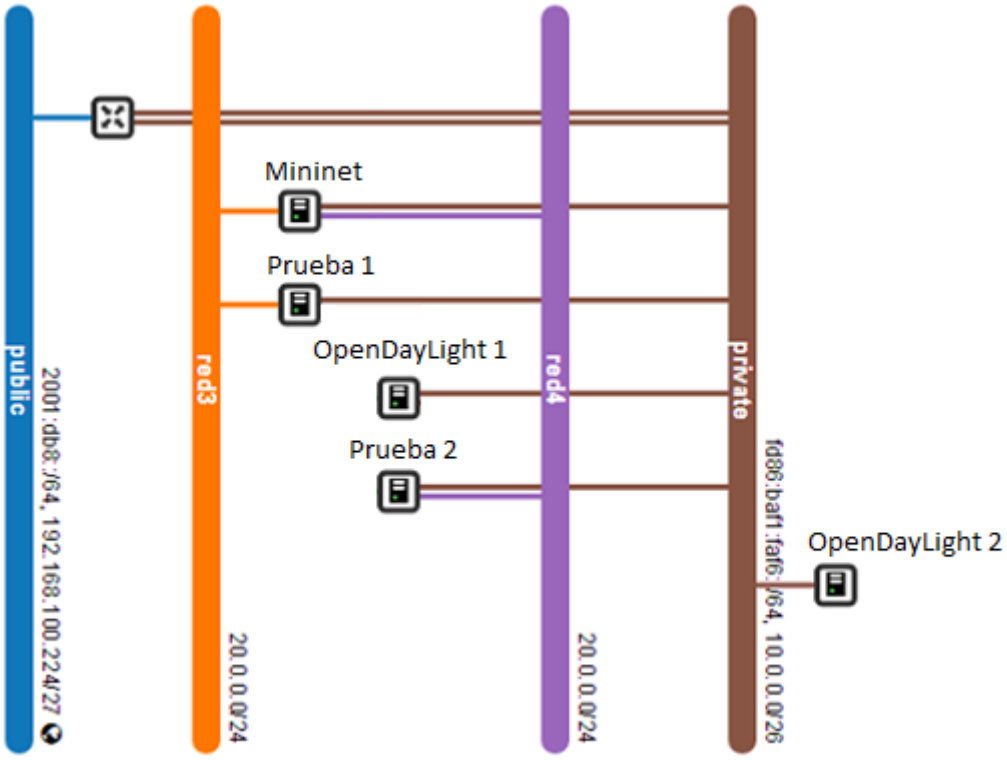

Figura 3.45 Topología de red obtenida de OpenStack para las pruebas

La máquina virtual Prueba1(*host*) contiene dos interfaces de red, la primera de ellas se encuentra conectada a *private* la cual provee Internet, y la segunda de ellas se conecta a red3 para poder conectarse al *host* Prueba2.

Finalmente, la máquina virtual Prueba2 tiene dos interfaces de red al igual que Prueba1, la única diferencia es que la segunda interfaz se encuentra conectada a la red4.

Prueba1 y Prueba2 se conectan atreves red3 y red4 mediante la maquina Mininet la cual contiene la infraestructura necesaria para conectar estas dos redes que tienen el mismo rango de direcciones IP.

Para las pruebas sobre OpenStack se utilizaron cuatro archivos de tamaño 10MB, 50MB, 500MB y 2GB. Se utilizó un cliente y servidor FTP para la transferencia de los archivos y se procedió a medir la velocidad.

La topología implementada dentro de la máquina virtual y que conecta a los dos hosts de prueba es la que se presenta en la Figura 3.46.

La topología de red en prueba es una red básica la cual consta de cinco switches y dos hosts que son las máquinas virtuales Prueba1 y Prueba2.

