UNIVERSIDAD DE MENDOZA
FACULTAD DE INGENIERIA

TESIS DE
MAESTRÍA EN TELEINFORMÁTICA

Sistema de Acceso y Autenticación en Redes Definidas por Software

Autor: Lic. Peñasco Andrés

Director: Mg. Ing. Méndez Garabetti Miguel

Mendoza-2018

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

Actualmente vivimos en una sociedad digital, donde casi todo está conectado y accesible desde cualquier lugar a través de Internet. Sin embargo, a pesar de su amplia adopción, las redes IP tradicionales de gran envergadura son complejas y muy difíciles de gestionar. Configurar grandes redes según políticas predefinidas, o reconfigurarla para responder a fallas, cambios en la infraestructura y cargas de trabajo, se torna una tarea laboriosa y complicada. Las Redes Definidas por Software (Software Defined Networking, SDN) son un paradigma emergente que promete mejorar las falencias de las redes convencionales, introduciendo la capacidad de programar la red. Para lograrlo, SDN separa el plano de control del plano de datos, promoviendo la centralización del control de la red y convirtiendo a los routers y switches subyacentes en dispositivos de reenvío de datos simples. Este trabajo consistió en realizar una revisión bibliográfica de investigaciones recientes, como así también el análisis de los diferentes tipos de soluciones disponibles, incluyendo controladores y tipos de switches, con el objetivo de desarrollar una aplicación que permite administrar el ingreso y la asignación de privilegios de usuarios a una red IP, mediante el uso de flujos manejados por un controlador SDN. De esta forma, fue posible demostrar que el desarrollo de aplicaciones que gestionan el plano de control, ofrecen soluciones completas capaces de solucionar los problemas y/o debilidades de las redes tradicionales.

CAPÍTULO 1: INTRODUCCIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA Y JUSTIFICACIÓN

Las redes de datos se han convertido en uno de los componentes esenciales de toda red corporativa, siendo de suma importancia que éstas operen de forma eficiente (Stallings, 2006). Por ello, es necesario realizar una adecuada gestión de las redes actuales, considerando la evolución de los sistemas informáticos y las tecnologías emergentes, tales como servicios de computación en la nube (NIST, 2011), sistemas distribuidos (Coulouris, Dollimore, & Kindberg, 2011), Big Data (ISO, 2014), entre otros. En este contexto, las redes de datos tradicionales se enfrentan a una gran cantidad de limitaciones de diseño, impidiendo la rápida adaptación de la red a cambios, como la reacción ante nuevas vulnerabilidades, implementación de nuevos servicios con requisitos especiales, ampliación de la infraestructura o incorporación de nuevos dispositivos.

Para mantener en funcionamiento las redes tradicionales que utilizamos hoy en día, es necesario configurar individualmente cada uno de los dispositivos de red, utilizando comandos de bajo nivel o específicos de cada proveedor. Cuando las redes son de gran dimensión, la complejidad de configuración y mantenimiento aumenta, generando grandes costos operacionales, debido a la especificidad de los conocimientos necesarios para llevar a cabo esta gestión.

Además, los entornos de red tienen que soportar la dinámica de las fallas y adaptarse a los cambios de carga. En este sentido, los mecanismos de reconfiguración y respuesta automática en las redes IP (Internet Protocol) actuales son prácticamente inexistentes. La lógica para la toma de decisiones y el reenvío de tráfico, se agrupan dentro de los dispositivos de red, reduciendo la flexibilidad y obstaculizando la innovación y la

evolución de la infraestructura. Por ejemplo, la transición de IPv4 a IPv6, iniciada hace más de una década y aún en gran parte incompleta (De León & LACNIC, 2014), da testimonio de este desafío, mientras que de hecho IPv6 representaba simplemente una actualización de protocolo.

Las redes tradicionales deterministas, que utilizamos hoy en día, en las que el comportamiento de los dispositivos depende de su configuración previa, necesitan evolucionar a una arquitectura de red dinámica, transformándose en entornos escalables, flexibles, fáciles de gestionar y proteger. En respuesta a estas necesidades, se considera que las Redes Definidas por Software (Software Defined Networking, SDN) (Kreutz, Ramos, Verissimo, Rothenberg, Azodolmolky, & Uhlig, 2015) y su evolución marcan un camino que ofrece soluciones óptimas a las debilidades planteadas.

Las SDN, son un concepto que viene evolucionando hace varios años y se encuentra en constante desarrollo y expansión. Este tipo de redes divide el plano de control (que decide cómo manejar el tráfico de red) del plano de datos (que reenvía el tráfico de acuerdo con las decisiones tomadas por el plano de control), logrando de esta forma infraestructuras programables, automatizadas, adaptables a las necesidades y problemas futuros (Open Networking Foundation, 2012). Al separar el plano de control es posible gestionar la red de forma centralizada, gracias a la incorporación de un controlador que mantiene una visión global de la red y del contenido de la misma, proporcionando la capacidad de manipular el tráfico de información según sea necesario. De esta manera se permite programar directamente sobre la arquitectura SDN, utilizando módulos de software instalados en el controlador, agilizando los procesos de configuración.

Además, las arquitecturas SDN pueden ser implementadas bajo estándares abiertos, de modo que no dependen de dispositivos de fabricantes específicos o protocolos propietarios. La ONF (Open Networking Foundation, 2016), ha definido el primer estándar abierto, denominado OpenFlow (McKeown, Anderson, Balakrishnan, Guru, & Peterson, 2008). OpenFlow es un protocolo que se encuentra en continuo desarrollo y permite manejar directamente el plano de reenvío de datos de los dispositivos de red, ya sean físicos o virtuales (Open Networking Foundation, 2012).

Como sucede en las redes de datos tradicionales, las SDN no se encuentran libres de inconvenientes de seguridad. En este aspecto uno de los grandes desafíos es el control de acceso. La necesidad de acceder a las redes corporativas utilizando distintos dispositivos, genera la aparición de innumerables y nuevos puntos débiles (Esmoris, 2010). Esto requiere que la red le garantice condiciones de seguridad y confidencialidad para el proteger el tráfico de datos que se intercambian.

Con el objetivo de abordar los inconvenientes mencionados en los párrafos anteriores y aprovechar las ventajas que presenta la arquitectura SDN, es que en este trabajo se realiza un estudio de la misma, desarrollando un prototipo de aplicación, capaz de gestionar el control de acceso de usuarios en una arquitectura SDN. Esta propuesta fue publicada en el XIX Workshop de Investigadores en Ciencias de la Computación (Peñasco, 2017).

Para llevar a cabo la investigación se evaluaron los últimos avances citados en la literatura, tal como los trabajos de (Dangovas & Kuliesius, SDN Enhanced Campus Network Authentication and Access Control System, 2016) y (Dangovas & Kuliesius, SDN-Driven Authentication and Access Control System, 2014), donde los autores proponen una solución

para el control de acceso a redes en una arquitectura SDN híbrida, utilizando switches virtuales y el protocolo de autenticación RADIUS (Willens, Rigney, Rubens, & Simpson, 2000). Además, se realiza un análisis e investigación de diferentes tipos de soluciones SDN disponibles, incluyendo controladores y tipos de switches, como así también la comunicación y configuración de los mismos a través del estándar abierto OpenFlow en sus diferentes versiones.

1.2 OBJETIVOS

El presente proyecto, se centró en la investigación y desarrollo de un método de acceso y autenticación para Redes Definidas por Software capaz de resolver determinados problemas presentes en las redes de datos tradicionales.

Para ello se llevaron a cabo las siguientes actividades:

1. Se estudiaron los diferentes tipos de controladores y dispositivos SDN existentes.
2. Se analizaron los últimos desarrollos e investigaciones realizadas para control de acceso sobre SDN.
3. Se implementó una SDN.
4. Se configuró y gestionó de forma centralizada los dispositivos de la SDN implementada.
5. Se desarrolló un prototipo de aplicación para el control de acceso de usuarios a una arquitectura SDN.
6. Se documentaron las características, arquitectura, ventajas y desventajas del protocolo OpenFlow y las SDN.
7. Se evaluó el alcance de la solución y su viabilidad.

1.3 ORGANIZACIÓN DE LA TESIS

La presente tesis se ha abordado en cuatro capítulos. El primer capítulo desarrolla la justificación del proyecto, el planteamiento del problema, objetivos y organización de la tesis. Posteriormente el segundo capítulo aborda los conceptos teóricos necesarios para poder conocer y comprender las Redes Definidas por Software, su arquitectura, componentes, características generales y tipos de dispositivos SDN. Además en este capítulo se realiza una revisión del protocolo OpenFlow y se efectúa un excautivo análisis de controladores SDN disponibles en la actualidad y que se adaptan al proyecto. Dicho análisis es llevado a cabo en el simulador Mininet.

El tercer capítulo está dedicado al desarrollo de la aplicación propuesta. Es un apartado netamente práctico, en él se puede apreciar el escenario implementado y cada uno de los componentes utilizados. Además del código fuente y el funcionamiento de cada una de las clases y métodos desarrollados. Al finalizar el capítulo se exponen las pruebas y resultados de la implementación.

Para finalizar el cuarto capítulo presenta las conclusiones de trabajo.

CAPÍTULO 2: MARCO TEÓRICO Y ANTECEDENTES

2.1 INTRODUCCIÓN

Este capítulo desarrolla un panorama completo de las SDN, con el objetivo brindar una base que sirva como referencia para interpretar los siguientes capítulos. Se abordan conceptos de la arquitectura SDN, sus ventajas y desventajas, el protocolo OpenFlow y sus componentes finalizando con un resumen de las características de los controladores que se adaptan a este trabajo.

Como se mencionó en el capítulo anterior, las SDN surgen como alternativa de las redes de datos tradicionales, en respuesta a la necesidad de mayor escalabilidad y manejo de tráfico más personalizable y eficiente. Ya que actualmente, se necesita que las redes de datos se ajusten y respondan de forma dinámica bajo políticas actualizadas. Es decir, se necesita poder implementar y ejecutar con rapidez nuevas aplicaciones dentro y por encima de las redes; este ambicioso desafío es afrontado por las SDN.

2.2 SDN (SOFTWARE DEFINED NETWORKING)

El fundamento de la arquitectura SDN se basa en la centralización de red, para ello divide el plano de control del plano de datos. Permitiendo que el control de red sea directamente programable y que la infraestructura subyacente sea abstraída para aplicaciones y servicios de red. Esto lleva a que se convierta en una solución de red dinámica, rentable y adaptable, ideal para las aplicaciones actuales. En este contexto el protocolo OpenFlow es uno de los componentes fundamentales para la construcción de soluciones SDN. Las principales características de la arquitectura SDN son:

- **Directamente programable:** El control de la red es directamente programable debido a que las funciones de reenvío de datos fueron desacopladas.
- **Gestión centralizada:** La inteligencia de la red está centralizada en un software llamado controlador SDN, que mantienen una visión global de la red, haciendo parecer una gran red de switches como un único switch lógico.
- **Ágil:** Al tener monitoreo constante, permite al administrador de red o a las aplicaciones hacer cambios dinámicos en los flujos de red según las necesidades.
- **Programación configurada:** SDN permite a los administradores de red configurar, gestionar, proteger y optimizar los recursos de red rápidamente a través de aplicaciones de SDN dinámicas, que pueden ser escritas por ellos mismos, ya que no dependen de software propietario.
- **Basada en estándares abiertos y de proveedor neutral:** Como se implementa a través de estándares abiertos, SDN simplifica el diseño de la red y la operación porque integra un solo protocolo para todos los fabricantes.

La arquitectura SDN está compuesta por 3 capas como se aprecia en la Figura 1:

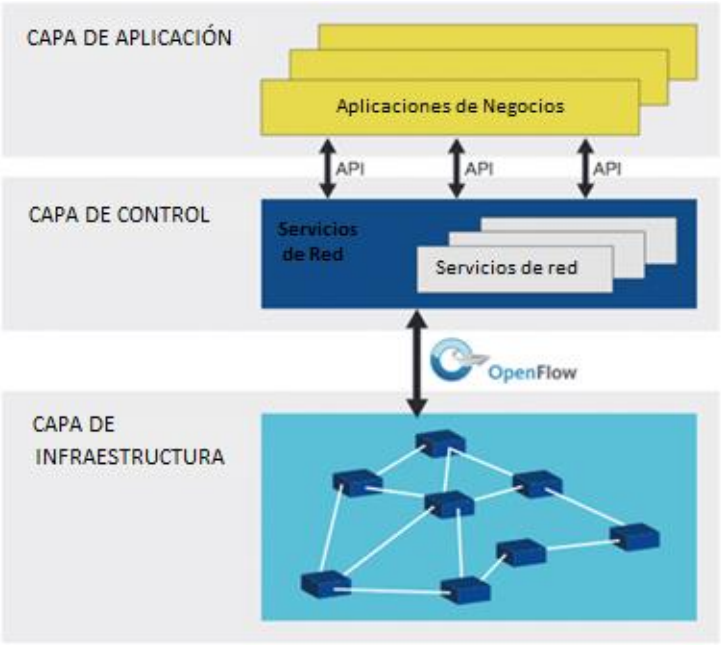

Figura 1 : Arquitectura SDN.
Fuente: (Open Networking Foundation, 2016).

- **Capa de aplicación:** Permite crear aplicaciones para automatizar tareas de configuración, provisión y despliegue de nuevos servicios en la red. Esta capa se comunica con la capa de control por medio de una API[1] denominada NorthBound. Así, logra tener una visión global de las condiciones actuales de la red, con el fin de mejorar la transmisión de datos y la toma de decisiones.
- **Capa de control:** Por medio de esta capa, se crea una vista centralizada de la topología de la red de datos, la cual permite gestionar el flujo de datos en la capa de infraestructura por medio del protocolo OpenFlow, también brinda un API denominada SouthBound para que desde la capa de aplicación se puedan programar flujos y retroalimentar a la aplicación, con información de la topología de red o el tráfico de paquetes.

[1] **API:** Interfaz de programación de aplicaciones, del inglés: Application Programming Interface, es un conjunto métodos y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

- **Capa de infraestructura:** Está conformada por elementos de red, los cuales son gestionados por medio del protocolo OpenFlow, permitiendo simplificar la configuración al administrador de red.

Como se mencionó anteriormente las capas de control y aplicación se comunican mediante el uso de dos APIs: la **NorthBound API** y la **SouthBound API**.

- **NorthBound API:** es la encargada de interpretar las aplicaciones y establecer la comunicación con el controlador.
- **SouthBound API:** se encarga de comunicar el hardware con el controlador de manera transparente e independiente del elemento hardware que se encuentra debajo.

Se puede observar en la Figura 2 como funcionan dichas API entre el plano de control y de aplicación, teniendo al controlador en un punto intermedio.

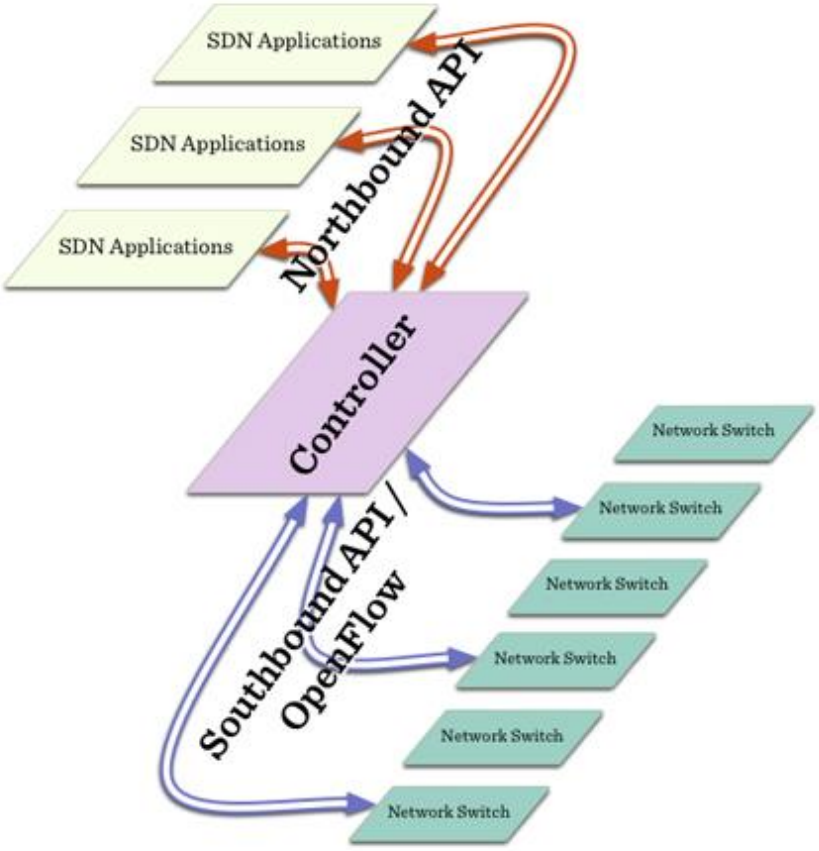

Figura 2: Northbound API y Southbound API.
Fuente: (Ferro, 2012).

2.3 VENTAJAS DE LA ARQUITECTURA SDN

En comparación a los esquemas de red tradicionales, la implementación de una SDN posee grandes beneficios, los mismos fueron extraídos de (IETF, 2014) y de se detallan a continuación:

- **Administración centralizada de la red:** a través de un solo dispositivo denominado controlador SDN, en el cual se programan todas las reglas para reenvío de tráfico a través de la red.
- **Entornos con dispositivos de diferentes fabricantes:** si bien en las redes tradicionales esto también es posible, el problema se genera cuando los equipos utilizan protocolos propietarios. En

cambio en las SDN los equipos solo deben soportar el protocolo OpenFlow. Los administradores pueden utilizar herramientas de orquestación y gestión basadas en SDN para implementar, configurar y actualizar rápidamente dispositivos en toda la red. Sin necesidad de esperar a que los fabricantes de sus equipos introduzcan nuevas tecnologías. Como resultado se obtiene una independencia de los vendedores de equipos de conectividad.