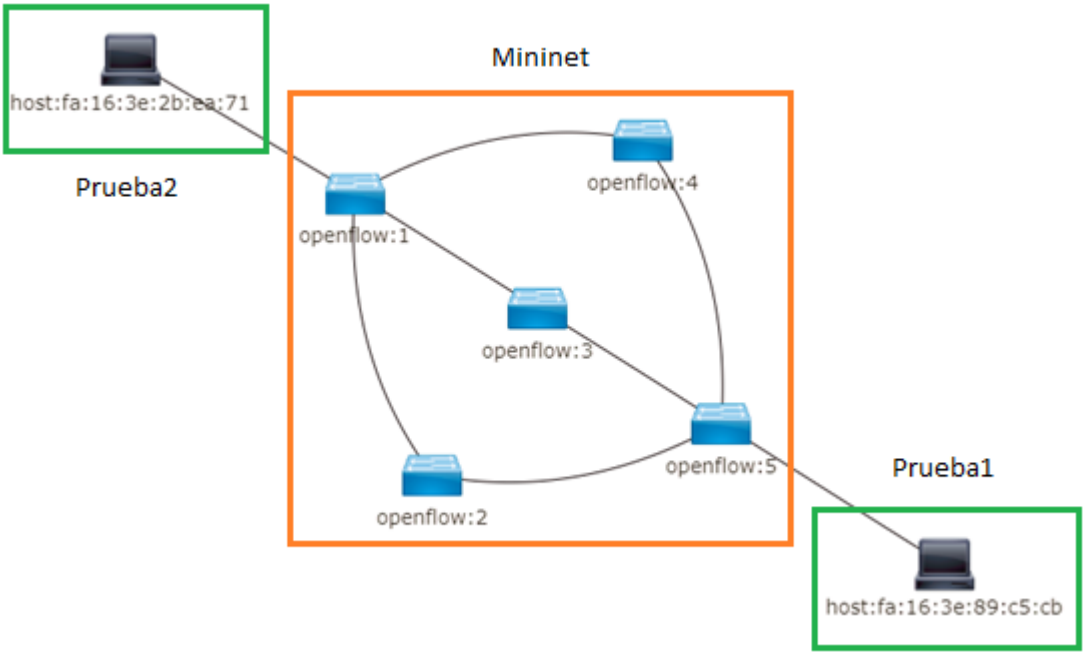

Figura 3.46 Topología de red básica de pruebas sobre OpenStack

Para realizar la prueba en OpenStack existieron algunos inconvenientes ya que existen políticas de seguridad que restringen la conexión entre diferentes máquinas virtuales. Para poder resolver este problema se observaron las reglas instaladas dentro de la máquina que contenía a OpenStack, esto se lo puede realizar con el comando iptables -S el resultado se lo puede observar en la Figura 3.47. De las reglas que se instalan mediante iptables (Figura 3.47 reglas marcadas de color rojo) se puede observar que se encuentra añadida tanto la dirección IP con la máscara y la dirección MAC de la interfaz de red, con esto se obtiene unas reglas muy restringidas para que no se pueda conectar entre máquinas virtuales que no pertenecen a un mismo usuario.

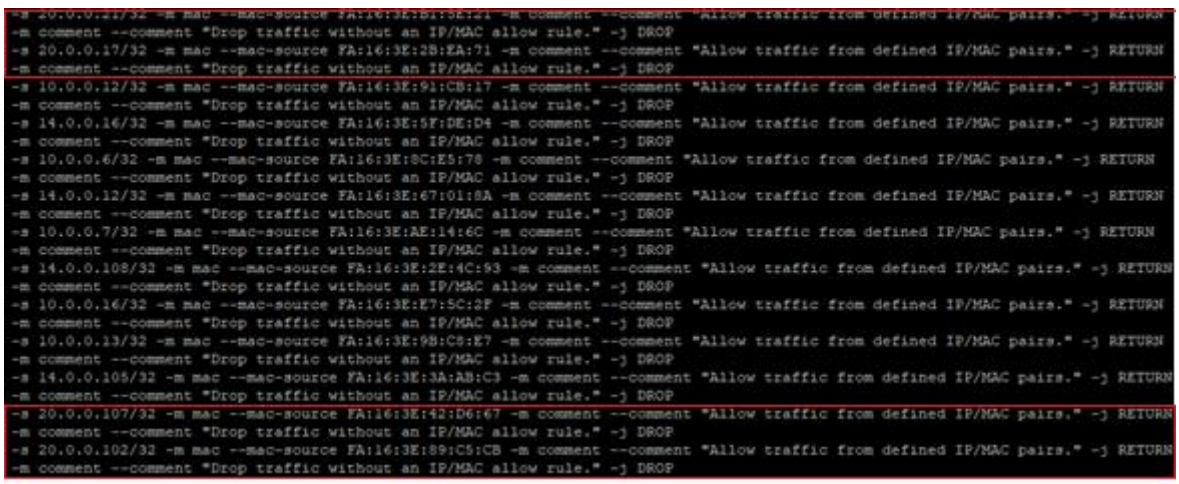

Figura 3.47 Reglas de *firewall* (iptables) instaladas en OpenStack

Para solventar el anterior problema se tiene que instalar reglas en los switches para que en los paquetes se intercambien en la cabecera tanto la dirección IP de destino como de origen, así como las direcciones MAC de origen y destino.

Estos cambios se los realizan con las configuraciones que establece OpenStack o se puede observar en las reglas de iptables para que los paquetes que se envíen desde un *host* a otro no sean eliminados por las reglas del *firewall* de OpenStack.

```
root@mininet1:/home/ubuntu# ovs-ofctl add-flow s1 priority=300,in_port=2,actions
=mod_dl_src=fa:16:3e:b1:5e:21,mod_dl_dst=fa:16:3e:2b:ea:71,mod_nw_src=20.0.0.21,
mod_nw_dst=20.0.0.17,output:1
root@mininet1:/home/ubuntu#
root@mininet1:/home/ubuntu# ovs-ofctl add-flow s1 priority=300,in_port=1,actions
=mod_dl_src=fa:16:3e:42:d6:67,mod_dl_dst=fa:16:3e:89:c5:cb,mod_nw_src=20.0.0.107
,mod_nw_dst=20.0.0.102,output:2
```

Figura 3.48 Reglas instaladas en los switches para solventar problemas que se presentaron en OpenStack

De las pruebas realizadas con L2Switch se obtuvieron los resultados que se observan en la Figura 3.49, donde se observa que la velocidad es constante en todos los archivos que se transfirieron.

```
ftp> get file10.out
local: file10.out remote: file10.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file10.out' (10000000 bytes).
226 Transfer complete.
10000000 bytes received in 4.26 secs (2292.8 kB/s)
ftp> get file50.out
local: file50.out remote: file50.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file50.out' (50000000 bytes).
226 Transfer complete.
50000000 bytes received in 21.27 secs (2295.8 kB/s)
ftp> get file500.out
local: file500.out remote: file500.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file500.out' (500000000 bytes).
226 Transfer complete.
500000000 bytes received in 219.62 secs (2223.3 kB/s)
ftp> get file2000.out
local: file2000.out remote: file2000.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file2000.out' (2000000000 bytes).
226 Transfer complete.
2000000000 bytes received in 867.15 secs (2252.4 kB/s)
```

Figura 3.49 Resultados de las pruebas sobre OpenStack con L2Switch

En la Figura 3.50 se observan los datos que se obtuvieron con la aplicación desarrollada de los archivos que se transfirieron.

```
ftp> get file10.out
local: file10.out remote: file10.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file10.out' (10000000 bytes).
226 Transfer complete.
10000000 bytes received in 4.59 secs (2126.1 kB/s)
ftp> get file50.out
local: file50.out remote: file50.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file50.out' (50000000 bytes).
226 Transfer complete.
50000000 bytes received in 9.92 secs (4923.5 kB/s)
ftp> get file500.out
local: file500.out remote: file500.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file500.out' (500000000 bytes).
226 Transfer complete.
500000000 bytes received in 82.83 secs (5894.8 kB/s)
ftp> get file2000.out
local: file2000.out remote: file2000.out
200 PORT command successful.
150 Opening BINARY mode data connection for 'file2000.out' (2000000000 bytes).
226 Transfer complete.
2000000000 bytes received in 335.78 secs (5816.7 kB/s)
```

Figura 3.50 Resultados de las pruebas sobre OpenStack con la aplicación desarrollada

En la Figura 3.51 se puede observar los resultados que se obtuvieron con la aplicación desarrollada como con L2Switch. Se puede observar que la aplicación desarrollada tiene un mejor desempeño con todos los archivos transferidos.

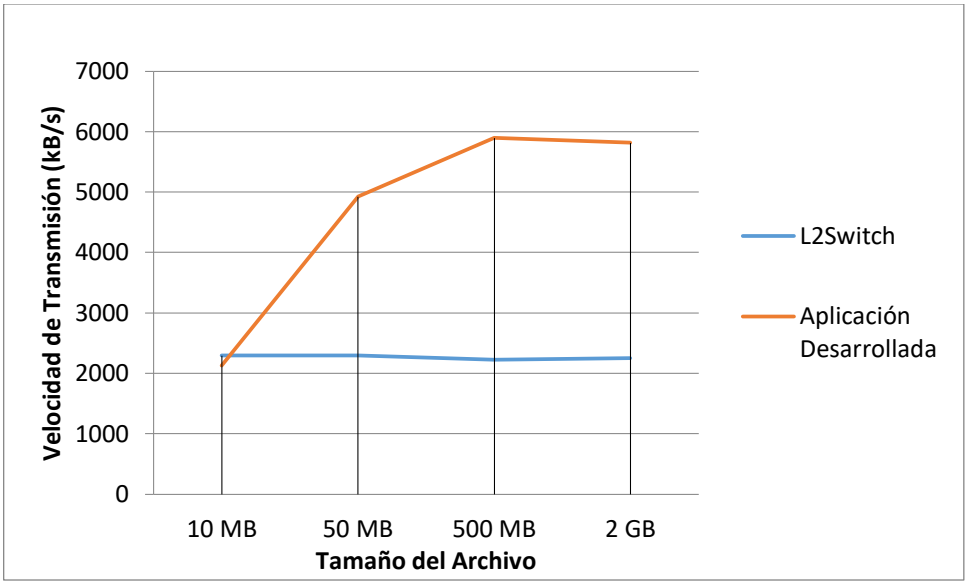

Figura 3.51 Grafica con los resultados obtenidos de las pruebas sobre OpenStack

De los resultados que se obtuvieron en la Figura 3.51 se puede decir que los resultados son esperados, ya que cuando se utiliza la aplicación desarrollada los resultados obtenidos son superiores a los que se consiguen cuando se utiliza el *feature* L2Switch salvo cuando se envía un archivo de tamaño pequeño como en la prueba del paquete de 10MB.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

- La aplicación desarrollada fue comprobada en diferentes topologías de red en las que se producían cuellos de botellas compartidos, en estas topologías usualmente solo se utiliza una ruta de todas las disponibles, desperdiciando las restantes que se tienen en la infraestructura. Con la aplicación desarrollada se utilizan estas rutas que usualmente constan como *backups*, aumentando de esta manera significativamente la velocidad de transmisión y los tiempos disminuyen a comparación de los datos que se obtienen con el *feature* L2Switch.
- Al ejecutarse la aplicación desarrollada en una infraestructura de red con cuellos de botella compartidos, se mejora notablemente las velocidades de transmisión en conexiones MP-TCP. Con los resultados obtenidos de las pruebas realizadas se puede concluir que: cuando los tamaños de los archivos son muy pequeños (10MB) las velocidades de transmisión obtenidas no tienen grandes mejorías inclusive pueden llegar a ser menores en comparación de las que se obtienen con el *feature* L2Switch, esto ocurre porque tanto con la aplicación desarrollada y L2Switch el tiempo de conexión es muy pequeño alrededor de 3 a 4 segundos y no se establecen nuevos subflujos. Por otro lado cuando se utilizan archivos de tamaños relativamente grandes (2GB) las velocidades de transmisión aumentan considerablemente con el uso de la aplicación, esto se debe a que el tiempo de la conexión es aumenta permitiendo de esta manera que se establezcan nuevas posibles rutas.
- Una limitante adicional que tienen los escenarios en los que se realizaron las pruebas es el ancho de banda entre el host y el cuello de botella, ya que si estos enlaces no tienen el ancho de banda suficiente, la aplicación genera las rutas pero no se alcanzan las velocidades de transmisión esperadas; esto se debe a que los enlaces entre el *host* y los cuellos de botellas compartidos se saturan.
- La aplicación desarrollada genera reglas solamente para conexiones que utilizan el protocolo MP-TCP, para el resto de conexiones, la aplicación no realiza cambios sino que se utilizan las reglas generadas por el *feature* L2Switch.Un ejemplo de lo antes descrito es cuando se utiliza FTP (TCP) va a utilizar las reglas que la aplicación desarrollada instala en cada nodo por otro lado si se utiliza DHCP (UDP) va a utilizar las reglas del *feature* L2Switch. Con el *feature* L2switch la transmisión de diferentes tipos de protocolos (UDP) es transparente.