- **Reducción de la complejidad a través de la automatización:** las SDN ofrecen un entorno flexible de automatización y gestión de la red, el cual permite desarrollar herramientas que automatizan muchas tareas de gestión que en la actualidad se realizan manualmente. Estas herramientas de automatización reducen la sobrecarga operacional, disminuyendo posibles fallos de red generados por errores humanos, dando soporte a los nuevos modelos de provisión de servicios de IT, como computación en la nube.
- **Mayor facilidad de innovación:** la adopción de SDN acelera la innovación tecnológica, permitiendo a los operadores de redes reprogramarlas en tiempo real, para satisfacer las necesidades específicas y los requisitos de los usuarios a medida que surjan.
- **Facilidad de programación**: el controlador SDN facilita la programación por parte del administrador mediante la utilización de lenguajes de programación comunes y conocidos por la mayoría de personas que trabajan en el medio.
- **Aumento de la fiabilidad y la seguridad de la red:** SDN permite definir configuraciones y políticas de alto nivel, que luego se traducen a la infraestructura a través del protocolo OpenFlow. Una arquitectura SDN basada en OpenFlow elimina la necesidad de configurar individualmente los dispositivos de red cada vez que se

agrega o mueve un punto final, lo que reduce la probabilidad de fallos de red debido a inconsistencias de configuración o políticas.

Debido a que el controlador SDN proporciona visibilidad y control completo sobre la red, las aplicaciones pueden garantizar que políticas como control de acceso, ingeniería de tráfico y calidad del servicio, se apliquen consistentemente a través de las infraestructuras de red cableada e inalámbrica.

- **Control de red granular:** el modelo de control basado en flujo de OpenFlow permite aplicar políticas a un nivel muy granular, incluyendo la sesión, el usuario, el dispositivo y los niveles de aplicación, de una manera altamente abstraída y automatizada. Este control permite a los operadores de entornos cloud soportar multitenancia[2] (Gomezcoello Yépez, 2017) mientras mantiene el aislamiento del tráfico, la seguridad y la administración dinámica de los recursos, cuando los clientes comparten la misma infraestructura.
- **Mejor experiencia de usuario:** Al centralizar el control de red y hacer que la información del estado de la misma, esté disponible para aplicaciones de nivel superior, una infraestructura SDN puede adaptarse mejor a las necesidades de los usuarios dinámicos. Por ejemplo, un proveedor de servicios de internet podría introducir un servicio de videoconferencia que ofrece a los suscriptores la mayor resolución posible de una manera automatizada y transparente. Hoy en día, los usuarios deben seleccionar explícitamente un ajuste de resolución, que la red puede o no ser capaz de soportar, dando lugar a retrasos e interrupciones que degradan la

[2] **Multitenancia:** Corresponde a un principio de arquitectura de software en la cual una sola instancia de la aplicación se ejecuta en el servidor, pero sirviendo a múltiples clientes u organizaciones.

experiencia del usuario. Con SDN basado en OpenFlow, la aplicación de vídeo sería capaz de detectar el ancho de banda disponible en la red en tiempo real y ajustar automáticamente la resolución de vídeo en consecuencia.

2.4 PROTOCOLO OPENFLOW

Según la definición dada por ONF (Open Networking Foundation, 2012), OpenFlow es la primera interfaz de comunicaciones estándar definida entre el plano de control y el plano de reenvió de datos de una arquitectura SDN. OpenFlow permite el acceso directo y la manipulación del plano de reenvío de datos de dispositivos de red como switches y enrutadores, tanto físicos como virtuales.

En las líneas siguientes se presenta un resumen del protocolo OpenFlow en la versión 1.5, extraído de la documentación brindada por la ONF (Open Networking Foundation, 2014).

2.4.1 SWITCH OPENFLOW

Un switch OpenFlow consiste en una o varias tablas de flujo y uno o más canales de comunicación OpenFlow hacia un controlador externo. Se observa, en la Figura 3, como un switch se comunica con el controlador y el controlador lo gestiona a través del protocolo de OpenFlow.

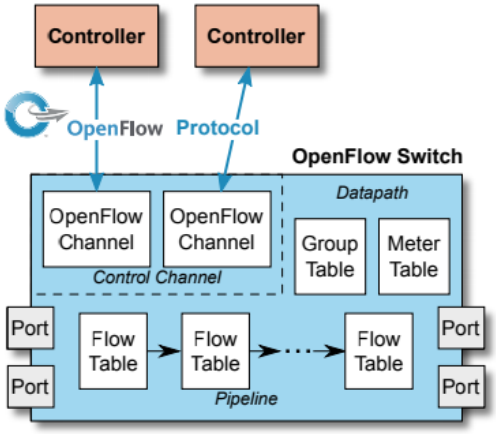

Figura 3 : Componentes de un switch OpenFlow.
Fuente: (Open Networking Foundation, 2014).

El switch OpenFlow solo proporciona una arquitectura lógica común que permite la comunicación entre un switch y un controlador a través de un canal seguro. Todo dispositivo que tenga soporte para OpenFlow podrá ser controlado por el software controlador sin importar el fabricante del equipo de red. El uso de OpenFlow proporciona:

- Manejo más fácil y eficiente de una red.
- Interacción dinámica de la red, es decir adaptabilidad.
- Separación del tráfico de datos del de control.
- Innovación tecnológica.
- Plataforma abierta de programación.

Cada tabla de flujo en el conmutador contiene un conjunto de entradas; cada entrada de flujo consiste en campos de coincidencia, contadores y un conjunto de instrucciones para aplicar a paquetes coincidentes.

La inteligencia de la red se encuentra en el controlador, éste puede agregar, actualizar y eliminar entradas de flujo en las tablas del switch, tanto de manera reactiva (en respuesta a los paquetes) como de manera proactiva (mediante reglas programadas con anterioridad).

OpenFlow cuenta con dos tipos de switches, los switches OpenFlow y los switches híbridos. Los switches OpenFlow solo soportan operaciones OpenFlow, mientras que los híbridos tienen la posibilidad de seleccionar los puertos que van a trabajar en modo OpenFlow y dejar el resto para trabajar en modo normal.

2.4.2 PUERTOS OPENFLOW

Los puertos OpenFlow son interfaces de red para el envío de paquetes entre el procesamiento de OpenFlow y el resto de la red. Los switches OpenFlow se conectan lógicamente entre sí a través de sus puertos OpenFlow, un paquete puede reenviarse de un switch a otro switch, solo a través de un puerto OpenFlow de salida en el primer switch y un puerto OpenFlow de entrada en el segundo switch.

Un switch OpenFlow define tres tipos de puertos OpenFlow: puertos físicos, puertos lógicos y puertos reservados:

- **Puertos físicos:** son puertos definidos por el switch que corresponden a una interfaz de hardware del mismo. Por ejemplo, en un conmutador Ethernet, los puertos físicos se correlacionan uno a uno con las interfaces Ethernet. En algunas implementaciones, el switch OpenFlow puede ser virtualizado sobre el hardware. En esos casos, un puerto físico OpenFlow puede representar una porción virtual de la interfaz de hardware correspondiente al switch.
- **Puertos lógicos:** no tienen correspondencia a un puerto físico como tal y son utilizados en métodos non-OpenFlow (túneles, agregación). Un puerto lógico puede estar relacionado a varios puertos físicos. La diferencia fundamental entre un puerto lógico de

un físico, es que el primero tiene un campo de metadata extra llamado *Tunnel-ID.*

- **Puertos reservados:** son puertos para acciones específicas, como por ejemplo el envío de información al controlador, multidifusión, etc. Algunos de estos puertos reservados son requeridos en comunicaciones OpenFlow, entre estos el puerto `ALL`, puerto *controller*, puerto *table*, puerto *in_port* y el puerto *any*.

2.4.3 CAMBIOS DE ESTADO DE PUERTOS

Es posible agregar o quitar puertos de un switch OpenFlow en cualquier momento. Por ejemplo, el switch puede cambiar el estado del puerto si el mismo se encuentra sin actividad. Cualquier cambio en el estado del puerto o en la configuración debe comunicarse al controlador OpenFlow.

La adición, modificación o extracción de puertos no modifica el contenido de las tablas de flujo, por lo tanto las entradas de flujo que hacen referencia a esos puertos no se modifican ni eliminan, y los paquetes enviados a puertos inexistentes se descartan.

Si se elimina un puerto y se reutiliza su número para un puerto físico o lógico distinto, todas las entradas de flujo seguirán haciendo referencia a ese número de puerto, pudiendo reenviar tráfico por el mismo, posiblemente con resultados no deseados. En este caso es necesario solicitarle al controlador la limpieza de las entradas de flujo que referencien a dicho puerto.

2.4.4 TABLAS Y ENTRADAS DE FLUJO OPENFLOW

Como se mencionó en la sección 2.4.1 un switch OpenFlow está compuesto de al menos una tabla, que a su vez está compuesta de múltiples entradas de flujo. Cada tabla es gestionada por el controlador

SDN y son utilizadas para llevar a cabo el direccionamiento de flujos de datos mediante el proceso pipeline[3] (Sección 2.5).

Existen tres tipos de tablas:

- **Tabla de flujo:** permite relacionar los paquetes entrantes con un flujo y el conjunto de acciones específicas que debe llevar a cabo. Puede existir una o más tablas de flujo.
- **Tabla de grupo:** consiste de un grupo de entradas. Un grupo representa un conjunto de acciones para multidifusión u operaciones de reenvío más complejas (multipath[4], link aggregation[5]). Permite que acciones de salida comunes sean tratadas de forma más eficiente.
- **Tabla Meter:** puede desencadenar algunas acciones de performance de un flujo, como calidad de servicio.

Cada entrada de la tabla de flujo está formada por siete componentes, como se visualiza en la Figura 4.

Figura 4 : Entradas de Flujo OpenFlow 1.5.
Fuente: (Open Networking Foundation, 2014).

- **Match Field:** este campo establece los prerrequisitos para que a un paquete se le aplique una serie de instrucciones, los campos

[3] **Pipeline o proceso pipeline:** define como los paquetes de datos van a interactuar con las tablas de flujo.

[4] **Multipath:** o MPTCP, tiene como objetivo permitir que una conexión de Protocolo de control de transmisión (TCP) utilice múltiples rutas para maximizar el uso de recursos y aumentar la redundancia.

[5] **Link aggregation**: o LACP (Link Aggregation Control Protocol), es un protocolo para unir varias interfaces de red de un dispositivo en una sola virtual, con el objetivo de tener mayor velocidad y tolerancia a fallos.

que se pueden evaluar son los encabezados de paquetes, puertos lógicos, puertos físicos y campos de metadata especificados en una tabla anterior.

- **Priority:** establece un valor que determina la prioridad de la entrada de flujo.
- **Counters:** se actualiza cada vez que es procesado un paquete.
- **Instructions:** contienen una serie de acciones, que serán ejecutadas si los paquetes son relacionados con alguna entrada de flujo de la tabla.
- **Timeouts:** tiempo máximo que permanece una entrada de flujo, ya sea porque no se utiliza durante un espacio de tiempo o simplemente porque se creó con un tiempo determinado.
- **Cookies:** es un valor seleccionado por el controlador para filtrar las estadísticas o modificaciones que se le hace a un flujo.
- **Flags:** son valores que alteran la forma en que se gestionan las entradas de flujo, por ejemplo, el indicador OFPFF_SEND_FLOW_REM activa los mensajes eliminados por el flujo para esa entrada de flujo.

Una entrada se identifica en la tabla de flujo por el conjunto de los dos primeros campos: *Match Fields* + *Priority.* Para remover entradas de flujo existen dos métodos:

- Por el requerimiento del controlador.
- Por los mecanismos de expiración de flujos del switch.

Cada entrada de datos tiene un valor de inactividad *idle_timeout* y un valor de tiempo de espera *hard_timeout.* Un campo *hard_timeout* diferente de cero causa que la entrada de flujo sea removida después de un tiempo, independientemente de si existen paquetes que hayan sido

relacionados. Un campo *idle_timoeout* diferente de cero permite que la entrada de flujo sea removida cuando no existen paquetes relacionados.

2.5 PROCESO PIPELINE

Un switch OpenFlow puede tener una o varias tablas de flujo que son organizadas en un pipeline, como se puede observar en la Figura 5. El pipeline describe el proceso de decisión y ejecución de un paquete que ingresa a un switch OpenFlow.

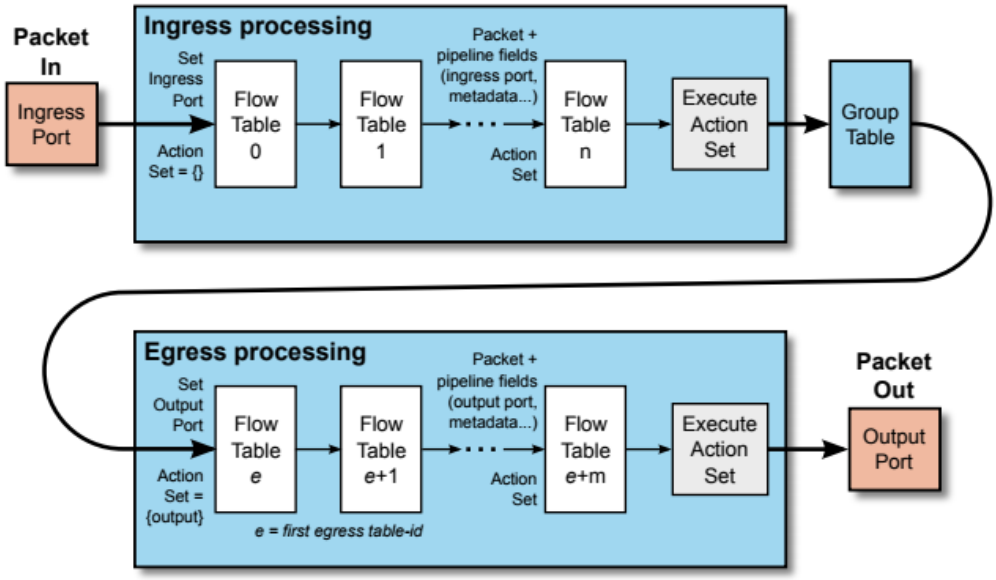

Figura 5 : Flujo de paquetes proceso pipeline OpenFlow 1.5.
Fuente: (Open Networking Foundation, 2014).

Las tablas de flujo, están numeradas de forma secuencial empezando desde 0. El proceso de pipeline consiste en ir evaluando las tablas de flujo secuencialmente de forma ascendente y siempre comenzando desde la tabla de flujo 0, una vez el paquete ingresa a la tabla de flujo, el switch busca en las entradas de flujos coincidencias entre el *match Field* y el paquete que está procesando, si no encuentra coincidencias el paquete pasa a la siguiente tabla y repite el mismo procedimiento hasta que llega a la tabla final y envía el paquete al controlador para que retorne una

acción. En el caso que encuentre alguna coincidencia (por ejemplo número de puerto, VLAN id, Ether type, dirección IP de origen o destino, etc.) se ejecutan las acciones de la entrada de flujo y el proceso termina. Si la entrada de flujo contiene en el campo *instructions* la acción *go_to_table*, el proceso sigue en busca de más entradas de flujo que coincidan con el campo `match`. Vale la pena aclarar que la acción *go_to_table* solo funciona si el número de la tabla es mayor al que se está evaluando.

2.6 CANAL OPENFLOW

El canal OpenFlow (OpenFlow Channel) Figura 3, es la interfaz que conecta el switch OpenFlow con el controlador. A través de esta interfaz el controlador configura y gestiona el switch. Un switch OpenFlow puede tener múltiples canales OpenFlow cada uno con un controlador diferente. Cada switch admite un único canal OpenFlow con un único controlador o múltiples canales OpenFlow que permiten a múltiples controladores compartir la gestión del switch. La comunicación a través de canal OpenFlow suele estar cifrada usando TLS (Transport Layer Security) (IETF, 2008), no siendo este un requisito obligatorio.

2.7 COMUNICACIÓN DEL PROTOCOLO OPENFLOW

El protocolo OpenFlow tiene tres clases de mensajes descritos a continuación:

a) **Controller-to-Switch:** son mensajes iniciados por el controlador y pueden o no tener respuesta del switch, entre ellos están:
   - **Features:** se usa para establecer la conexión con el switch mediante el protocolo TLS, con esta solicitud el switch proporciona al controlador los parámetros para establecer la conexión.

- **Configuration:** mensaje con el cual el controlador pregunta los parámetros de configuración del switch.
- **Modify-State:** el propósito principal de este mensaje es agregar, quitar o modificar los flujos del switch.
- **Read-State:** este mensaje es utilizado para recolectar estadísticas de las tablas de flujo, puertos y flujo de entrada en el switch.
- **Packet-out:** con este mensaje el controlador le indica al switch el puerto por el cual debe direccionar un grupo de paquetes determinados.
- **Barrier:** es usado por el controlador para asegurar que las dependencias de un mensaje se han cumplido, o para recibir notificaciones de operaciones terminadas.
- **Role-Request:** es utilizado por los controladores para establecer un rol o consultar su *ID*. Esto es útil cuando el switch se conecta a varios controladores

b) **Asynchronous:** son mensajes generados por el switch, para la notificación de llegada de paquetes que no coinciden con ningún `match`, cuando ocurre un cambio en el dispositivo o cuando se presentan errores.

- **Packet-in:** este mensaje es enviado por el switch al controlador, informando que entró un paquete que no coincide con ningún `match`. El controlador analiza el paquete y responde con un *Send-Packet*.
- **Flow-Removed:** el switch notifica al controlador que un flujo fue eliminado porque se terminó el tiempo para el cual estaba programado.
- **Port Status:** el switch notifica al controlador que un puerto cambio su estado.

- **Role Status:** el switch informa al controlador de un cambio de en su rol. Cuando un nuevo controlador se elige como maestro, se espera que el switch envíe mensajes de estado de función al controlador maestro anterior
- **Controller Status:** el switch informa al controlador cuando cambia el estado de un canal OpenFlow. Estos mensajes son enviados a todos los controladores.
- **Flow Monitor:** el switch informa al controlador de un cambio en una tabla de flujo. Un controlador puede definir un conjunto de monitores para realizar un seguimiento de los cambios en las tablas de flujo.

c) **Symmetric:** son mensajes enviados bidireccionalmente, entre los mensajes se encuentran:

- **Hello:** son mensajes intercambiados entre el controlador y el switch, se utilizan para establecer la conexión entre los dos.
- **Echo:** *request/reply* este mensaje es utilizado para confirmar que hay conexión entre el controlador y el switch, y sirve para hacer medidas de latencia y ancho de banda.
- **Experimenter:** permite dar funcionalidades extra al Switch, pero es usado principalmente para futuras versiones de OpenFlow.
- **Error:** los mensajes de error son utilizados por el switch o el controlador para notificar los problemas de conexión. Son utilizados principalmente por el switch para indicar un fallo de una petición iniciada por el controlador.