- Al utilizar la aplicación desarrollada se observó que cuando se maneja flujos continuos de datos como en el protocolo FTP se obtiene mejores resultados que al utilizar flujos de tipo ráfaga como es el caso del protocolo HTTP. En el caso del protocolo FTP al momento que se establece la conexión mediante MP-TCP tiene la posibilidad de generar todos los subflujos dependiendo del tamaño del archivo, por otro lado, las conexiones de tipo ráfaga son de corta duración y no se establecen los subflujos y no se obtienen los beneficios de la aplicación desarrollada.
- Al realizar las pruebas con Mininet se crearon plantillas para la creación de entornos de pruebas esto ahorra significativamente el tiempo de realización de pruebas y se verifica el funcionamiento de la aplicación desarrollada. Al instalar el protocolo MP-TCP sobre un sistema operativo y utilizar Mininet, las máquinas virtuales generadas ya que se encuentran con el protocolo MP-TCP instalado y funcionando de forma similar que el sistema operativo anfitrión.
- Para que no existan errores o inconsistencias en la aplicación desarrollada todos los paquetes de tipo TCP se envían al controlador OpenDayLight, esto se lo realizó ya que en un inicio solo se utilizaban las reglas que implementaba el feature L2Switch. Al utilizar solamente L2Switch se generaban errores ya que al momento que se establecían las reglas en los switches solamente se enviaban paquetes al controlador cuando la topología de red sufría algún cambio, después de eso no se enviaba ningún otro paquete al controlador ya que no era necesario porque la topología no se alteraba. Si no se envían los paquetes MP-TCP al controlador OpenDayLight no es posible procesar la información que contiene cada paquete y no se puede establecer ni el primer subflujo.
- Cuando la aplicación desarrollada diferenciaba entre diferentes conexiones MP-TCP se agrega una robustez ya que solo se utilizará las posibles rutas que se asignaron para cada conexión MP-TCP. Cuando la aplicación no podía diferenciar entre diferentes conexiones MP-TCP al momento que la red tenía más de una conexión, se producía un sobrelapamiento de caminos ya que compartían el mismo contador con lo que reduce el *throughput* porque se van a repartir los caminos ya asignados a una conexión MP-TCP.
- La aplicación solo utiliza los paquetes de tipo MP-TCP recibidos en el controlador para procesarlos y crear nuevas reglas y así enviar por rutas establecidas mediante el Algoritmo de Dijkstra, para el resto de paquetes solo se va a utilizar las reglas que establecieron por medio del *feature* L2Switch. Las reglas que

instala L2Switch tienen una prioridad más baja que las reglas que instala la aplicación desarrollada.

- Al procesar la información de los caminos con los nodos que se los denominó como nodos intermedios mejoró el performance de la aplicación ya que si no se los eliminaba existían problemas porque cuando se utiliza el Algoritmo de Dijkstra, solo empleará un camino desperdiciando los restantes que disponga la infraestructura de red. Lo que se obtuvo al manejar los nodos intermedios es eliminarlos momentáneamente y solo procesar a los nodos que se encuentran dentro de los cuellos de botella ya que entre este segmento se encuentran las rutas adicionales que mejoran el performance de la infraestructura de red. Después de obtener las posibles rutas se adjuntan los nodos intermedios para obtener una conexión extremo a extremo y tener una conexión entre los *host* implicados.
- DevOps no solo se limita a la utilización de una herramienta sino que intenta cambiar la ideología de cómo generar y mantener un proyecto sin que exista una separación entre el desarrollo y la operación de la aplicación. Además, DevOps elimina la creación de tareas repetitivas utilizando herramientas que facilitan la implementación de las tareas de forma automáticas y sin cometer errores ya que se crean en base de una estructura ya probada y establecida.
- DevOps se encuentra directamente ligado con la creación de funciones de red virtualizadas ya que al momento de producir de forma comercial o en laboratorio la función se necesita que el despliegue sea lo más rápido posible y que no se produzcan errores. Al utilizar DevOps se crea una plantilla donde se prueban todos los posibles errores para obviarlos, además que para realizar la tarea se tiene que utilizar una herramienta la cual internamente realiza pruebas unitarias a cada módulo implementado.
- Las herramientas de DevOps actualmente se encuentran instaladas dentro de paquetes que abarcan instalaciones muy grandes o que conllevan mucho tiempo, tal es el caso de OpenStack que utiliza el software DevStack para la instalación completa. Dentro de DevStack se encuentra la herramienta de DevOps Ansible la cual maneja toda la instalación y pruebas de OpenStack, esta herramienta también fue utilizada en la creación de la función de red virtualizada que solvente el problema de shared bottleneck MP-TCP.
- Al utilizar la herramienta de DevOps Ansible para crear la función de red virtualizada, se ahorró tiempo al momento de implementar la función ya que para que funcione correctamente OpenDayLight existen algunos requerimientos previos

los cuales son necesarios para la implementación de la aplicación. Ansible no solo instala los requerimientos de software necesarios sino que también realiza pruebas de los elementos instalados, si existe algún problema con la tarea encomendada a Ansible en el *host* servidor se va a presentar los log necesarios para resolver el problema.

- Las herramientas disponibles para realizar DevOps es muy extensa como Puppet o Chef las cuales necesitan un mayor conocimiento sobre la estructura de la herramienta, tienen su propio lenguaje de programación y es necesario la instalación de un cliente en cada *host* que se desea implementar. Por otro lado, existe la herramienta Ansible la que se maneja de forma sencilla sin la necesidad de instalar un cliente, además su lenguaje de programación es a base de etiquetas las cuales facilitan realizar las tareas encomendadas.
- Cuando se realizaron las pruebas de la aplicación en ambientes virtuales como es el caso de Mininet fueron muy rápidas, además se pudieron establecer una cantidad muy amplia de pruebas ya que para modificar la estructura de red solo se tenía que crear un archivo programado en Python el cual tenía elementos muy básicos de configuración.
- Las reglas que utiliza OpenStack son restrictivas ya que esta plataforma utiliza medios compartidos con varios usuarios y la integridad de la información es la base de la seguridad implementada por OpenStack. Al desarrollar el trabajo de titulación existieron algunos problemas subyacentes de la seguridad implementada por OpenStack los cuales fueron resueltos observando las reglas implementadas dentro del *firewall* del sistema. Para la solución del problema se agregaron reglas en los switches OpenFlow para que cambien tanto las direcciones IP y MAC de origen como de destino y que coincidan con las reglas que se encuentran en el *firewall* de OpenSatck.
- Al crear una función virtualizada de red de la aplicación que solventa el problema del *shared bottleneck* del protocolo MP-TCP mediante Ansible se ahorrará tanto tiempo como pruebas las cuales no serán realizadas por el administrador de la red. Esto se lo comprobó al momento que se realizó las pruebas por medio de OpenStack y la función de red virtualizada donde se instaló la función de red en varias máquinas virtuales en las que se pudo realizar de forma paralela ahorrando tiempo en la instalación y no se generaron errores en la red ni en la aplicación desarrollada.

4.2. Recomendaciones

- Para el manejo de OpenDayLight se recomienda el uso de Maven en una versión superior a la 3.3 ya que si se instalan versiones anteriores se producirán problemas en el manejo de los proyectos que utilice OpenDayLight y se generarán errores.
- Se recomienda utilizar la última versión de OpenStack ya que muchas versiones anteriores existen errores y paquetes que se encuentran descontinuados los cuales no permiten instalar OpenStack.
- Utilizar OpenStack sobre un servidor de buenas características ya que si se utiliza en máquinas virtuales tanto el despliegue como el manejo se vuelve lento además que los recursos son limitados por el *host* cliente.
- Para trabajos futuros se puede proponer la utilización de una infraestructura dedicada para la creación y administración de funciones de red virtualizadas como es el caso de OPENFV.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] C. Valdivieso, C. Cajas y I. Bernal, Implementación de un Analizador MP-TCP (MultiPath - TCP) empleando una Red Definida por Software, Quito: Escuela Politécnica Nacional , 2017.

[2] C. Cajas, C. Valdivieso, D. Mejía y I. Bernal, "Simulación en NS3 del problema denominado Cuello de Botella Compartido (Shared Bottleneck) que se presenta en el protocolo MP-TCP," *Revista de Investigación en Tecnologías de Información (RITI),* vol. 5, nº 9, pp. 68-76, Junio 2017.

[3] Netmanias, “Analysis of LTE – WiFi Aggregation Solutions," 21 Marzo 2016. [En línea]. Available: https://www.netmanias.com/en/post/reports/8532/laa-lte-lte-u-lwa-mptcp-wi-fi/analysis-of-lte-wifi-aggregation-solutions. [Último acceso: 20 Septiembre 2017].

[4] S. Ferlin, Ö. Alay, T. Dreibholz, D. A. Hayes y M. Welzl, "Revisiting congestion control for multipath TCP with shared bottleneck detection," *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications,* pp. 1-9, 2016.

[5] M. Sandri, A. Silva, L. Rocha y F. L. Verdi, "On the Benefits of Using Multipath TCP and Openflow in Shared Bottlenecks," *2015 IEEE 29th International Conference on Advanced Information Networking and Applications,* pp. 9-16, 2015.

[6] A. Ford, C. Raiciu, M. Handley y O. Bonaventure, "Internet Engineering Task Force (IETF)," Enero 2013. [En línea]. Available: https://tools.ietf.org/html/rfc6824. [Último acceso: 21 Septiembre 2017].

[7] Oracle, " Protocolo SCTP," Mayo 2010. [En línea]. Available: https://docs.oracle.com/cd/E19957-01/820-2981/ermih/index.html. [Último acceso: 21 Septiembre 2017].

[8] M. N. Ellanti, S. S. Gorshe, L. G. Raman y W. D. Grover, Next Generation Transport Networks: Data, Management, and Control Planes, Nueva York: Springer, 2005.