2.8 CONTROLADORES SDN

Uno de los elementos principales de la arquitectura SDN es el controlador, núcleo de la arquitectura del cual depende el comportamiento de la red. Actualmente existen diferentes alternativas a la hora de seleccionar un

controlador, según si son controladores comerciales, o bien controladores de código abierto. A continuación se presentan las características de los controladores de código abierto que se ajustan a las necesidades del proyecto.

2.8.1 BEACON

Beacon (Erickson, The Beacon OpenFlow Controller, 2012) es un controlador OpenFlow de código abierto basado en Java creado en 2010. Ha sido ampliamente utilizado para la enseñanza, la investigación y como base para el desarrollo de Floodlight (sección 2.8.4). Fue creado con tres objetivos principales: mejorar la productividad de los desarrolladores, proporcionar la capacidad de ejecución para iniciar y detener aplicaciones existentes y lograr alto rendimiento.

Gracias a estar desarrollado en Java es multiplataforma y ofrece una serie de paquetes que proveen funciones avanzadas, como la detección de la topología, la determinación de la ruta más corta a nivel de capa enlace, una interfaz web de usuario, entre otras, que están habilitadas en la distribución por defecto.

Algunas de las características más relevantes son:

- **Código abierto:** está licenciado bajo una combinación de la licencia GPL11 v.2 y la Licencia de la Universidad de Stanford.
- **Dinámico:** los paquetes de código en Beacon pueden ser inicializados, detenidos, actualizados e instalados en tiempo de ejecución, sin interrumpir a otros paquetes.
- **Estable:** Beacon ha sido utilizado en muchos proyectos de investigación y algunas implementaciones de prueba.
- **Desarrollo Ágil:** es fácil de descargar y ejecutar. Java y Eclipse simplifican el desarrollo y la depuración de sus aplicaciones.

- **Rápido:** debido a su característica de multiproceso.
- **Interfaz web de usuario:** posee una interfaz web amigable e intuitiva.
- **Documentación:** incluye tutoriales y guías para ayudar a conseguir un funcionamiento productivo.

2.8.1.1 API Y APLICACIONES

La Interfaz de Programación de Aplicaciones de Beacon está diseñada para ser simple y no impone ninguna restricción. Incluye la biblioteca OpenFlowJ[6] para trabajar con mensajes OpenFlow.

La interacción con los switches OpenFlow se produce a través de la interfaz *IBeaconProvider*. Al agregar o eliminar los switches se envían notificaciones a los registros de escucha a través de la interfaz *IOFInitializerListener* y para la recepción de los diferentes tipos de mensajes OpenFlow se lo realiza mediante *IOFMessageListener*. La explicación detallada de todas las interfaces de Beacon se encuentra en su wiki en (Erickson, Basic Guides, 2012).

La Figura 6 muestra aplicaciones de referencia incluidas por Beacon:

[6] **OpenFlowJ:** es una implementación Java de la especificación OpenFlow 1.0 orientada a objetos.

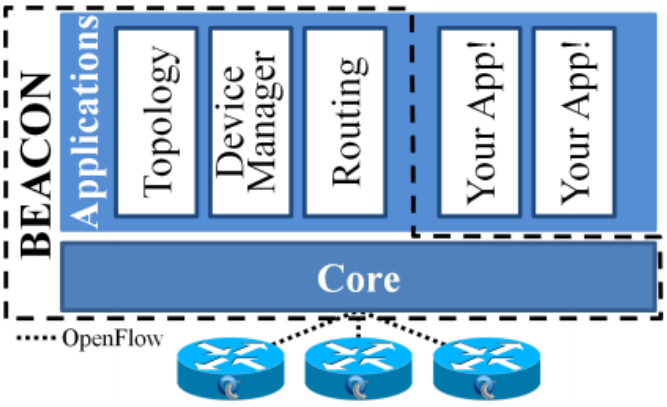

Figura 6 : Aplicaciones controlador Beacon.
Fuente: (Erickson, The Beacon OpenFlow Controller, 2012).

- **Topology:** descubre los enlaces entre los switches OpenFlow conectados. Su interfaz (*ITopology*) permite la creación de una lista de enlaces cuando se conectan a o desconectan dispositivos de la red.
- **Device Manager:** provee la interfaz *IDeviceManager* para realizar la detección de dispositivos y registrar eventos recibidos al agregar, actualizar o remover un dispositivo específico.
- **Routing:** proporciona la ruta de acceso más corta entre dispositivos de la red a nivel de capa de enlace. Para lograrlo utiliza la interfaz *IRoutingEngine.*
- **Web:** proporciona una interfaz web de usuario para Beacon. La aplicación web depende de la interfaz *IWebManageable*.

2.8.2 OPENDAYLIGHT

OpenDayLight (The Linux Fundation Projects, 2017) es un proyecto desarrollado por la Linux Fundation y va más allá de las implementaciones SDN con OpenFlow. Contribuyen en el desarrollo un gran número de empresas y fabricantes de soluciones de red como Cisco, Citrix, HP, Intel,

Brocade, etc. Por ello este proyecto además de ser compatible con OpenFlow, tiene la intención de ser compatible con la mayoría de los protocolos de los diferentes fabricantes, para poder controlar no sólo dispositivos OpenFlow.

La arquitectura base es parecida a las del resto de controladores. Se puede acceder al controlador mediante una API REST que permite interactuar con el mismo. En la Figura 7 se puede observar su arquitectura:

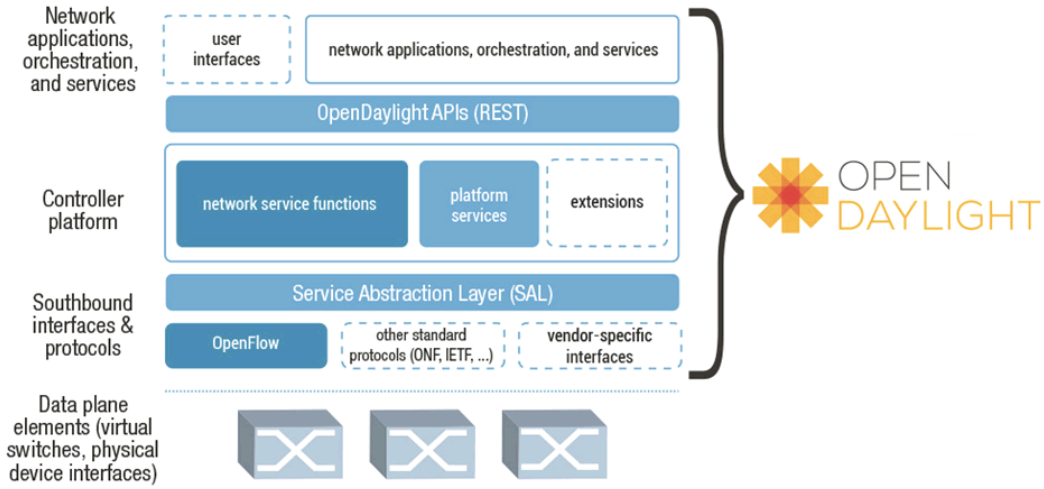

Figura 7 : Arquitectura Opendaylight.
Fuente: (Morgan, 2013).

Además de lo anterior, Opendaylight también ofrece soporte para OpenStack[7] de forma que se puede integrar en esta arquitectura facilitando así la gestión de la red en entornos cloud.

[7] **OpenStack:** es un proyecto de computación en la nube para proporcionar una infraestructura como servicio (IasS).

2.8.3 TREMA

Trema es un framework completo y fácil de usar para programar un controlador OpenFlow, en lenguajes Ruby y C. El código fuente de Trema incluye módulos básicos y bibliotecas que funcionan como interfaz con los switches OpenFlow. Es importante mencionar que Trema incluye un emulador de red OpenFlow y no es necesario disponer de switches OpenFlow o de equipos de usuario final para probar las aplicaciones del controlador. Este entorno autónomo ayuda a agilizar todo el proceso de desarrollo.

2.8.3.1 CARACTERÍSTICAS:

- Ejecución en una red virtualizada, es decir, permite el desarrollo de un entorno de red de pruebas sin necesidad de switches físicos. Además de poder utilizarlo en un entorno físico y en producción.
- Definición y construcción de topologías virtuales arbitrarias.
- Soporte GNU/Linux
- Requiere Ruby 2.0.0 y OpenvSwitch para su funcionamiento.
- Posee licencia GPLv2 y una comunidad abierta.

2.8.3.2 ARQUITECTURA DE TREMA

La arquitectura está conformada por módulos del núcleo del controlador, componentes de configuración y módulos de soporte adicionales. Como se mencionó anteriormente Trema es una plataforma que provee un emulador de red y herramientas de depuración. Esto se puede observar en la Figura 8; un controlador OpenFlow basado en Trema es en sí, un entorno de pruebas, módulos de usuario y un emulador de red integrado. Todo en la misma plataforma del controlador OpenFlow.

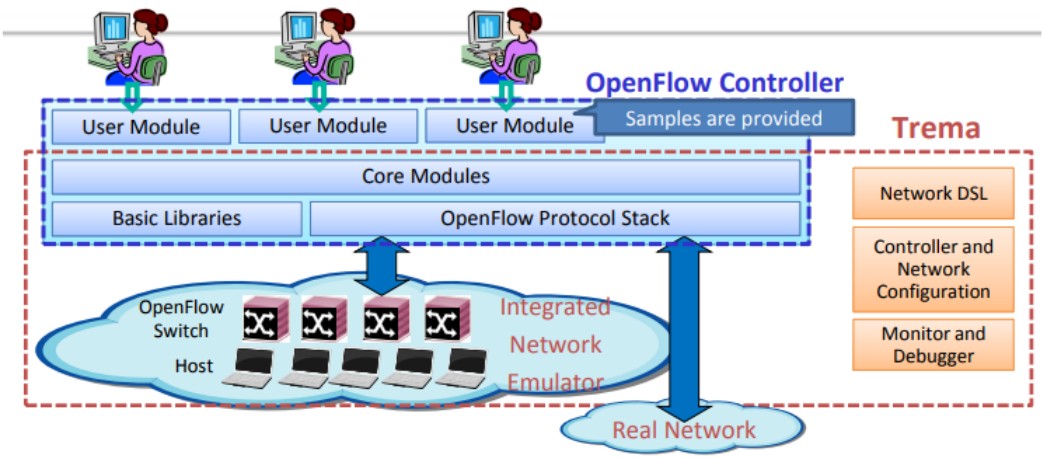

Figura 8 : Arquitectura de Trema.

Fuente: (Dietz, 2012).

2.8.3.3 MÓDULOS

Los módulos de Trema pueden ser escritos en C o Ruby, debido a que se dispone de una API para los dos lenguajes. Es importante destacar que las dos API provistas, son totalmente compatibles y únicamente depende de la elección del programador. Además se incluyen ejemplos útiles que pueden utilizase como guías a la hora de programar.

Los principales módulos provistos son:

- **Learning Switch:** funciona como un switch de capa 2.
- **Routing Switch:** funciona como un switch de capa 3.
- **Topology Discovery:** realiza el descubrimiento de la topología.

Además, es importante considerar el módulo de depuración *Tremashark* el cual recopila y monitorea eventos OpenFlow.

2.8.4 FLOODLIGHT

Floodlight (Project Floodlight, 2016) es un controlador basado en Java desarrollado bajo licencia Apache. Es apoyado por una gran comunidad de desarrolladores, incluyendo ingenieros de Big Switch Networks. Está diseñado para trabajar con una gran cantidad de switches, routers, switches virtuales y puntos de acceso que soporten el estándar OpenFlow. Además, está compuesto de una colección de aplicaciones que proveen funcionalidades para instalar, controlar e investigar una red OpenFlow.

La Figura 9 muestra la relación entre el controlador Floodlight, las aplicaciones creadas como módulos Java (compilados con Floodlight) y las aplicaciones creadas sobre la API REST.

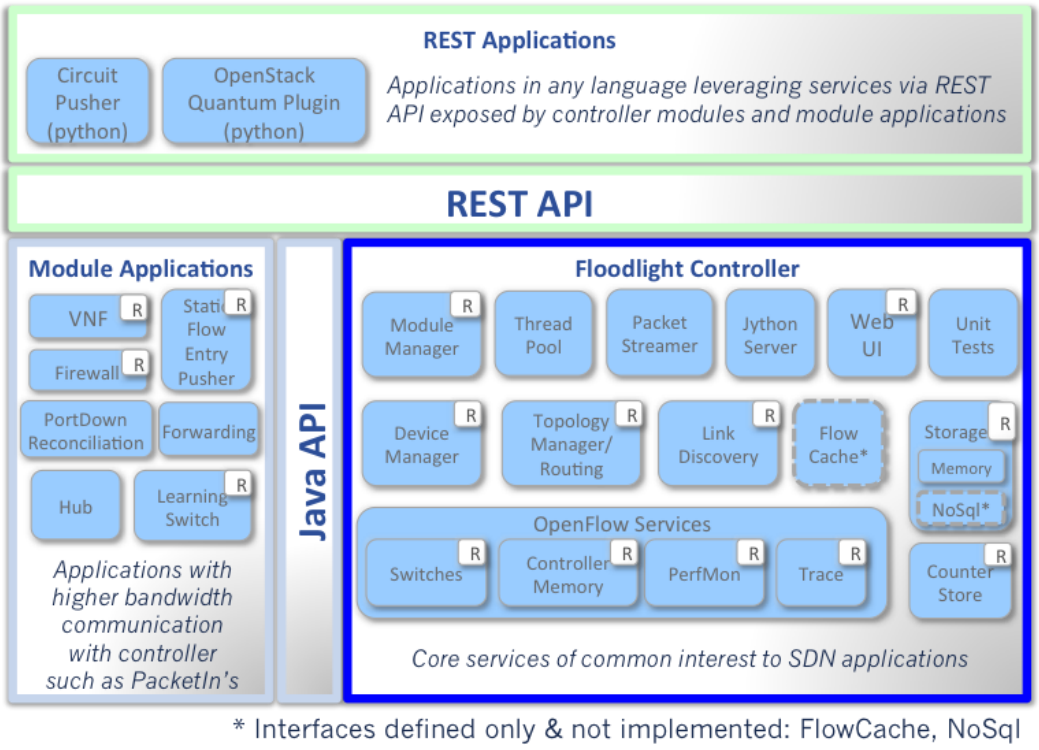

Figura 9 : Arquitectura Floodligth.
Fuente: (Project Floodlight, 2016).

2.8.4.1 MÓDULOS DE FLOODLIGHT

Floodlight implementa los módulos en dos grupos diferentes. Los cuales se denominan módulos de Aplicación y módulos de Controlador.

2.8.4.2 MÓDULOS DE APLICACIÓN

Los módulos de aplicación (Project Floodlight, 2012) implementan funcionalidades específicas, como la detección de enlaces fuera de servicio dentro de la red, virtualización de un switch de capa de enlace o cumplimiento de reglas ACL y Firewall. Es importante aclarar que los módulos se cargan al iniciar el controlador. Dentro de estos módulos se encuentran:

- **VirtualNetworkFilter:** permite virtualizar un switch a nivel de capa de enlace, logrando crear varias redes lógicas.
- **Forwarding:** Realiza el envío paquetes entre dos dispositivos. Los dispositivos de origen y de destino serán clasificados por el *IDeviceService.* Una descripción más detallada de este módulo se encuentra en la sección 0.
- **Firewall:** realiza el cumplimiento de reglas ACL (Lista de control de acceso) mediante flujos dinámicos y analizando el comportamiento de los paquetes enviados por los dispositivos. En este caso las reglas ACL son sólo conjuntos de condiciones que permiten o niegan un flujo de tráfico en el switch de ingreso. Una descripción más detallada de este módulo se encuentra en la sección 0.
- **Port Down Reconciliation:** busca y descarta flujos de tráfico dirigidos o un puerto sin servicio.
- **ACL:** realiza el cumplimiento de reglas ACL (Lista de control de acceso) mediante flujos estáticos. A diferencia del módulo de firewall, funciona de manera proactiva y no supervisa los paquetes

enviados por los dispositivos. Logrando de esta forma reducir los tiempos de respuesta.

2.8.4.3 MÓDULOS DE CONTROLADOR

Los módulos de controlador (Project Floodligth, 2017) implementan funciones que son de uso común para una mayoría de aplicaciones de un entorno SDN, tales como: comunicación entre switches y el controlador descubrimiento de la topología de red y dispositivos, flujos de paquetes e interfaz web de administración. A continuación se enumeran los módulos de controlador actualmente implementados:

- **FloodlightProvider:** gestiona las conexiones a los switches y convierte los mensajes OpenFlow en eventos que otros módulos pueden interpretar.
- **DeviceManagerImpl:** rastrea dispositivos que se mueven dentro de una red y define el dispositivo de destino para un nuevo flujo.
- **LinkDiscoveryManager:** es responsable de descubrir y mantener el estado de los enlaces en la red OpenFlow.
- **TopologyService:** mantiene la información de topología de red para el controlador, así como para seleccionar el enrutamiento en la red.
- **RestApiServer:** expone los módulos en la API REST a través de HTTP.
- **ThreadPool:** permite lanzar hilos de ejecución de módulos en momentos específicos o periódicamente.
- **MemoryStorageSource:** es una fuente de almacenamiento de estilo NoSQL en memoria.
- **Packetstreamer:** es un servicio de transmisión de paquetes que puede transmitir de manera selectiva los paquetes de OpenFlow intercambiados entre cualquier switch y el controlador.

- **High Availability Support:** proporciona soporte de alta disponibilidad cuando se ejecutan varias instancias del controlador. Se utiliza para publicar y suscribirse a las actualizaciones de varios controladores.
- **OFSwitchManager:** gestiona todos los switches OpenFlow conectados al controlador.

2.8.4.4 FLOODLIGHT API REST

FloodLight maneja una Interfaz de Programación de Aplicaciones denominada API REST (Project Floodlight, 2016). La cual permite interactuar con el controlador y algunos de sus módulos. Por defecto utiliza el puerto 8080 del controlador. Los módulos accesibles mediante la API REST son: *Firewall*, *StaticFlowEntryPusher*, *VirtualNetworkFiltrer*, *ACL* y *web GUI*. La documentación de como interactuar con cada uno de estos módulos mediante el comando curl se encuentra en (Project Floodlight, 2016).

2.8.4.5 TOPOLOGÍAS DE RED SOPORTADAS

FloodLight soporta dos tipos de topologías (Project Floodligth, 2012). Para ello define Zonas OpenFlow y Zonas no OpenFlow. A la hora de reenviar flujos de paquetes, el comportamiento del controlador varía según a donde tenga que reenviarlos.