[9] C. Raiciu y C. Paasch, "Multipath TCP," University College London, Londres, 2010.

[10] R. Moyano, " Diseño e implementación de escenarios virtuales de red para evaluar el funcionamiento de MP-TCP," 12 Julio 2013. [En línea]. Available: http://www.dit.upm.es/~posgrado/doc/TFM/TFMs2012-2013/TFM_Ricardo_Flores_2013.pdf. [Último acceso: 21 Septiembre 2017].

[11] D. Gómez, « Uso de técnicas de Network Coding y Multi-path para la mejora de las comunicaciones sobre redes malladas inalámbricas,» 2 Marzo 2015. [En línea]. Available: https://repositorio.unican.es/xmlui/bitstream/handle/10902/6411/TesisDGF.pdf?sequence=1. [Último acceso: 21 Septiembre 2017].

[12] I. Bernal y D. Mejía, «Las Redes Definidas por Software y los Desarrollos Sobre Esta Temática en la Escuela Politécnica Nacional,» *Revista Politécnica Nacional,* vol. 37, nº 1, pp. 1-11, 2016.

[13] O. N. Foundation, “Software-Defined Networking (SDN) Definition," Open Networking Foundation, 23 Enero 2016. [En línea]. Available: https://www.opennetworking.org/sdn-definition/. [Último acceso: 22 Septiembre 2017].

[14] D. Meyer, " What is Software Defined Networking (SDN)?," SDNCentral, 4 Mayo 2015. [En línea]. Available: https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/. [Último acceso: 22 Septiembre 2017].

[15] S. Velrajan, “SDN Architecture," The Tech, 25 Diciembre 2012. [En línea]. Available: http://www.thetech.in/2012/12/. [Último acceso: 22 Septiembre 2017].

[16] Y. Marín, " Plataforma de pruebas para evaluar el desempeño de las redes definidas por software basadas en el protocolo OpenFlow," 2016. [En línea]. Available: https://www.researchgate.net/publication/312187600_PLATAFORMA_DE_PRUEBAS_PARA_EVALUAR_EL_DESEMPENO_DE_LAS_REDES_DEFINIDAS_POR_SOFTWARE_BASADAS_EN_EL_PROTOCOLO_OPENFLOW. [Último acceso: 22 Septiembre 2017].

[17] kspviswa, "OpenFlow Version RoadMap," kspviswa, Julio 26 2016. [En línea]. Available: http://kspviswa.github.io/OpenFlow_Version_Roadmap.html. [Último acceso: 9 Febrero 2018].

[18] D. Samociuk, "Secure Communication Between OpenFlow Switches and Controllers," *AFIN 2015 : The Seventh International Conference on Advances in*

Future Internet, vol. I, nº 1, pp. 32-37, 2015.

[19] Mininet, "Mininet Walkthrough," Octopress, Abril 2014. [En línea]. Available: http://mininet.org/walkthrough/. [Último acceso: 23 Septiembre 2017].

[20] R. Álvarez, " Estudio de las redes definidas por software mediante el desarrollo de escenarios virtuales basados en el controlador OpenDayLight," Junio 2015. [En línea]. Available: http://www.dit.upm.es/~posgrado/doc/TFM/TFMs2014-2015/TFM_Raul_Alvarez_Pinilla_2015.pdf. [Último acceso: 23 Septiembre 2017].

[21] J. L. Muñoz, “OpenDaylight SDN controller platform," Octubre 2015. [En línea]. Available: https://upcommons.upc.edu/bitstream/handle/2117/79422/odl-completo.pdf?sequence=1&isAllowed=y. [Último acceso: 23 Septiembre 2017].

[22] C. Santamaría, " Protocolos Multicast en redes SDN," Julio 2016. [En línea]. Available: http://dtstc.ugr.es/it/pfc/proyectos_realizados/downloads/Memoria2016_CarlosSantamaria.pdf. [Último acceso: 23 Septiembre 2017].

[23] R. Toghraee, Learning OpenDaylight, Birmingham: Packt Publishing, 2017.

[24] M. R. Hinkle, "Socialized Software," Creative Commons, 8 Abril 2013. [En línea]. Available: http://socializedsoftware.com/2013/04/08/the-linux-foundations-collaboration-opendaylight-project-open-source-sdn/. [Último acceso: 23 Septiembre 2017].

[25] OpenDaylight, "Wiki OpenDayLight," MediaWiki, Febrero 2014. [En línea]. Available: https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:L2_Switch. [Último acceso: 23 Septiembre 2017].

[26] Inocybe, " OpenDayLight controller md-sal faqs," Inocybe Technologies, Marzo 2015. [En línea]. Available: http://docs.inocybe.com/dev-guide/content/_opendaylight_controller_md_sal_faqs.html. [Último acceso: 23 Septiembre 2017].

[27] A. García, " Simple y rápido. Entiende qué es Maven en menos de 10 min.," JavierGarzas, 6 Junio 2014. [En línea]. Available: http://www.javiergarzas.com/2014/06/maven-en-10-min.html. [Último acceso: 25 Septiembre 2017].

[28] Privotal, “Building Java Projects with Maven," Pivotal Software, 12 Mayo 2016. [En línea]. Available: https://spring.io/guides/gs/maven/. [Último acceso: 25 Septiembre

2017].