Cuando el reenvío se realiza dentro de una zona OpenFlow el módulo de aplicación *Forwarding* calcula la ruta para el envío de paquetes entre dos dispositivos (desde A hasta B).

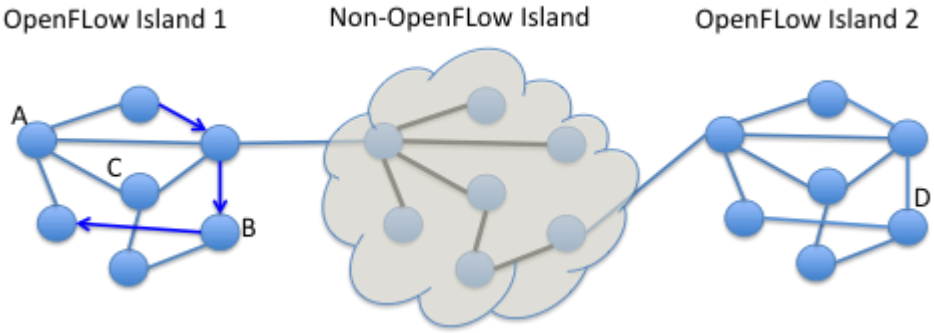

Figura 10 : Topología soportada por Floodligth.
Fuente: (Project Floodligth, 2012).

Por el contrario, cuando se debe enviar paquetes o flujo de datos en una zona mixta, *Forwarding* calcula la ruta en cada zona OpenFlow y espera que los paquetes sean remitidos hacia las zonas no OpenFlow. Es importante destacar que cada zona OpenFlow solo puede tener un enlace con una zona no OpenFlow, con el objetivo de no formar bucles como se observa en la Figura 11.

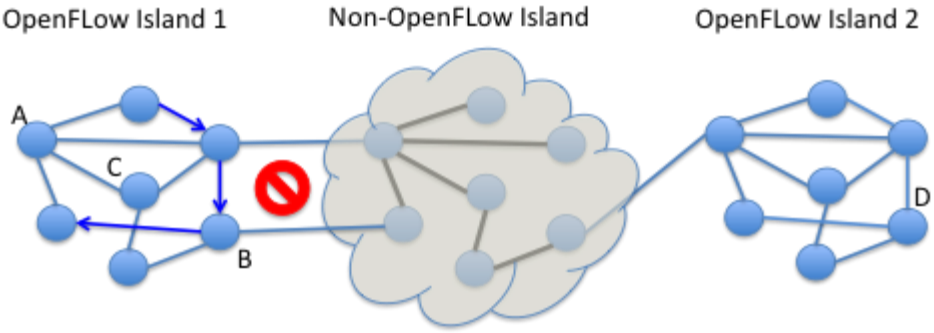

Figura 11 : Topología no soportada por Floodligth.
Fuente: (Project Floodligth, 2012).

CAPÍTULO 3: DESARROLLO Y RESULTADOS

3.1 INTRODUCCIÓN

En este capítulo se presenta el prototipo implementado del sistema para el control de acceso y autenticación en Redes Definidas por Software. El mismo fue desarrollado y probado en un entorno físico, utilizando un dispositivo Mikrotik rb1100ahx2, con soporte OpenFlow 1.0 y tres equipos de escritorio, dos con sistema operativo Debian 8 y uno con sistema operativo Windows 7, los cuales funcionan como servidor de autenticación, controlador SDN (virtualizado) y usuarios de la red. La topología de la red creada se puede observar en la Figura 12.

La aplicación permite controlar el ingreso a la red mediante la utilización de reglas que definen qué equipos tienen acceso a la misma, a través del inicio de sesión previo de los usuarios. Para ello se establecen dos condiciones: PERMITIR o DENEGAR. La primera se aplica cuando un usuario inicia sesión satisfactoriamente en el servidor de autenticación. La segunda en cambio, cuando el inicio de sesión no es satisfactorio y se ha superado el límite de intentos permitidos. Este procedimiento se puede observar con mayor detalle en los diagramas de flujo de la aplicación representados en la Figura 13 y la Figura 14. Los usuarios que no logren iniciar sesión correctamente no podrán acceder ni interactuar con los demás elementos de red. La aplicación utiliza el modelo cliente-servidor y su funcionamiento y arquitectura de red se describen en la Figura 12.

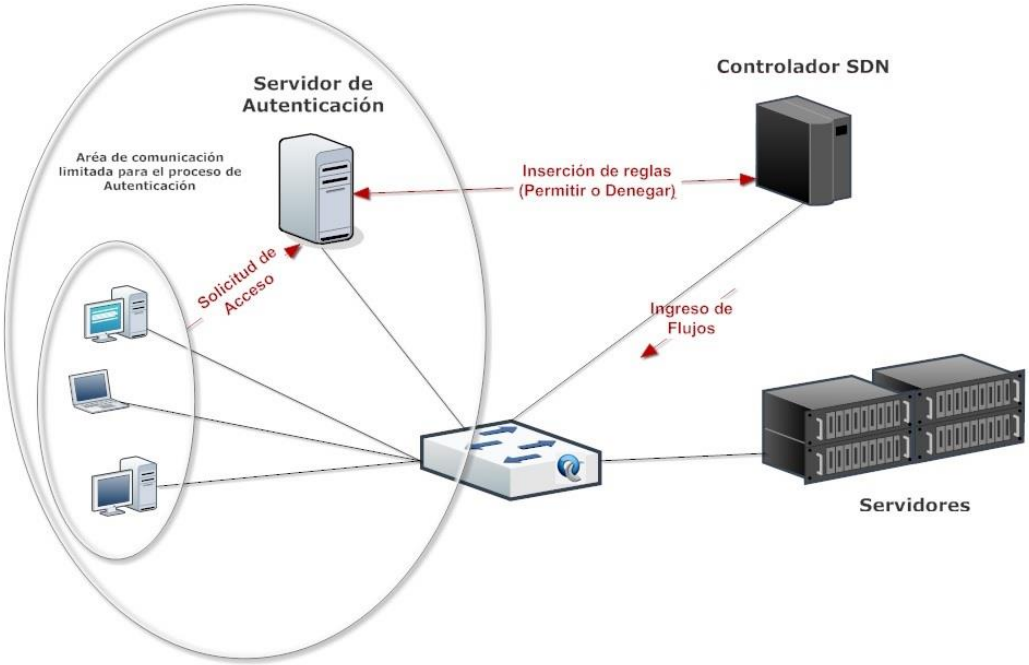

Figura 12: Prototipo desarrollado.

Tal como se puede observar, el modelo está compuesto por un Servidor de Autenticación (SA), encargado de almacenar credenciales y privilegios de los usuarios que poseen autorización para conectarse a la red, un switch Mikrotik con soporte OpenFlow, un controlador SDN virtualizado y usuarios que intentarán conectarse para utilizar la infraestructura. Cuando un cliente desea acceder a la red, se debe conectar como lo hace habitualmente en una red tradicional y recibirá una dirección IP. Este procedimiento se realiza gracias a un servidor DHCP el cual funciona en el SA, los detalles de funcionamiento y configuración se encuentran en la sección 3.6.6. Una vez conectado, el usuario solo tiene acceso al SA y está en condiciones de ejecutar la aplicación Cliente. La aplicación Cliente se encarga de establecer una comunicación cifrada con el servidor, mediante el uso de sockets SSL y el envío de una petición con puerto destino TCP 8443 (los detalles de esta comunicación se exponen en la sección 3.6.5). Gracias a este tipo de comunicación las credenciales del usuario viajan por un medio seguro. Es importante aclarar que este

acceso limitado, el cual le permite al cliente acceder solamente al servidor de autenticación, es posible gracias a reglas estáticas predefinidas en el controlador SDN al momento de iniciar la aplicación. Posteriormente el cliente debe proporcionar su usuario y contraseña de acceso al servidor de autenticación, el cual verifica los datos, y si la comprobación es correcta, inserta en el controlador SDN la regla correspondiente para permitir el ingreso al equipo del usuario a la red, con los privilegios que corresponda. Por ejemplo, accesos a dispositivos y servicios específicos. Si la verificación es incorrecta, insertará una regla para denegarlo después de tres intentos fallidos.

3.2 CONSIDERACIONES DE DISEÑO

El Servidor de autenticación cumple con las siguientes funcionalidades:

- Asignar direcciones IP a los clientes.
- Verifica constantemente la conexión de nuevos clientes, e implementa un mecanismo de conexión segura con los mismos, con el objetivo de que las credenciales viajen cifradas a través de la red.
- Implementa un mecanismo de conexión con el controlador SDN, para poder insertar reglas de firewall previamente generadas.[8]
- Considera la inserción de reglas repetidas, para evitar llenar las tablas de flujo con información redundante.
- Elimina las reglas de los dispositivos que se han autentificado correctamente y eventualmente se van desconectando.
- Elimina las reglas de los dispositivos bloqueados después de un periodo de tiempo.

---

[8] **Generación de reglas:** Para formar una regla, previamente se obtienen los valores que la constituyen (por ejemplo: número de puerto, MAC, id de la regla, etc.) de distintos métodos. Todos esos valores son concatenados en un string que posteriormente se envía al controlador por intermedio de la API REST mediante el comando curl.

- Tiene la capacidad de ensamblar las reglas de tráfico, considerando que los argumentos de las mismas deben estar serializados, concatenados y correctos, antes de ser enviados al controlador.
- Verificar credenciales de usuarios previamente almacenadas.

La aplicación Cliente considera las siguientes tareas:

- Implementa un mecanismo que permite establecer la conexión segura con el servidor de autenticación.
- Solicitar las credenciales del usuario.
- Envía las credenciales del usuario al servidor de autenticación, y recibe la respuesta del mismo, con el objetivo de que el usuario pueda reingresar las credenciales si son incorrectas, o saber si ha accedido correctamente a la red.

3.3 DIAGRAMA DE FLUJO APLICACIÓN CLIENTE

La Figura 13 representa el diagrama de flujo de la aplicación cliente.

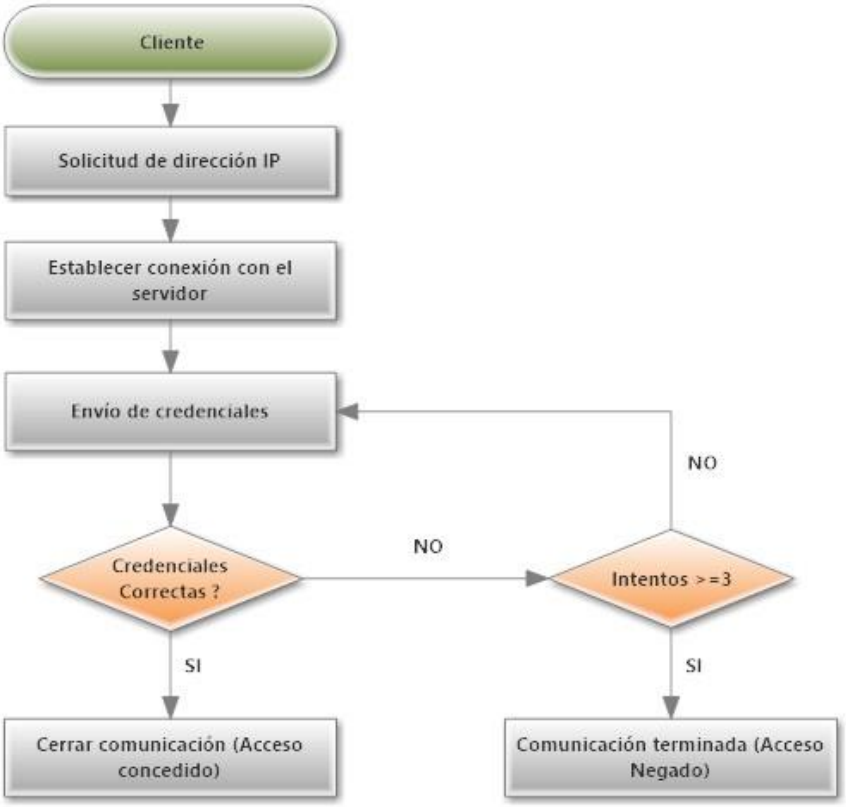

Figura 13: Diagrama de flujo Cliente.

3.4 DIAGRAMA DE FLUJO SERVIDOR DE AUTENTICACIÓN

La Figura 14 representa el diagrama de flujo de la aplicación Servidor.

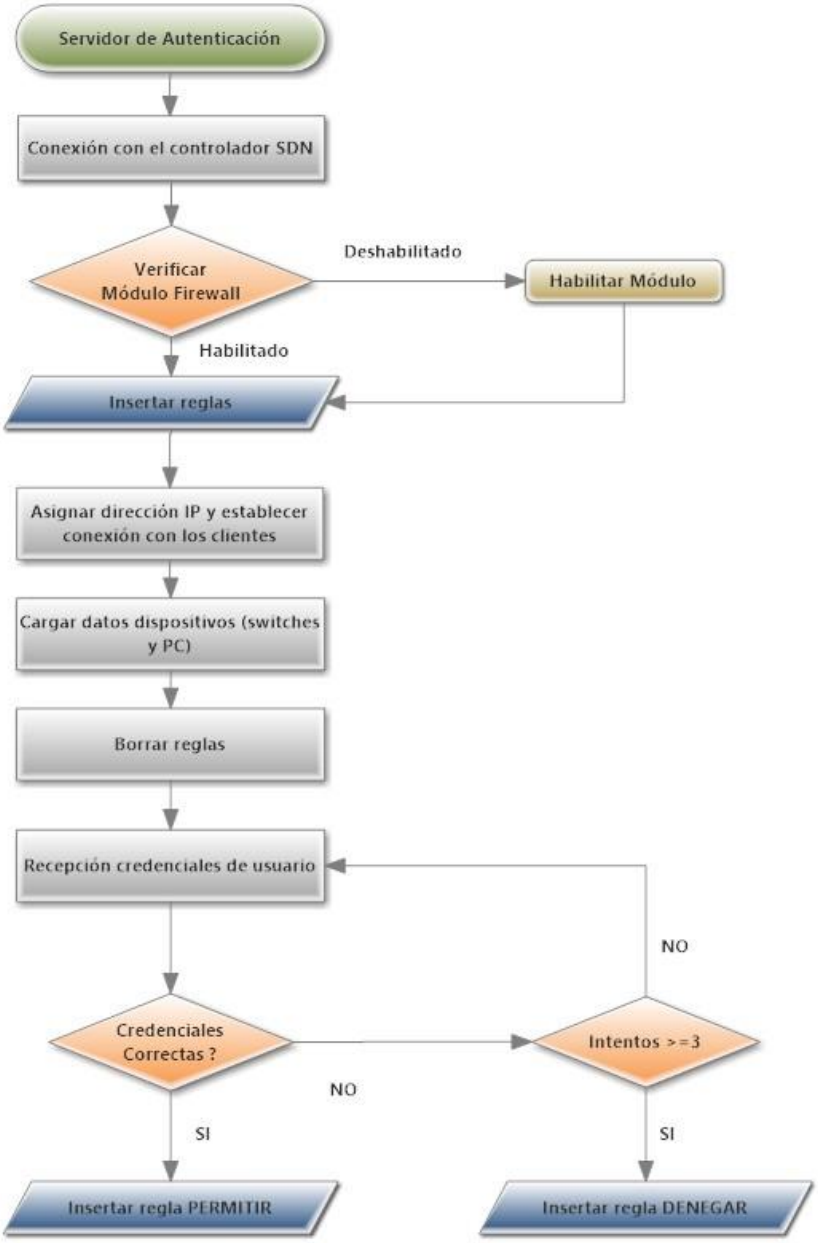

Figura 14: Diagrama de flujo Servidor.

3.5 EVALUACIÓN DE CONTROLADORES

Elegir cuál sería el controlador adecuado para este trabajo ha sido una tarea muy importante y determinante en el resultado final. Se han probado y analizado cuatro controladores de código abierto. Para ello fueron instalados en máquinas virtuales KVM (KVM, 2016) con sistema operativo Debian 8.

En esta sección se presentan las ventajas y desventajas de los controladores evaluados, además se detalla porque se elige Floodligth para realizar el prototipo.

Beacon: provee una gran cantidad de módulos para el desarrollo de una infraestructura SDN. Sin embargo, no se encuentra bien documentada la instalación de estos módulos por lo que ponerlo en funcionamiento resulta una tarea pesada ya que requiere descargar e instalar una gran cantidad de librerías para su funcionamiento.

Trema: es un framework completo y dispone de una serie de recursos que permiten el desarrollo de un controlador SDN. A pesar de esto no se adapta a las necesidades del prototipo a desarrollar, ya que adecuarlo para que ofreciera las funcionalidades necesarias requeriría gran cantidad de tiempo y no es el objetivo de este trabajo.

OpenDayligth: es una alternativa muy completa y fácil de instalar. Pero la curva de aprendizaje para interactuar mediante la API REST es abrupta. Además de que la documentación no es del todo amigable y se necesita invertir demasiado tiempo para poder utilizarlo correctamente.

FloodLight: es simple de instalar y configurar. La interfaz web que ofrece es intuitiva y expone las características de los dispositivos de la red. Adicionalmente se encuentra muy bien documentado, lo que facilita la interacción con el controlador y el desarrollo de aplicaciones. El punto principal por el cual se seleccionó este controlador para realizar el prototipo propuesto, es la diversidad de módulos que maneja. En especial los módulos Firewall y Forwarding. Además de ser totalmente compatible con el switch con soporte OpenFlow utilizado.

La Tabla 1 compara los principales aspectos de los controladores probados:

Tabla 1. Tabla comparativa controladores SDN.

Controlador	Trema	Opendayligth	Beacon	Floodligth
Dificultad de instalación	Fácil	Fácil	Regular	Fácil
Código Abierto	Si	Si	Si	Si
Documentación	Regular	Media	Media	Excelente
REST API	SI	SI	SI	SI
Soporte	Malo	Bueno	Regular	Bueno
Manejo de interfaz web	SI	SI	SI	SI
Soporte OpenFlow	OF 1.3	OF 1.3	OF 1.0	OF 1.5
Lenguaje de programación	Ruby y C	Java	Java	Java

3.5.1 MININET

Mininet (Mininet, 2016) es un simulador de red el cual es capaz de crear redes de hosts, switches, controladores SDN y links virtuales. Para ello ejecuta una colección de equipos finales dentro de un solo núcleo de Linux y utiliza virtualización para hacer que un solo sistema parezca una red completa. Logrando crear, personalizar e interactuar con un prototipo SDN.

Además, es una solución poco costosa, ya que está desarrollado bajo licencia Open Source BSD y se puede obtener de manera gratuita. Posee gran facilidad de instalación; es posible descargar una máquina virtual con el software instalado o instalarlo manualmente en una gran cantidad de distribuciones Linux.

Debido a estas ventajas mencionadas en el párrafo anterior, en este trabajo se utilizó Mininet para realizar la evaluación de los controladores SDN. Para ello se instaló manualmente sobre una máquina virtual KVM

con sistema operativo Debian 8, siguiendo la documentación presentada en el sitio de Mininet. Posteriormente se instaló cada uno de los controladores mencionados en la sección 3.5, en máquinas virtuales separadas.

La Figura 15 muestra la topología creada donde se puede apreciar los controladores probados, el swicht OpenVSwitch y los host, estos dos últimos emulados por Mininet.

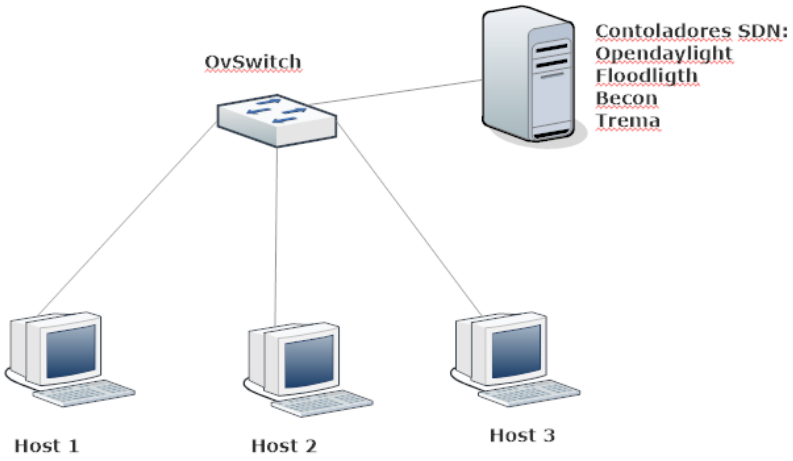

Figura 15 : Topología de red Mininet.

El comando utilizado para crear la topología es:

```
# mn --topo single,3 --controller remote,ip=10.0.0.20,port=6633
```

Dónde:

- **mn:** ingresa al entorno Mininet y crea la topología según los parámetros indicados.
- **--topo single,3:** Crea una topología simple con tres host.
- **--controller remote:** indica que se va a utilizar un controlador remoto del cual se provee la dirección IP y el puerto a utilizar.

Cabe aclarar que si no se indica la utilización de un controlador remoto, Mininet creará la topología usando en controlador POX.

3.6 COMPONENTES DEL SISTEMA

3.6.1 CONTROLADOR SDN

Tal como se ha mencionado en la sección 3.5, el controlador SDN seleccionado es Floodligth en la versión 0.91 (Floodlight v0.91, 2015), debido a la compatibilidad con la versión 1.0 del protocolo OpenFlow, utilizada por el switch Mikrotik. El controlador funciona en una máquina virtual KVM sobre un sistema operativo Debian 8.6, la cual utiliza como host físico al mismo equipo del SA.

3.6.2 MÓDULOS DE FLOODLIGTH UTILIZADOS

Como se detalló en la sección 2.8.4, Floodligth dispone de una gran cantidad de módulos que ofrecen distintas funcionalidades. Para este desarrollo se emplearon los módulos Forwarding y Firewall de la API REST de FloodLight ya que proporcionan las características necesarias para el manejo de paquetes y el reenvío de los mismos. Además dispone de las acciones para permitir o denegar flujos de paquetes.

3.6.3 FORWARDING

Forwarding (Forwarding (Dev), 2015) es un módulo del controlador que se encarga de reenviar paquetes entre los dispositivos. Tiene la capacidad de establecer una ruta para cada flujo enviado por los dispositivos de la red SDN. Por defecto se encuentra habilitado y carga automáticamente al inicio del controlador.

El reenvío se realiza analizando los campos del encabezado: ID de VLAN, direcciones MAC de origen/destino, direcciones IP de origen/destino y

puertos de transporte de origen/destino. Adicionalmente este módulo depende de los siguientes servicios para funcionar:

- *IDeviceService.*
- *IFloodlightProviderService.*
- *IRestApiService.*
- *IRoutingService.*
- *ITopologyService.*
- *IDebugCounterService.*
- *IOFSwitchService.*

3.6.4 FIREWALL

Firewall (Amer & Izard, 2015) es el módulo encargado de hacer cumplir las reglas ACL (Access Control List) (Ryan, 2017) en switches OpenFlow mediante la inserción de flujos y el seguimiento del comportamiento de los paquetes. Las reglas ACL son definidas como un conjunto de condiciones que pueden permitir o denegar la entrada de un flujo de tráfico al switch.

Firewall funciona de una manera reactiva y las reglas se ordenan según la prioridad en el momento de su creación (a través de la API REST). Para cada paquete que entra del tipo *Packet-In* se verifica una coincidencia en la tabla de flujo, y posteriormente se compara su valor prioridad para elegir la más alta. Si se encuentra una coincidencia, la acción de la regla (permitir o denegar) se almacena en un objeto, luego el flujo se seguirá transmitiendo hasta llegar al módulo Forwarding para su respectivo procesamiento.

En este trabajo, se utiliza Forwarding para enviar un flujo de entrada de reenvío si la acción es permitir, o una entrada de flujo de descarte si la acción es denegar.

Firewall maneja los métodos de HTTP: GET, POST y DELETE; la explicación detallada de estos y su estructura se encuentra disponible en (Firewall REST API, 2015).

3.6.5 COMUNICACIÓN ENTRE SERVIDOR DE AUTENTICACIÓN Y CLIENTES

Debido a que cada cliente que desee acceder a la red deberá proporcionar sus credenciales (usuario y contraseña) al SA y es necesario que la información viaje de manera segura por la red, se utilizó el protocolo SSL (Secure Socket Layer) (IETF, 2001) para la comunicación entre los clientes y el servidor. Puesto que SSL utiliza certificados digitales en el proceso de autenticación, fue necesario generar tales certificados para el servidor. Se utilizó la herramienta proporcionada por Java denominada keytool (Oracle, 2016), la cual, junto con los parámetros pertinentes, permite la creación y manipulación del almacén y los certificados.

Antes de ser creados se tuvieron en cuenta algunas consideraciones. Java distingue entre los keystores y los truststores. Keystore, (almacén de certificados), es una base de datos de pares llaves y de certificados utilizados para la autentificación SSL. En un Truststore se almacenan los certificados de confianza o certificados de las Autoridades de Certificación (CA), se utilizan para verificar las identidades de otros clientes y servidores (Oracle, 2010).

Cuando un cliente o un servidor inician una sesión SSL, extrae sus certificados y claves de su almacén de certificados (keystore). En cambio, cuando verifica las identidades de otros clientes o servidores, extraerá los certificados de su almacén de certificados de confianza (truststores). En este trabajo se optó por que la autenticación fuera unidireccional, es decir, los clientes poseen el certificado del SA en su trutstore por lo tanto

“confían” en el mismo. Para ello se generó un keystore en el servidor de autenticación con su respectivo certificado. Posteriormente se exportó dicho certificado al trutstore de los clientes. En ANEXO I se detallan los parámetros utilizados con la herramienta keytool.

3.6.6 ASIGNACIÓN DE DIRECCIONES IP

Dado que cada cliente que se conecta solicita una dirección IP, fue necesario configurar un servidor DHCP (Dynamic Host Configuration Protocol) en el servidor de autenticación. Se utilizó el servidor ISC-DHCP-SERVER (Internet Systems Consortium, 2017), el cual es una implementación libre del dicho protocolo. En las líneas siguientes se describe la instalación y configuración del mismo.

Gracias a que los paquetes de instalación se encuentran en los repositorios de Debian, la instalación del servidor se resume al siguiente comando:

```
#aptitude install isc-dhcp-server
```

Una vez instalado fue necesario modificar dos archivos de configuración. El primero ubicado */etc/dhcp/dhcpd.conf* el cual quedó configurado de la siguiente manera:

```
subnet 192.168.10.0 netmask 255.255.255.0{
range 192.168.10.10 192.168.10.200; #Rango de la 10 a la 200 inclusive
option broadcast-address 192.168.10.255; # Dirección de difusión
option routers 192.168.10.1; # Puerta de enlace
option domain-name-servers 192.168.10.1;
default-lease-time 3600; # Tiempo en segundos de la sesión
max-lease-time 7200; # Máximo tiempo en segundos que durará la sesión
}
```

La configuración pertenece a la red 192.168.10.0/24, donde se asignarán direcciones IP desde la dirección 192.168.10.10 a la dirección 192.168.10.254. Proporcionando la dirección 192.168.10.1 como puerta de enlace y la dirección 192.168.10.255 como broadcast. El período de

tiempo en segundos de concesión de las direcciones es 3600 seg y el máximo de 7200 seg.

Posteriormente se modificó el archivo ubicado en */etc/default/isc-dhcp-server* en el cual solo se agregó la línea `INTERFACES="eth1"`. Para indicarle al servidor DHCP que solo atienda peticiones en la interfaz eth1.

3.6.7 SWITCH UTILIZADO Y CONFIGURACIÓN DEL MISMO

Como se mencionó al principio de este capítulo, para armar la infraestructura del prototipo realizado se utilizó un dispositivo Mikrotik RouterBoard 1100 X2 AH (Mikrotik, 2015) con las especificaciones técnicas descritas en la Tabla 2.

Tabla 2. Especificaciones RB1100AHx2.

RB1100AHx2	
Puertos Ethernet 10/100/1000	**Cantidad: 13**
CPU	**Modelo: P202ASSE2KFB**
Núcleos CPU	**Cantidad: 2**
Nivel de Licencia	**6**
Sistema Operativo	**RouterOS**
Memoria RAM	**2 GB**
Tamaña de disco	**128 MB**

En la configuración de fábrica, este dispositivo no incluye el paquete de soporte para OpenFlow, por lo que fue necesario instalarlo obteniéndolo de la página oficial de Mirkotik. El proceso de instalación se realizó siguiendo la documentación oficial del equipo (MikroTik, 2002). Además fue necesario actualizar sistema operativo RouterOS de la versión 6.30 a la versión 6.39.2, el proceso se puede consultar en (Mikrotik, 2016).

La Figura 16 muestra la lista de paquetes instalados en el dispositivo.

```
[admin@MikroTik] > system package print
Flags: X - disabled
 #   NAME                      VERSION
 0   routeros-powerpc          6.39.1
 1   system                    6.39.1
 2 X ipv6                      6.39.1
 3   wireless                  6.39.1
 4   hotspot                   6.39.1
 5   dhcp                      6.39.1
 6   mpls                      6.39.1
 7   routing                   6.39.1
 8   ppp                       6.39.1
 9   security                  6.39.1
10   advanced-tools            6.39.1
11   openflow                  6.39.1
```

Figura 16 : Lista de paquetes instalados.

Con el objetivo de poder conectarnos al dispositivo para administrarlo y que posteriormente se pueda comunicar con el controlador SDN, se debe configurar una dirección IP en uno de los puertos. En este caso se eligió el puerto *"ether1"* y se configuró la dirección IP 10.0.0.10/24 mediante el siguiente comando:

```
[admin@MikroTik] > ip address add address=10.0.0.1/24 interface=ether1
```

Una vez configurada la interfaz de administración y comunicación con el controlador SDN se procedió a configurar el switch OpenFlow asignándole un nombre y la dirección IP del controlador con el siguiente comando:

```
[admin@MikroTik] > openflow add name=ofswitch controllers=10.0.0.20
```

Posteriormente se asignó cada uno de los puertos a usar con OpenFlow de la siguiente manera:

```
[admin@MikroTik] > openflow port add switch=ofswitch interface=ether2
[admin@MikroTik] > openflow port add switch=ofswitch interface=ether3
[admin@MikroTik] > openflow port add switch=ofswitch interface=ether4
[admin@MikroTik] > openflow port add switch=ofswitch interface=ether5
[admin@MikroTik] > openflow port add switch=ofswitch interface=ether6
[admin@MikroTik] > openflow port add switch=ofswitch interface=ether7
[admin@MikroTik] > openflow port add switch=ofswitch interface=ether8
[admin@MikroTik] > openflow port add switch=ofswitch interface=ether9
[admin@MikroTik] > openflow port add switch=ofswitch interface=ether10
[admin@MikroTik] > openflow port add switch=ofswitch interface=ether11
[admin@MikroTik] > openflow port add switch=ofswitch interface=ether12
[admin@MikroTik] > openflow port add switch=ofswitch interface=ether13
```

La Figura 17 describe gráficamente como quedaron asignados los puertos del dispositivo.

Figura 17 : Asignación de puertos.

3.7 LENGUAJE Y CÓDIGO UTILIZADOS

La aplicación fue desarrollada en lenguaje Java (Oracle, 2018), utilizando la plataforma Eclipse (Eclipse Foundatio, 2018). Además se reutilizaron algunas clases del código Avior (Marist College, 2013) escrito por Jason Parraga del Marits College y licenciado bajo “The MIT License” (MIT, 1988), el cual se puede obtener en (Parraga, 2013). Avior proporciona una interfaz gráfica para la comunicación con el controlador Floodligth. A continuación se describen las clases reutilizadas y sus funcionalidades en la aplicación desarrollada.

3.7.1 CLASES DE AVIOR REUTILIZADAS

Dado que ciertas clases de Avior se adaptan a la necesidades de la aplicación desarrollada, se decidido reutilizarlas y realizarles las modificaciones necesarias para que funcionen adecuadamente. Las clases modificadas son: FloodLightProvider.java, Rule.java y FirewallPusher.java, ubicadas respectivamente en los directorios:

- */src/model/tools/firewall*
- */src/controller/FloodlightProvider*
- */src/controller/tools/firewall/push*

Además, para el descubrimiento de los equipos de conectividad y de las PCs de los clientes se han utilizado las clases Switches.java y DevicesSummary.java sin modificaciones, ubicadas en el directorio */src/model/overview.*

Adicionalmente, para la comunicación con el controlador mediante la API REST se utilizó las clases JSON (FirewallJSON, RuleJSON, DevicesJSON y SwitchJSON), también sin modificaciones.

A continuación se detalla una descripción de las funciones que realizan las clases mencionadas y los cambios realizados.

- **FirewallPusher.java**: define las funciones *push()* y *remove()* para agregar y eliminar los flujos en el controlador.
- **FloodLightProvider.java**: permite obtener la lista de las reglas definidas, los switches conectados e información del controlador. Esta clase fue adaptada para que adicionalmente se obtenga la lista de los dispositivos de los usuarios conectados utilizando el método *getDeviceSummaries()* de la clase DevisesJSON.java.

- **Rule.java:** define cada uno de los parámetros de las reglas de tráfico a insertar como dirección MAC, direcciones IP de origen y destino, protocolos, puertos y acciones. Además realiza la concatenación de los mismos. En esta clase se modificó el método *rule()*, con el objetivo de que incluya el DPID[9] del switch como argumento inicial de cada regla.
- **Switches.java**: define las características de los switches OpenFlow conectados. Por ejemplo DPID, puertos y especificaciones de hardware.
- **DevicesSummary.java:** define las características de los equipos de los usuarios conectados como la dirección IP, dirección MAC, puerto del switch al cual están conectados y tiempo de conexión.

3.8 DIAGRAMA DE CLASES Y DIAGRAMA DE SECUENCIA

3.8.1 DIAGRAMA DE CLASES

En la Figura 18 y la Figura 19, se muestra el diagrama de clases de la aplicación desarrollada, en el cual se describe la estructura del sistema exponiendo clases y relaciones.

[9] **DPID: (Data path ID):** Es el identificador de cada switch OpenFlow. Como su nombre lo indica, identifica inequívocamente a cada switch dentro de la red. Está formado por 64 bits determinados de la siguiente manera: los primeros 16 bits son configurables por el operador de la red y los 48 restantes corresponden a la dirección MAC del dispositivo.

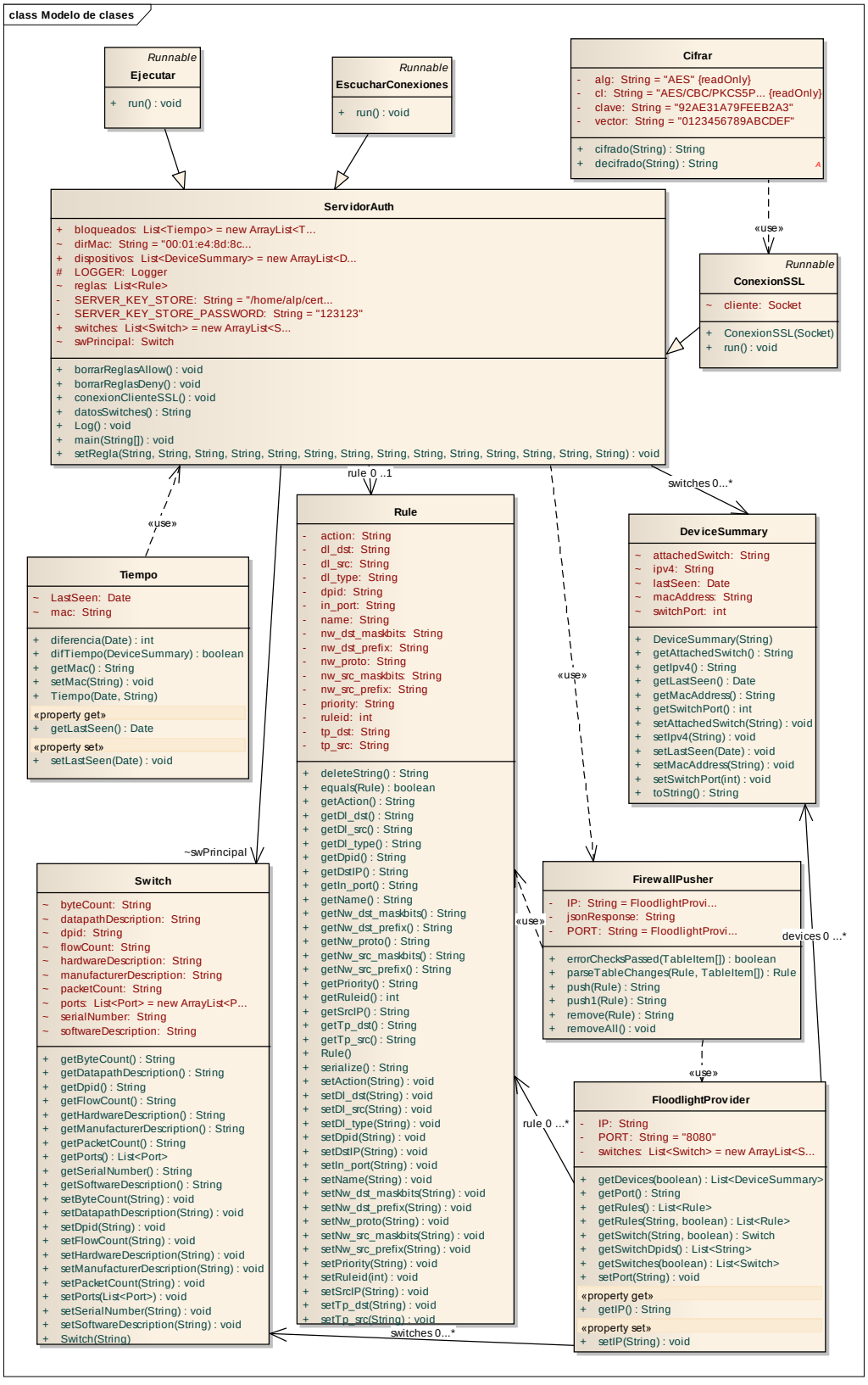

Figura 18 : Diagrama de clases Servidor

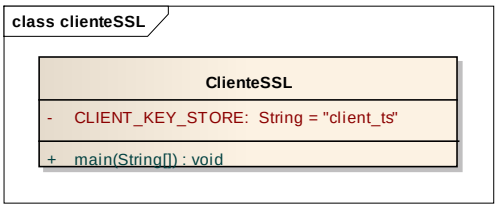

Figura 19 : Diagrama de clases Cliente.