[29] L. Ferrero, " Qué es OSGi?," LinkedIn Corporation, 3 Septiembre 2014. [En línea]. Available: https://www.linkedin.com/pulse/20140903145139-12717948-qu%C3%A9-es-osgi/. [Último acceso: 25 Septiembre 2017].

[30] S. Patil, " Hello, OSGi, Part 1: Bundles for beginners," IDG Communications, Inc., 8 Marzo 2008. [En línea]. Available: https://www.javaworld.com/article/2077837/application-development/java-se-hello-osgi-part-1-bundles-for-beginners.html. [Último acceso: 25 Septiembre 2017].

[31] OSGi, “Architecture OSGi," OSGi™ Alliance, 21 Enero 2011. [En línea]. Available: https://www.osgi.org/developer/architecture/. [Último acceso: 25 Septiembre 2017].

[32] J. Goodyear, J. Edstrom y H. Kesler, Learning Apache Karaf, Londres: Packt Publishing , 2013.

[33] L. M. Garcia, " Un poco de Apache Karaf," Un poco de java y +, 3 Septiembre 2013. [En línea]. Available: https://unpocodejava.wordpress.com/2013/09/03/un-poco-de-apache-karaf/. [Último acceso: 25 Septiembre 2017].

[34] S. Beak, "OpenDayLight," 16 Marzo 2015. [En línea]. Available: https://es.slideshare.net/rkawkxms/2015316opendaylight-45868156/3. [Último acceso: 25 Septiembre 2017].

[35] J. d. J. Gil y J. Botero, "Network Functions Virtualization: A Survey," *IEEE Latin America Transactions,* vol. 14, nº 2, pp. 983-997, 2016.

[36] M. Rouse, "network functions virtualization (NFV)," TechTarget, 13 Marzo 2016. [En línea]. Available: http://searchsdn.techtarget.com/definition/network-functions-virtualization-NFV. [Último acceso: 26 Septiembre 2017].

[37] SDxCentral, " What is NFV – Network Functions Virtualization – Definition?," SDxCentral, 21 Marzo 2014. [En línea]. Available: https://www.sdxcentral.com/nfv/definitions/whats-network-functions-virtualization-nfv/. [Último acceso: 26 Septiembre 2017].

[38] ETSI, " Network Functions Virtualisation," ETSI, Noviembre 2012. [En línea]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv. [Último acceso:

27 Septiembre 2017].

[39] P. Pate, " NFV and SDN: What's the Difference?," SDxCentral, 30 Marzo 2013. [En línea]. Available: https://www.sdxcentral.com/articles/contributed/nfv-and-sdn-whats-the-difference/2013/03/. [Último acceso: 26 Septiembre 2013].

[40] M. Contact, " Diferencias entre SDN y NFV," Contact, 13 Mayo 2016. [En línea]. Available: http://mundocontact.com/diferencias-entre-sdn-y-nfv/. [Último acceso: 26 Septiembre 2017].

[41] M. Malgosa, " Evaluación de prestaciones mediante NVF," 2015. [En línea]. Available: http://upcommons.upc.edu/bitstream/handle/2117/78028/NFV.pdf ?sequence=1. [Último acceso: 27 Septiembre 2017].

[42] SDxCentral, " What is OPNFV or Open Platform for NFV Project?," SDxCentral, 12 Enero 2014. [En línea]. Available: https://www.sdxcentral.com/nfv/definitions/opnfv/. [Último acceso: 26 Septiembre 2017].

[43] A. Kapadia y N. Chase, Understanding OPNFV: Accelerate NFV Transformation using OPNFV, Sunnyvale: Mirantis Inc., 2017.

[44] D. Núñes, "Implementación de un prototipo de Software como Servicio (SaaS) para pequeñas y medianas empresas," Diciembre 2013. [En línea]. Available: http://bibdigital.epn.edu.ec/bitstream/15000/7108/1/CD-5294.pdf. [Último acceso: 28 Septiembre 2017].

[45] B. Umbarila, "Cloud Computing," Diciembre 2011. [En línea]. Available: http://www.fce.unal.edu.co/media/files/documentos/uifce/proyectos/Cloud_computing.pdf. [Último acceso: 27 Septiembre 2017].

[46] B. Galarza, G. Zaccardi, D. Encinas y M. Morales, "Análisis de despliegue de una IaaS utilizando Openstack," Agosto 2015. [En línea]. Available: http://sedici.unlp.edu.ar/bitstream/handle/10915/50514/Documento_completo.pdf-PDFA.pdf?sequence=1. [Último acceso: 27 Septiembre 2017].

[47] A. Cobos, "Despliegue de arquitectura cloud basada en OpenStack y su integración con Chef sobre CentOS," 2014. [En línea]. Available: http://bibing.us.es/proyectos/abreproy/90140/fichero/Memoria.pdf. [Último acceso: 27 Septiembre 2017].

[48] R. Hat, "El concepto de OpenStack," Red Hat, Inc., Junio 2015. [En línea]. Available: https://www.redhat.com/es/topics/openstack. [Último acceso: 27 Septiembre 2017].

[49] K. Proulx, "OpenStack: The Big Tent," LinkedIn Corporation, 24 Septiembre 2016. [En línea]. Available: https://www.linkedin.com/pulse/openstack-big-tent-ken-proulx/. [Último acceso: 27 Septiembre 2017].

[50] P. Bruna, "¿Qué es DevOps?," IT Linux , 23 Octubre 2013. [En línea]. Available: http://blog.itlinux.cl/blog/2013/10/23/que-es-devops/. [Último acceso: 28 Septiembre 2017].

[51] H. Chawla, "DevOps: cómo romper la barrera entre Desarrollo y Operaciones," Atlassian, 12 Diciembre 2015. [En línea]. Available: https://es.atlassian.com/devops. [Último acceso: 27 Septiembre 2017].

[52] J. Ruiz, "Qué es DevOps (y sobre todo qué no es DevOps)," {Paradigma, 26 Noviembre 2015. [En línea]. Available: https://www.paradigmadigital.com/techbiz/que-es-devops-y-sobre-todo-que-no-es-devops/. [Último acceso: 27 Septiembre 2017].

[53] D. Fosted, "What is DevOps? Patrick Debois Explains," The Linux Foundation, 21 Junio 2016. [En línea]. Available: https://www.linux.com/blog/what-devops-patrick-debois-explains. [Último acceso: 28 Septiembre 2017].

[54] J. Verona, Practical DevOps, Birmingham: Packt Publishing Ltd., 2016.

[55] K. Beck y M. Beedle, "Manifiesto por el Desarrollo Ágil de Software," Ward Cunningham , Marzo 2001. [En línea]. Available: http://agilemanifesto.org/iso/es/manifesto.html. [Último acceso: 28 Septeimbre 2017].

[56] J. J. Mora, "El movimiento DevOps," Telefónica, 28 Enero 2015. [En línea]. Available: https://blogthinkbig.com/el-movimiento-devops. [Último acceso: 29 Septiembre 2017].

[57] S. Seetharaman, "OpenDaylight Application Developer's tutorial," SDN Hub, 25 Marzo 2016. [En línea]. Available: http://sdnhub.org/tutorials/opendaylight/. [Último acceso: 10 Octubre 2017].

[58] Devstack, "DevStack," 28 Noviembre 2017. [En línea]. Available: https://docs.openstack.org/devstack/latest/. [Último acceso: 2 Diciembre 2017].

[59] OpenStack, "Keystone, the OpenStack Identity Service," 11 Noviembre 2017. [En línea]. Available: https://docs.openstack.org/keystone/pike/. [Último acceso: 10 Diciembre 2017].

[60] OpenStack, "Welcome to Glance’s documentation!," 11 Noviembre 2017. [En línea]. Available: https://docs.openstack.org/glance/latest/. [Último acceso: 10 Diciembre 2017].

[61] OpenStack, "OpenStack Compute (nova)," 11 Noviembre 2017. [En línea]. Available: https://docs.openstack.org/nova/pike/. [Último acceso: 12 Diciembre 2017].

[62] OpenStack, "Cinder," 11 Novimebre 2017. [En línea]. Available: https://wiki.openstack.org/wiki/Cinder. [Último acceso: 10 Diciembre 2017].

[63] Openstack, "OpenStack Networking ("Neutron")," 11 Noviembre 2017. [En línea]. Available: https://wiki.openstack.org/wiki/Neutron. [Último acceso: 10 Diciembre 2017].

[64] OpenStack, "Horizon: The OpenStack Dashboard Project," 11 Noviembre 2017. [En línea]. Available: https://docs.openstack.org/horizon/latest/. [Último acceso: 10 Diciembre 2017].

[65] J. Geerling, Ansible for DevOps: Server and configuration management for humans, California: Leanpub, 2017.

[66] A. Reyes, "Introducción a Ansible," The Big Computer, 26 Mayo 2017. [En línea]. Available: https://thebigcomputer.com/ansible/introduccion-a-ansible/. [Último acceso: 20 Diciembre 2017].

[67] J. D. M. Nieto, "Introducción a Ansible," Open Source technologies explained, 4 Mayo 2017. [En línea]. Available: http://www.dmartin.es/2017/04/introduccion-a-ansible-2-vol-2/. [Último acceso: 20 Diciembre 2017].

[68] S. Rao, "What is Network Functions Virtualization?," The New Stack, 30 Octubre 2015. [En línea]. Available: https://thenewstack.io/de-ossify-the-network-with-function-virtualization/. [Último acceso: 24 Diciembre 2017].

[69] "MultiPath TCP - Linux Kernel implementation : Users - Apt Repository browse," Multipath-tcp.org, 2 Enero 2016. [En línea]. Available: https://multipath-

tcp.org/pmwiki.php/Users/AptRepository. [Último acceso: 23 Enero 2018].

[70] L. Carvajal, Metodología de la Investgación Científica. Curso general y aplicado, 28 ed., Santiago de Cali: U.S.C., 2006, p. 139.

[71] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker y J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review,* vol. 38, nº 2, pp. 69-74, 2008.

[72] SDxCentral, "OPNFV Tests its Software on OCP Hardware," SDxCentral, 3 Junio 2015. [En línea]. Available: https://www.sdxcentral.com/articles/news/opnfv-tests-software-ocp-hardware/2017/02/. [Último acceso: 27 Septiembre 2017].

[73] E. Segal, "NFV (network-functions-virtualization)," Vega-BI, 24 Agosto 2014. [En línea]. Available: http://vega-bi.blogspot.com/2014/08/nfv-network-functions-virtualization.html. [Último acceso: 27 Septiembre 2017].

6. ANEXOS

ANEXO I. Código de la aplicación desarrollada en OpenDayLight (DVD)

ANEXO II. Código de la función virtualizada de red (DVD)

ANEXO III. Configuración de switch openflow(DVD)

ORDEN DE EMPASTADO