3.8.2 DIAGRAMA DE SECUENCIA

La Figura 20 representa el diagrama de secuencia correspondiente a la interacción entre las aplicaciones ServidorAuth, ClienteSSL y el controlador Floodligth a medida que transcurre el tiempo. En él se puede observar el intercambio de mensajes llevado a cabo entre los componentes de sistema desde el inicio de las aplicaciones. Cada una de estas interacciones es realizada por los métodos desarrollados.

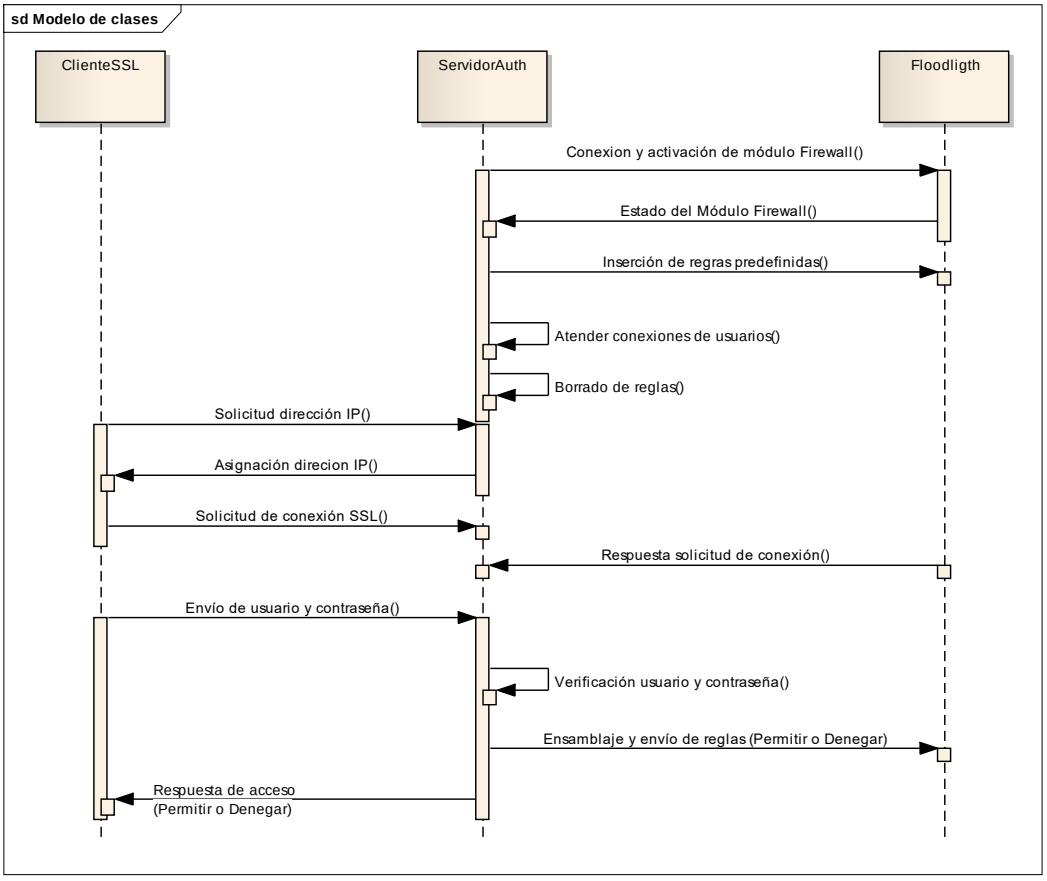

Figura 20 : Diagrama de Secuencia.

3.9 DETALLE DE CLASES Y MÉTODOS DESARROLLADOS

Como se observa en la Figura 18, la aplicación desarrollada está dividida 11 clases. En las líneas posteriores se detalla el funcionamiento de cada una de ellas con sus respectivos métodos.

3.9.1 CLASE SERVIDORAUTH

La clase principal de la aplicación se denomina ServidorAuth.java está ubicada en */src/servidor/* y se constituye de 5 métodos principales:

- *main().*
- *borrarReglasAllow().*
- *borrarReglasDeny().*
- *conexiónClienteSSL().*
- *datosSwitches().*
- *setRegla().*
- *Log().*

3.9.2 MÉTODO MAIN()

El Código 1 corresponde al método *main()*, el cual es el encargado de iniciar la aplicación. Al ponerlo en ejecución la primera tarea es comprobar que se haya iniciado con el parámetro correspondiente a la dirección IP del controlador. Con el objetivo de almacenarlo en una variable de la clase Floodligthprovider (línea 3) para poder realizar la comunicación con el mismo. En caso de que la aplicación se hubiera iniciado sin el parámetro correspondiente se mostrará un error y terminara la ejecución de la misma (línea 29 a 31). La Figura 21 muestra el error mencionado.

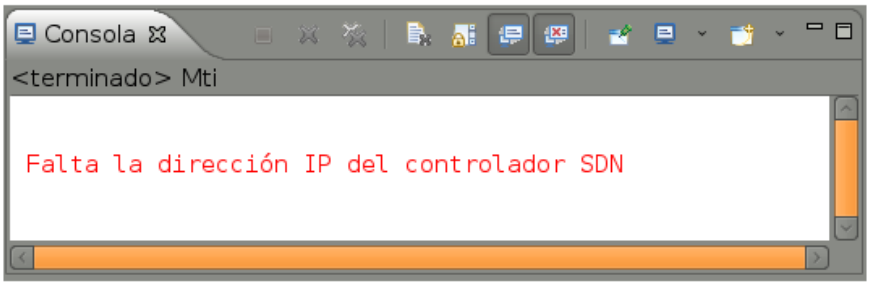

Figura 21: Error falta de argumentos al iniciar aplicación Servidor.

El siguiente paso es verificar que el módulo Firewall del controlador se encuentra activo (línea 6), ya que por defecto inicia desactivado y permite el reenvió de todo tipo de tráfico. En caso de que este desactivado se activará (línea 7). Este procedimiento se ha sido encerrado a través de un bloque try, ya que en caso de producirse algún tipo de excepción al activar el módulo debe ser capturada. Una vez activado el módulo Firewall se procede a consultar el DPID del switch (línea 12) mediante el método *datosSwitches()*, con el objetivo de utilizarlo posteriormente al insertar reglas. En las líneas sucesivas (línea 14 a 20) se insertan 5 reglas de Firewall mediante el método *setRegla()*. La finalidad de las mismas es permitir la comunicación inicial de los clientes con el servidor de autenticación. La primera regla insertada habilita el tráfico ARP (línea 14), para que los clientes puedan llenar sus tablas y lograr la comunicación con el servidor de autenticación. Posteriormente se permite el tráfico DHCP (línea 16 y 17), para ello se habilitan los puertos 67 y 68 del protocolo UDP. Por último se habilita el puerto TCP 8443 (línea 19 y 20) para permitir la comunicación mediante los sockets SSL.

```
1.  public static void main(String[] args) {
2.      if(args.length == 1){
3.              FloodlightProvider.setIP(args[0]);
4.              String resp = "";
5.                   try {
6.                           if (!FirewallJSON.isEnabled()) {
7.                             resp = FirewallJSON.enable(true);
8.                           }
9.                   } catch (JSONException | IOException e1) {
10.                                 e1.printStackTrace();
11.                          }
12.     String pid =datosSwitches();
13.     //Tráfico ARP
14.     setRegla("","","","ARP", "","","","","","","","","", pid);
15.     //Tráfico DHCP servidor de autenticacion
16.     setRegla("", "", "", "", "", "", "", "", "UDP", "67", "", "", "", pid);
17.     setRegla("", "", "", "", "", "", "", "", "UDP", "", "68", "", "", pid);
18.     //Tráfico comunicación Socket
19.     setRegla("", "", "", "", "", "", "", "", "TCP", "8443", "", "", "", pid );
20.     setRegla("", "", "", "", "", "", "", "", "TCP", "", "8443", "", "", pid );
21.
22.     Log();
23.
24.      Thread h = new Thread(new Ejecutar());
25.      h.start();
26.      Thread conexiones = new Thread(new EscucharConexiones());
27.       conexiones.start();
28.
29.     }else{
30.             System.out.println("Falta la dirección IP del controlador SDN");
31.     }
32. }
```

Código 1: Método main().

Una vez insertadas las reglas para permitir el tráfico inicial de los clientes con el servidor de autenticación, se realiza una llamada al método *Log()* (línea 22) el cual es el encargado de registrar en un archivo la autenticación de los clientes.

Posteriormente se lanzan dos hilos de ejecución (línea 24 a 27). El primer hilo se lanza mediante una instancia de la clase Ejecutar, y tiene como objetivo estar en constante verificación de los nuevos dispositivos que se conecten a la red. El segundo hilo se lanza mediante una instancia de la clase EscucharConexiones, la cual ejecuta el método *conexiónClienteSSL()* heredado de la clase ServidorAuth, con el objetivo de atender conexiones de los clientes mediante el socket SSL y crear un hilo individual para cada conexión nueva.

En las secciones 3.9.7 y 3.9.12 respectivamente se explica el funcionamiento de estas dos clases y se expone el código de las mismas.

3.9.3 MÉTODO DATOSSWITCHES()

El método *datosSwitches()* permite detectar los switches conectados y almacenar la información de los mismos. En el Código 2 se puede observar su funcionamiento.

```
1.  public static String datosSwitches() {
2.                  switches = FloodlightProvider.getSwitches(true);
3.                  String pid=null;
4.                  for(Switch sw : switches)
5.                                  {
6.                                  if(sw.getDpid().equals(dirMac))
7.                                  pid=sw.getDpid();
8.                                  }
9.                  return pid;
10. }
```

Código 2: Método datosSwitches().

La primera tarea del método es obtener un arreglo de los switches detectados por el controlador (línea 2) durante el descubrimiento de la topología mediante el envío de paquetes LLDP. Esta información es obtenida gracias al método *getSwitches()* de la clase Floodligthprovider, que a su vez utiliza a la clase SwitchJSON. La información obtenida corresponde a los DPID de los switches. Y es imprescindibles al momento de insertar reglas en el controlador, debido a que es necesario indicar mediante el DPID del switch a que dispositivo se aplicará cada regla insertada. Posteriormente se recorre el arreglo obtenido anteriormente (línea 4 a 8) y se almacena el DPID del switch en la variable pid.

3.9.4 MÉTODO BORRARREGLASALLOW()

```
1.  public static void borrarReglasAllow()
2.     {
3.     dispositivos = FloodlightProvider.getDevices(true);
4.     for(Switch lista : switches)
5.        {
6.        reglas = FloodlightProvider.getRules(lista.getDpid(), true);
7.        for(DeviceSummary aux : dispositivos){
8.           if(aux.getSwitchPort()==0 && !reglas.isEmpty()){
9.              try{
10.                for (Rule rl : reglas){
11.                   if(!rl.getAction().equals("DENY")){
12.                     if(!rl.getDl_src().equals("")){
13.                       if(!aux.getMacAddress().equals("")){
14.                         if(rl.getDl_src().toLowerCase().
                                                    equals(aux.getMacAddress())){
15.                            String resp = FirewallPusher.remove(rl);
16.                            System.out.println(resp);
17.                            return;
18.          }}}}}
19.             }catch(IOException | JSONException e){
20.               e.printStackTrace();
21.             }catch(NullPointerException e){
22.                 System.out.println(e.getMessage());
23.  }}}}
24. }
```

Código 3: Método borrarReglasAllow().

En el Código 3 se define el método *borrarReglasAllow()*. El objetivo del mismo es realizar la eliminación de las reglas de los dispositivos que accedieron satisfactoriamente a la red y se han desconectando. Es decir, el equipo de un usuario que ha ingresado las credenciales correctamente, ha accedido a la red y después de un período de tiempo se ha desconectado. Por lo que es necesario borrar la regla que se había insertado para permitirle el acceso. De esta forma cuando se conecte nuevamente deberá volver a ingresar sus credenciales. Para ello se han declarado los siguientes ArrayList:

-dispositivos: (línea 3) almacena la información de todos los dispositivos detectados, como dirección IP, puerto al que está conectado, dirección MAC, fecha de la última conexión, entre otras. Esta información es obtenida mediante el método *getDevices()* de la clase FloodlightProvider, que a su vez utiliza a la clase DeviceJSON.

-switches: (línea 4) almacena la información de los equipos de conectividad, la cual es obtenida mediante el método *getSwitches()* de la clase Floodligthprovider. Esta lista es actualizada cada vez que se ejecuta el método *datosSwitches()*.

-reglas: (línea 6) contiene la lista de reglas de firewall insertadas el controlador, esta información es obtenida mediante el método *getRules()* de FloodlightProvider, que a su vez utiliza a la clase FirewallJSON.

Para determinar si un cliente se ha desconectado, se recorren las listas y se realizan diferentes verificaciones: primero se verifica que el puerto físico del switch al cual estaba conectado este libre y que la lista reglas no este vacía (línea 8). Posteriormente se verifica que las reglas posean el campo *"action"* distinto de "DENY" (línea 11) ya que estas últimas serán borradas con el método *borrarReglasDeny()*. A continuación se comprueba que los campos *"dl_src"* y *"macAddress"* no estén vacíos (línea 12 y 13). Por último se verifica que la dirección MAC del dispositivo sea igual a la dirección MAC de origen de la regla (línea 14), lo que quiere decir, que es la regla que permite el acceso a la red al dispositivo y debe ser eliminada. Para ello, se utiliza el método *remove()* de la clase FirewallPusher (línea 15).

3.9.5 MÉTODO BORRARREGLASDENY()

El Código 4 corresponde al método *borrarReglasDeny()*, cuya funcionalidad es borrar las reglas de los dispositivos que no lograron acceder correctamente a la red después de haber superado los intentos permitidos.

```
1.  public static void borrarReglasDeny()
2.     {
3.     for(Switch lista : switches){
4.  reglas = FloodlightProvider.getRules(lista.getDpid(), true);
5.  for(DeviceSummary aux : dispositivos){
6.        if(Tiempo.difTiempo(aux) && !reglas.isEmpty()){
7.           try{
8.              for (Rule rl : reglas){
9.                 if(!rl.getDl_src().equals("")){
10.                   if(!aux.getMacAddress().equals("")){
11.                      if(rl.getDl_src().toLowerCase().
                                                  equals(aux.getMacAddress ())){
12.                         String resp = FirewallPusher.remove(rl);
13.                         System.out.println(resp);
14.                         return;
15.             }}}}
16.          }catch(IOException | JSONException e){
17.             e.printStackTrace();
18.          }catch(NullPointerException e){
19.                System.out.println(e.getMessage());
20.    }}}}
21. }
```

Código 4: Método borrarReglasDeny.

A diferencia del método *borrarReglasAllow()*, en este caso no se borran las reglas cuando el cliente se desconecta del puerto físico del switch, sino que cada vez que un cliente es bloqueado deberá esperar 10 minutos para que sea borrada la regla que le prohíbe el ingreso y pueda volver intentar conectarse.

El funcionamiento de método es muy similar al método *borrarReglasAllow()* anteriormente explicado, la diferencia está en que no se comprueba que el puerto este libre, sino que se comprueba que el tiempo de bloqueo de un cliente haya superado los 10 minutos. Para ello utiliza la clase Tiempo (línea 6), la cual maneja la lista de objetos denominada bloqueos, en la que se almacenan los clientes bloqueados con los datos de la última conexión. Gracias a estos datos se puede verificar cuanto es el tiempo transcurrido desde que fue bloqueado cada dispositivo. Cuando el tiempo es mayor a 10 minutos se procede al borrado de la regla (línea 12).

3.9.6 MÉTODO SETREGLA()

El Código 5 corresponde al método *setRegla()*, cuya funcionalidad es almacenar cada uno de los argumentos de la clase Rule de avior, para posteriormente concatenarlos e insertar una regla de firewall.

```
1.  public static void setRegla(String dpid, String in_port, String dl_src, String
     dl_dst, String dl_type, String nw_src_prefix, String nw_src_maskbits, String
    nw_dst_prefix, String nw_dst_maskbits, String nw_proto, String tp_src, String
    tp_dst, String action, String priority)
2.  {
3.     Rule rule = new Rule();
4.     rule.setDpid(dpid);
5.     rule.setIn_port(in_port);
6.     rule.setDl_src(dl_src);
7.     rule.setDl_dst(dl_dst);
8.     rule.setDl_type(dl_type);
9.     if(!nw_src_prefix.equals(""))
10.       rule.setNw_src_prefix(nw_src_prefix);
11.    if(!nw_src_maskbits.equals(""))
12.        rule.setNw_src_maskbits(nw_src_maskbits);
13.    if(!nw_dst_prefix.equals(""))
14.        rule.setNw_dst_prefix(nw_dst_prefix);
15.    if(!nw_dst_maskbits.equals(""))
16.        rule.setNw_dst_maskbits(nw_dst_maskbits);
17.    rule.setNw_proto(nw_proto);
18.    rule.setTp_src(tp_src);
19.    rule.setTp_dst(tp_dst);
20.    rule.setAction(action);
21.    rule.setPriority(priority);
22.
23.    String respuesta;
24.    try{
25.          respuesta = FirewallPusher.push(rule);
26. if(respuesta.equals("Rule added")) {
27.          System.out.println("\n Regla de Firewall insertada");
28.          }else{
29.          System.out.println("Un problema ocurrió al insertar la regla:
                                                                  " + respuesta);
30.          }
31.        }catch(IOException | JSONException e1){
32.        e1.printStackTrace();
33.        }
34. }
```

Código 5: Método setRegla().

Se puede observar en la definición del método, cada una de los parámetros que recibe, los cuales corresponden a cada valor que puede incluir una regla de firewall. La Tabla 3 detalla cada uno de ellos.

Tabla 3. Parámetros método serRegla().

Valor	Descripción
dpid	Define el dpid del swich OpenFlow al cual se le aplicará la regla.
in_port	Define el puerto físico del switch por donde ingresa el flujo de datos.
dl_src	Define la dirección MAC de origen.
dl_dst	Define la dirección MAC de destino.
dl_type	Define el tipo de protocolo Ethernet.
nw_src_prefix	Define la dirección IP origen.
w_src_maskbits	Define la máscara correspondiente a la dirección IP origen.
nw_dst_prefix	Define la dirección IP de destino.
w_dst_maskbits	Define la máscara correspondiente a la dirección IP de destino.
nw_proto	Define el protocolo de capa Red.
tp_src	Define el puerto TCP/UDP origen.
tp_dst	Define el puerto TCP/UDP destino.
action	Define una de dos acciones ALLOW y DENY.
priority	Define la prioridad del flujo.

La primera tarea del método es instanciar un objeto de la clase Rule (línea 3), en el que posteriormente se almacenan en sus atributos, cada uno de los parámetros enviados al método (línea 4 a 21).

Posteriormente se realiza la inserción de la regla en el controlador mediante el método *push()* de la clase FirewallPusher (línea 25), que a su vez utiliza el método *serialize()* de la clase Rule para poder concatenar la regla. Este proceso se realiza dentro de un bloque try, con el objetivo de capturar algún error producido al insertar las reglas.

3.9.7 MÉTODO CONEXIONCLIENTESSL()

En el Código 6 se puede apreciar el método *conexionClienteSSL()*, el cual establece una conexión cifrada con los clientes. Para lograrlo, crea un socket SSL a partir del cual se instancia un objeto de la clase ConexiónSSL, la cual se encarga de autentificar a los clientes.

```
1.  public static void conexionClienteSSL (){
2.      try{
3.         Socket cliente = null;
4.         System.setProperty("javax.net.ssl.trustStore", SERVER_KEY_STORE);
5.         SSLContext context = SSLContext.getInstance("TLS");
6.         KeyStore ks = KeyStore.getInstance("jceks");
7.         ks.load(new FileInputStream(SERVER_KEY_STORE), null);
8.         KeyManagerFactory kf = KeyManagerFactory.getInstance("SunX509");
9.         kf.init(ks, SERVER_KEY_STORE_PASSWORD.toCharArray());
10.        context.init(kf.getKeyManagers(), null, null);
11.        ServerSocketFactory factory = context.getServerSocketFactory();
12.        ServerSocket _socket = factory.createServerSocket(8443);
13.        ((SSLServerSocket) _socket).setNeedClientAuth(false);
14.
15.        while(true){
16.            cliente = _socket.accept();
17.            if(cliente.isConnected())
18.                {
19.                System.out.println("Usuario conectado");
20.                Thread hcliente = new Thread(new ConexionSSL(cliente));
21.                hcliente.start();
22.                }
23.         }
24.     }catch (Exception e) {
25.         Logger.getLogger(ServidorAuth.class.getName()).
26. log(Level.SEVERE, null, e);
27.     }
28. }
```

Código 6: Método conexionClienteSSL().

En el primer fragmento de código (línea 3 a 13), utilizando el keystore y las claves generadas en la sección 3.6.5, se define un socket para escuchar las peticiones de conexión de los clientes a través del puerto TCP 8443. Posteriormente (línea 15 a 23), se define un lazo *while* con el cual se escuchan permanentemente las solicitudes de conexión de los clientes. Cada vez que un cliente se conecta se crea e inicializa un objeto del tipo ConexiónSSL. Todo el proceso se realiza dentro de un bloque try, con el objetivo de gestionar las posibles excepciones que se produjeran.

3.9.8 MÉTODO LOG()

EL Código 7 corresponde al método *Log()*, el cual es encargado de registrar la información de inicio de sesión de los clientes en un archivo de registro, utilizando la API "Java Logging" (Oracle, Java Logging Overview, 2011).

```
1.  public static void Log() {
2.
3.          Handler fileHandler;
4.          try {
5.               fileHandler = new FileHandler("auth.log");
6.               SimpleFormatter simpleFormatter =new SimpleFormatter();
7.               fileHandler.setFormatter(simpleFormatter);
8.
9.               LOGGER.addHandler(fileHandler);
10.              fileHandler.setLevel(Level.ALL);
11.
12.         }catch(SecurityException | IOException e){
13.                e.printStackTrace();
14.              }
15. }
```

Código 7: Método Log().

Como se mencionó en la explicación del método *main()* en sección 3.9.2, el método *Log()* es llamado al iniciar la aplicación Servidor, con el objetivo de instanciar los objetos necesarios para su posterior utilización. La primer tarea del método es la creación un objeto *FileHandler* (línea 5), el cual es el encargado de escribir los registros formateados en un archivo de registro denominado "*auth.log*". A continuación se crea un objeto *SimpleFormatter* (línea 6) y se lo establece en el *FileHandler* (línea 7), de este modo los registros se escribirán en el archivo como texto plano simple, de no indicarlo se escribirían por defecto en formato XML.

Junto con la clase ServidorAuth se ha creado un objeto estático de tipo *Logger* denominado LOGGER, al cual se le agrega el *FileHander* (línea 9) creado anteriormente. Todo el proceso se realiza dentro de un bloque try, con el objetivo de gestionar las posibles excepciones que se produjeran.

Para registrar un mensaje en el archivo, solo se deberá llamar al método log del *Logger*, indicando el nivel del log y el mensaje que deseamos registrar, de esta forma se reportará la fecha, la hora, el mensaje y el nivel del mismo. En el Código 8 de la sección 3.9.9 correspondiente a la clase ConexionSSL se puede observar como como se utiliza el objeto LOGGER para registrar mensajes (línea 14, 26, 51 y 55).

Es importante aclarar que se modificó el archivo *logging.properties* ubicado en */usr/lib/jvm/java-7-openjdk-amd64/jre/lib/* con el objetivo de formatear los registros según las necesidades de la aplicación. Para ello se agregó la siguiente línea:

```
java.util.logging.SimpleFormatter.format=%1$tY/%1$tm/%1$td %1$tH:%1$tM:%1$tS %4$s: %5$s%n
```

El significado de cada parámetro se puede consultar en (Oracle, Class SimpleFormatter, 2017).

3.9.9 CLASE CONEXIÓNSSL

Como se mencionó en la sección 3.9.7, esta clase es instanciada desde el método *conexionClienteSSL()* de la clase ServidorAuth, mediante un hilo de ejecución por cada cliente que se conecta. Su función es la recepción de las credenciales de usuarios, la comprobación de las mismas y permitir o denegar el acceso a la red insertando reglas de firewall en el controlador.

```
1.  public class ConexionSSL extends ServidorAuth implements Runnable{
2.  Socket cliente;
3.  public ConexionSSL(Socket cliente)
4.   {
5.     this.cliente = cliente;
6.     }
7.  public void run(){
8.   try{
9.       DeviceSummary host = null;
10.      for(DeviceSummary obj : dispositivos){
11.      if(obj.getIpv4()!=null){
12.        if(obj.getIpv4().equals(cliente.getInetAddress().getHostAddress())){
13.            System.out.println("Direccion IP: "+obj.getIpv4());
14.            LOGGER.log(Level.INFO,"["+cliente.hashCode()+ "] Usuario
                                    Conectado: Direccion IP: "+obj.getIpv4());
```

```
15.             host = obj;
16.             break;
17.             }
18.         }
19.         }
20.     BufferedReader reader = new BufferedReader(new InputStreamReader(cliente.
                                                        getInputStream()));
21.     PrintWriter writer = new PrintWriter(cliente.getOutputStream());
22.     salto:
23.     for(int intentos=0;intentos<3;intentos++){
24.       String userName = reader.readLine();
25.       String password = reader.readLine();
26.       LOGGER.log(Level.INFO,"["+cliente.hashCode()+ "] User: " + userName +"
                                                          Intentos: " +intentos);
27.
28.       File ficheroEnt = new File ("/home/alp/claves/usuarios");
29.       if (!ficheroEnt.exists()) {System.err.println ("No existe el fichero
                                                        de entrada especificado");
30.         }else{
31.             Scanner scan1 = new Scanner(ficheroEnt);
32.             Scanner scan2 = new Scanner(ficheroEnt);
33.             int contador=0, contador1 =0;
34.             while(scan2.hasNext()){
35.               scan2.nextLine();
36.               contador ++;
37.               }
38.          scan2.close();
39.             while(scan1.hasNext()){
40.               String linea = scan1.nextLine();
41.               contador1 ++;
42.               try{
43.     if(userName.equals(linea.split(",")[0])&& password.equals(Cifrar.
                                              descifrado(linea.split(",")[1]))){
44.         System.out.print("Usuario y contraseña correctos" + userName);
45.         LOGGER.log(Level.INFO,"["+cliente.hashCode()+ "] Usuario y
                                              contraseña correcto: " + userName );
46.         setRegla("",host.getMacAddress(),"","",host.getIpv4(),"32","",
                                  "","","","","", "", host.getAttachedSwitch());
47.         writer.println("valido");
48.         break salto;
49.     }else if(contador == contador1 && intentos<2){
50.        System.out.print("Usuario o contraseña incorrectos\n");
51.        LOGGER.log(Level.WARNING,"["+cliente.hashCode()+ "] Usuario o
                                                  contraseña incorrectos");
52.        writer.println("invalido,true");
53.      }else if(contador == contador1 && intentos==2){
54.        System.out.print("Usuario o contraseña incorrectos.
                                                  \n Intentos superados");
55.        LOGGER.log(Level.WARNING,"["+cliente.hashCode()+ "] Usuario o contraseña
                                incorrecto / Número máximo de intentos superados");
56.
57.        writer.println("invalido,false");
58.        setRegla(Integer.toString(host.getSwitchPort()),
                    host.getMacAddress(),"","","", "","","","","","","DENY","",
                                                        host.getAttachedSwitch());
59.        Tiempo temp = new Tiempo(host.getLastSeen(),host.getMacAddress());
60.        bloqueados.add(temp);
61.       }
62.               }catch(Exception e){
63.                  e.printStackTrace();
64.               }
65.             }//while
66.             scan1.close();
67.           }//else fichero
68.           writer.flush();
69.     }//fin for
70.     writer.close();
71.     cliente.close();
72.     }catch(IOException e){
```

```
73.         System.err.println(e);
74.     }
75.   }//run
76. }//fin clase
```

Código 8: Clase ConexionSSL.

El primer fragmento del Código 8 tiene como objetivo localizar los datos del cliente que se ha conectado, para ello se recorre la lista dispositivos (línea 10 a 18) y se compara la dirección IP de cada objeto de la lista con la dirección IP extraída de los datos del socket, proporcionados cuando se creó la instancia de la clase. Cabe mencionar que la lista de dispositivos se hereda de la clase ServidorAuth y es actualizada cada vez que se ejecuta el método *borrarReglasAllow()* de la misma clase.

Posteriormente se reciben las credenciales de usuario a través del socket (líneas 24 y 25) y se procede a compararlas con las credenciales almacenadas en un archivo de texto (línea 43). Este paso se realiza dentro de un bloque *while* con el objetivo de ir comparando cada uno de los usuarios y contraseñas almacenadas en el archivo. Es importante mencionar que las contraseñas se encuentran cifradas y se recurre al método *descifrar()* de la clase Cifrar para poder realizar la comprobación.

Una vez realizada la comprobación se tiene 3 escenarios:

1°_ El usuario y la contraseña son correctos por lo que se inserta una regla en el controlador para permitir el acceso al equipo del cliente. Además se le comunica a la aplicación cliente mediante el socket que el ingreso se ha realizado correctamente (líneas 43 a 48).

2°_ El usuario o la contraseña son incorrectos pero no se ha superado el número de intentos permitidos, por lo que se comunica a la aplicación cliente que los datos son inválidos para que el cliente vuelva a ingresarlos (línea 49 a 52).

3°_ El usuario o la contraseña son incorrectos y se ha superado el número de intentos establecido. Por lo que se comunica a la aplicación cliente que se han superado el número de intentos y se ha denegado el acceso. Acto seguido se inserta una regla en el controlador para denegar el acceso al cliente (línea 53 a 60).

Es importante señalar que en caso de que se cumpla el tercer escenario, se agrega a la lista "*bloqueados*" un objeto de tipo Tiempo (línea 60), el cual incluye la dirección MAC del dispositivo y la hora de la última conexión.

3.9.10 CLASE TIEMPO

El Código 9 corresponde a la clase Tiempo, la cual tiene como objetivo informar cuando un dispositivo bloqueado ha superado 10 minutos desde que se le negó el acceso.

```
1.  public class Tiempo extends ServidorAuth {
2.  Date LastSeen;
3.  String mac;
4.
5.  public static int diferencia(Date lastseen)
6.  {
7.        Date actual=new Date();
8.        long diferencia=(actual.getTime()- lastseen.getTime());
9.        long segundos =diferencia /1000;
10.       long horas=segundos/3600;
11.       segundos-=horas*3600;
12.       long minutos =segundos/60;
13.       segundos-=minutos*60;
14.       return (int)minutos;
15. }
16.
17. public static boolean difTiempo(DeviceSummary aux)
18. {
19.       for(Tiempo tmp : bloqueados){
20.          if(tmp.getMac().equals(aux.getMacAddress())){
21.             if(diferencia(tmp.LastSeen)>10){
22.                bloqueados.remove(tmp);
23.                return true;
24.             }
25.          }
26.       }
27.    return false;
```

```
28.     }
29.
30. public Tiempo(Date lastSeen, String mac)
31. {
32.     this.LastSeen = lastSeen;
33.        this.mac = mac;
34. }
35. }
```

Código 9: Clase Tiempo.

Para su funcionamiento cuenta con dos métodos:

- Método *diferencia():* (línea 5 a 15), el mismo recibe como argumento la última hora de conexión de un dispositivo bloqueado y calcula la diferencia con la hora actual en milisegundos. En este caso retorna la diferencia en minutos, pero de ser necesario puede retornarla en horas, minutos y segundos.
- Método *difTiempo():* (línea 17 a 28), el objetivo de este método es devolver valores booleanos, true cuando un dispositivo ha superado los 10 minutos de bloqueo y false cuando todavía nos los supera o no se encuentra en la lista de dispositivos bloqueados. Es llamado desde el método *borrarReglasDeny()* de la clase ServidorAuth. Para lograr el objetivo recibe como argumento un objeto del tipo "*dispositivo*" y utiliza una la lista de objetos de tipo "*Tiempo*" denominada *bloqueados* (línea 19). En la que como su nombre lo indica, se encuentran los dispositivos bloqueados y la hora en que se produjo el bloqueo. La primera verificación es comprobar que el dispositivo recibido como parámetro se encuentre dentro de la lista de dispositivos bloqueados (línea 20). Si el dispositivo se encuentra en la lista y se comprueba que el tiempo de bloqueo superó los 10 minutos (línea 21), se procede a borrarlo de la lista de *bloqueados* y devolver el valor *true* para que sea eliminada la regla de firewall que lo mantiene bloqueado. Es importante destacar que este valor es totalmente configurable y es este caso por ser un prototipo se ha ajustado a 10 minutos.

3.9.11 CLASE EJECUTAR

El Código 10 define la clase Ejecutar, la misma se encarga de llamar a los métodos necesarios para cargar la información de los switches, de las PC de los clientes y borrar las reglas de clientes desconectados. Como se mencionó anteriormente en la sección 3.9.2, una instancia de esta clase es creada mediante el lanzamiento de un hilo de ejecución, en el método *main()* de la clase ServidorAuth al iniciar la aplicación.

```
1.  public class Ejecutar extends ServidorAuth implements Runnable {
2.  public void run(){
3.          try{
4.             do{
5.               datosSwitches();
6.                borrarReglasAllow();
7.                borrarReglasDeny();
8.               Thread.sleep(5000);
9.             }while(true);
10.         }catch(InterruptedException e){}
11.    }
12. }
```

Código 10: Clase Ejecutar.

Se puede observar la definición de un lazo *do-while* en el que se llama a los métodos: *datosSwitches()*, *borrarReglasAllow()* y *borrarReglasDeny()*. Los cuales fueron explicados en las secciones 3.9.3, 3.9.4 y 3.9.5 respectivamente.

Posteriormente se utiliza la función *sleep()* (línea 8) para dormir al hilo durante 5 segundos, antes de volver a ejecutar nuevamente los métodos.

3.9.12 CLASE ESCUCHARCONEXIONES

El Código 11 corresponde a la clase EscucharConexiones, cuya funcionalidad es atender las solicitudes de conexión de los clientes, para lo cual utiliza el método *conexionClienteSSL()* de la clase ServidorAuth. Como se mencionó anteriormente en la sección 3.9.2, una instancia de esta clase es creada mediante el lanzamiento de un hilo de ejecución, en el método *main()* de la clase ServidorAuth al iniciar la aplicación.

```
1.  public class EscucharConexiones extends ServidorAuth implements Runnable{
2.  public void run()
3.          {
4.          conexionClienteSSL();
5.          }
6.   }
```

Código 11: Clase EscucharConexiones.

3.9.13 CLASE CIFRAR

Debido a que las contraseñas de los usuarios se encuentran cifradas y almacenadas en un archivo, al momento de autentificar a los usuarios es necesario descifrarlas. Para ello se han creados dos métodos tomando como referencia el código publicado por Julio Chinchilla (Chinchilla, 2015).

```
1.  public class Cifrar{
2.    private final static String alg = "AES";
3.    private final static String cI = "AES/CBC/PKCS5Padding";
4.    private static String clave = "92AE31A79FEEB2A3"; //llave
5.  private static String vector = "0123456789ABCDEF"; //vector_inicialización
6.
7.    public static String cifrado(String texto) throws Exception {
8.    Cipher cif = Cipher.getInstance(cI);
9.      SecretKeySpec skeySpec = new SecretKeySpec(clave.getBytes(), alg);
10.     IvParameterSpec ivParameterSpec = new IvParameterSpec(vector.getBytes());
11.     cif.init(Cipher.ENCRYPT_MODE, skeySpec, ivParameterSpec);
12.     byte[] claveCifrada = cif.doFinal(texto.getBytes());
13.     return new String(encodeBase64(claveCifrada));
14.     }
15.   public static String descifrado(String textoCifrado) throws Exception {
16.   Cipher cif = Cipher.getInstance(cI);
17.     SecretKeySpec skeySpec = new SecretKeySpec(clave.getBytes(), alg);
18.     IvParameterSpec ivParameterSpec = new IvParameterSpec(vector.getBytes());
19.     byte[] enc = decodeBase64(textoCifrado);
20.     cif.init(Cipher.DECRYPT_MODE, skeySpec, ivParameterSpec);
21.     byte[] claveDescifrada = cif.doFinal(enc);
22.     return new String(claveDescifrada);
23.    }
24. }
```

Código 12: Clase Cifrar.

Los métodos desarrollados utilizan el algoritmo AES en modo CBC (IETF, 2003). Tal como su nombre lo referencia, el método *cifrado()* recibe un argumento de tipo *String* en texto plano y devuelve el mismo valor cifrado. A diferencia del método *decifrar()* que recibe un argumento de tipo *String* cifrado y lo retorna descifrado.

3.9.14 CLASE CLIENTESSL

El Código 13 corresponde a la aplicación Cliente, la cual deberá ser ejecutada por los usuarios cada vez que deseen acceder a la red.

```
1.  public class ClienteSSL {
2.    private static String CLIENT_KEY_STORE = "/client_ts";
3.
4.    public static void main(String[] args) throws Exception
5.    {
6.    JTextField ip = new JTextField();
7.    JLabel titulo0 = new JLabel("Ingrese dirección IP del Servidor
                                                    de Autenticación");
8.    int valor0 =JOptionPane.showConfirmDialog (null, new Object[]{titulo0,
9.                        ip}, "Iniciode sesión", JOptionPane.OK_CANCEL_OPTION);
10.   if(valor0==2){ System.exit(1);}
11.
12.   System.setProperty("javax.net.ssl.trustStore", CLIENT_KEY_STORE);
13.   SocketFactory sf = SSLSocketFactory.getDefault();
14.   Socket s = sf.createSocket(ip.getText(), 8443);
15.
16.   PrintWriter writer = new PrintWriter(s.getOutputStream());
17.   BufferedReader reader = new BufferedReader(new InputStreamReader(s.
                                                    getInputStream()));
18.   for(int intentos=0;intentos<3;intentos++)
19.    {
20.    JTextField us = new JTextField();
21.    JLabel titulo = new JLabel ("Ingrese su usuario");
22.    int valor =JOptionPane.showConfirmDialog (null, new Object[]{titulo,us},
                              "Inicio de sesión", JOptionPane.OK_CANCEL_OPTION);
23.    if(valor==2){writer.close();reader.close(); s.close();System.exit(1);}
24.
25.    JPasswordField jpf = new JPasswordField();
26.    JLabel titulo1 = new JLabel ("Ingrese su contraseña");
27.    int valor1= JOptionPane.showConfirmDialog (null, new Object[]{titulo1,
                        jpf}, "Inicio de sesión", JOptionPane.OK_CANCEL_OPTION);
28.
29.    char p[] = jpf.getPassword();
30.    String passw = new String(p);
31.    if(valor1==2){writer.close(); reader.close(); s.close();System.exit(1);}
32.    writer.println(us.getText());
33.    writer.println(passw);
34.    writer.flush();
35.
36.    String mensaje=reader.readLine();
37.
38.    if(mensaje==null){
39.       JOptionPane.showMessageDialog ( null, "No hay respuesta del
                          servidor","ACCESO NEGADO", JOptionPane.ERROR_MESSAGE );
40.       break;
```

```
41.    }else if(mensaje.equals("valido")){
42.       JOptionPane.showMessageDialog ( null, "Bienvenido a la red","ACCESO
                                    CONCEDIDO", JOptionPane.INFORMATION_MESSAGE );
43.       break;
44.    }else if(mensaje.split(",")[0].equals("invalido") && mensaje.split(",")
                                                              [1].equals("true")){
45.       JOptionPane.showMessageDialog (null, "Usuario contraseña
                        incorrectos","ACCESO NEGADO", JOptionPane.ERROR_MESSAGE );
46.
47.    }else if(mensaje.split(",")[0].equals("invalido") && mensaje.split(",")
                                                             [1].equals("false")){
48.       JOptionPane.showMessageDialog  ( null, "Usuario o contraseña
                incorrectos.El numero maximo de intentos ha sido superado.",
                                    "ACCESO NEGADO", JOptionPane.ERROR_MESSAGE );
49.       break;
50.    }
51.  }//fin for
52.   reader.close();
53.   s.close();
54. }
```

Código 13: Código clase Cliente.

La aplicación está compuesta por una clase denominada ClienteSSL.java, la que posee un método *main()*, a partir del cual se ejecutan cada una de sus funcionalidades.

Una vez ejecutada, la primera tarea es solicitar al usuario la dirección IP del Servidor de Autenticación (línea 6 a 9). Para lo cual se utilizó un campo *JTextField* como se muestra en la Figura 22.

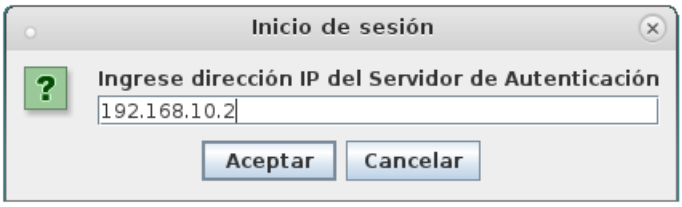

Figura 22 : Inicio de sesión.

Una vez ingresada la dirección IP por parte del usuario se crea el Socket y establece la conexión con el servidor en el puerto TCP 8443 (línea 10 a 17).

Posteriormente dentro de un lazo *for()* y mediante un campo *JTextField* se solicita el usuario y la contraseña (línea 20 a 30), como se puede observar en la Figura 23 y Figura 24.

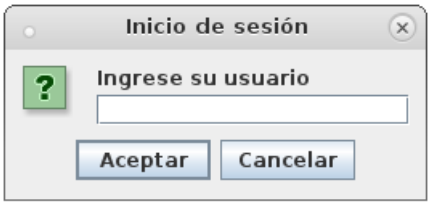

Figura 23 : Ingreso usuario.

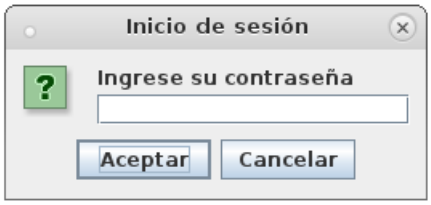

Figura 24: Ingreso contraseña.

Una vez recibidas las credenciales del usuario, son enviadas al servidor de autenticación mediante el socket SSL (línea 32 y 33) y se procede a esperar la respuesta del mismo.

Según el resultado de la comprobación de las credenciales, el servidor puede responder de diferentes maneras, generando cuatro escenarios:

1_ No hay respuesta del servidor: Se produjo algún error de red en la comunicación, lo que impidió recibir la respuesta del servidor. Motivo por el cual se procede a cerrar el socket de comunicación y la aplicación (línea 38 a 40).

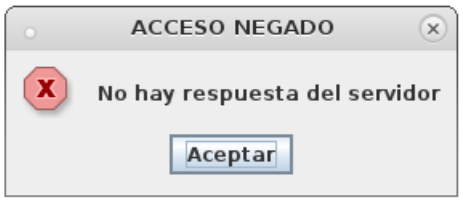

Figura 25 : Error de conexión.

2_ Usuario y contraseña correctos: Las credenciales proporcionadas por el usuario son correctas por lo que se concede el acceso a la red (línea 41 a 43).

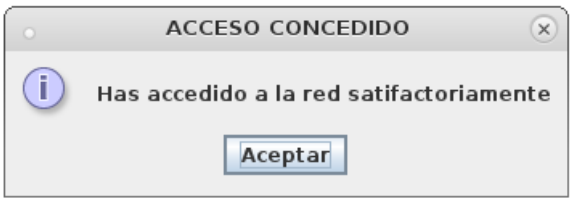

Figura 26 : Acceso concedido.

3_ Usuario o contraseña incorrecta: Las credenciales introducidas por el usuario no son válidas, por lo que se solicita que vuelva a ingresarlas (línea 45).

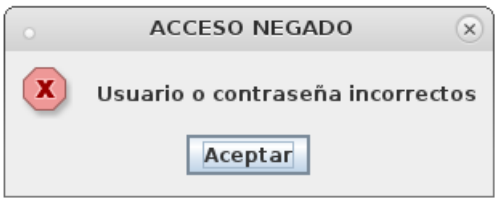

Figura 27 : Credenciales incorrectas.

4_ Usuario o contraseña incorrectos y se superó el límite de intentos permitidos: Las credenciales ingresadas por tercera vez por el usuario son incorrectas, por lo que se le ha negado el acceso y no podrá intentar conectarse nuevamente por un período de 10 minutos (línea 47).

Estos mensajes son generados según las respuestas obtenidas desde el servidor.

3.10 EXPERIMENTACIÓN

En la Figura 28 se observa el escenario utilizado para evaluar el funcionamiento de la aplicación desarrollada. Se conectaron tres equipos; a dos de ellos se les concedió el acceso a la red y uno fue bloqueado

temporalmente. A continuación se expone detalladamente como fue llevado a cabo el procedimiento y la respuesta de cada aplicación durante la prueba.

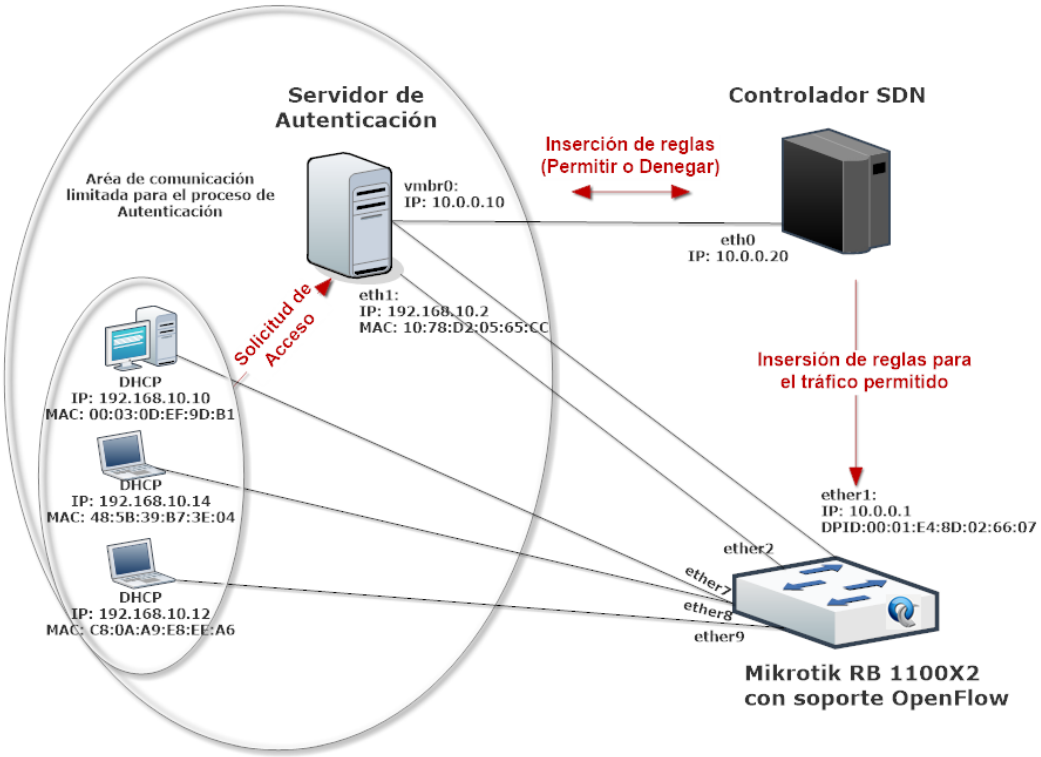

Figura 28: Esquema de red prototipo desarrollado.

Se puede ver la dirección IP y MAC de cada equipo, las cuales serán mencionadas más adelantes en la inserción de reglas. Es importante destacar que el controlador Floodlight se encuentra virtualizado en el mismo equipo físico que utiliza el SA, por lo tanto se conecta al switch mediante la interfaz *vmbr0* de mismo en modo bridge.

Antes de ejecutar la aplicación es necesario iniciar la máquina virtual que contiene al controlador Floodligth, ya que la primera acción que realiza la aplicación es comprobar el estado del módulo Firewall de Floodlight y en caso de estar deshabilitado se habilita. Si bien se ha programado para que por consola se muestre el estado del mismo, también se puede

verificar consultando al controlador desde la línea de comandos mediante el comando *curl* como se aprecia en la Figura 29:

```
root@alp-pc:/# curl http://10.0.0.20:8080/wm/firewall/module/status/json

{"result" : "firewall enabled"}
```

Figura 29: Estado módulo Firewall.

La Figura 30 corresponde a un fragmento de la información arrojada por el controlador cuando se está en ejecución. Se puede observar en la línea 1 que el controlador se encuentra a la espera de conexiones en el puerto 6633. En la línea 3 se informa la solicitud de una nueva conexión desde la dirección IP 10.0.0.1. Posteriormente la línea 4, brinda información del switch que intenta conectase, se puede observar la dirección IP, el DPID, la marca, el modelo y la versión del sistema operativo del dispositivo. La línea 5 informa que se han limpiado las entradas de flujo de dispositivo conectado, y por último en la línea 6 se alerta que el dispositivo se ha conectado correctamente.

```
1.  19:34:16.690 INFO Listening for switch connections on 0.0.0.0/0.0.0.0:6633
2.  19:34:20.902 INFO Starting DebugServer on :6655
3.  19:35:22.919 INFO New switch connection from /10.0.0.1:59260
4.  19:35:23.108 INFO Switch OFSwitchBase [/10.0.0.1:59260 DPID
    [00:01:e4:8d:8c:02:66:c7]], description Switch Desc - Vendor: MikroTik  Model:
    RB1100AHx2 Make: Version: RouterOS 6.39.1  S/N:
5.  19:35:23.119 INFO Clearing all flows on switch OFSwitchBase [/10.0.0.1:59260
    DPID[00:01:e4:8d:8c:02:66:c7]]
6.  19:35:23.129 WARN Switch 00:01:e4:8d:8c:02:66:c7 connected.
```

Figura 30: Log Floodlight.

Una vez iniciado el controlador, conectado el switch y habilitado el módulo Firewall, se insertan las reglas para permitir el tráfico inicial, es decir, que los clientes se puedan conectar al SA. En la Figura 31 se observan cinco reglas insertadas por la aplicación una vez iniciada, las mismas se pueden verificar accediendo a la URL: *http://10.0.0.20:8080/wm/firewall/rules/json*. Cada uno de los parámetros de las reglas se puede consultar en la Tabla 3 de la sección 3.9.6.

```
1. [{"ruleid":276402308,"dpid":"00:01:e4:8d:8c:02:66:c7","in_port":0,"dl_src":"00:
   00:00:00:00:00","dl_dst":"00:00:00:00:00:00","dl_type":2048,"nw_src_prefix":"0.
   0.0.0","nw_src_maskbits":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_pr
   oto":6,"tp_src":0,"tp_dst":8443,"wildcard_dpid":false,"wildcard_in_port":true,"
   wildcard_dl_src":true,"wildcard_dl_dst":true,"wildcard_dl_type":false,"wildcard
   _nw_src":true,"wildcard_nw_dst":true,"wildcard_nw_proto":false,"wildcard_tp_src
   ":true,"wildcard_tp_dst":false,"priority":0,"action":"ALLOW"}

2. {"ruleid":534568974,"dpid":"00:01:e4:8d:8c:02:66:c7","in_port":0,"dl_src":"00:0
   0:00:00:00:00","dl_dst":"00:00:00:00:00:00","dl_type":2048,"nw_src_prefix":"0.0
   .0.0","nw_src_maskbits":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_pro
   to":6,"tp_src":8443,"tp_dst":0,"wildcard_dpid":false,"wildcard_in_port":true,"w
   ildcard_dl_src":true,"wildcard_dl_dst":true,"wildcard_dl_type":false,"wildcard_
   nw_src":true,"wildcard_nw_dst":true,"wildcard_nw_proto":false,"wildcard_tp_src"
   :false,"wildcard_tp_dst":true,"priority":0,"action":"ALLOW"}

3. {"ruleid":1126323778,"dpid":"00:01:e4:8d:8c:02:66:c7","in_port":0,"dl_src":"00:
   00:00:00:00:00","dl_dst":"00:00:00:00:00:00","dl_type":2048,"nw_src_prefix":"0.
   0.0.0","nw_src_maskbits":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_pr
   oto":17,"tp_src":0,"tp_dst":68,"wildcard_dpid":false,"wildcard_in_port":true,"w
   ildcard_dl_src":true,"wildcard_dl_dst":true,"wildcard_dl_type":false,"wildcard_
   nw_src":true,"wildcard_nw_dst":true,"wildcard_nw_proto":false,"wildcard_tp_src"
   :true,"wildcard_tp_dst":false,"priority":0,"action":"ALLOW"}

4. {"ruleid":492174355,"dpid":"00:01:e4:8d:8c:02:66:c7","in_port":0,"dl_src":"00:0
   0:00:00:00:00","dl_dst":"00:00:00:00:00:00","dl_type":2048,"nw_src_prefix":"0.0
   .0.0","nw_src_maskbits":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_pro
   to":17,"tp_src":67,"tp_dst":0,"wildcard_dpid":false,"wildcard_in_port":true,"wi
   ldcard_dl_src":true,"wildcard_dl_dst":true,"wildcard_dl_type":false,"wildcard_n
   w_src":true,"wildcard_nw_dst":true,"wildcard_nw_proto":false,"wildcard_tp_src":
   false,"wildcard_tp_dst":true,"priority":0,"action":"ALLOW"}

5. {"ruleid":800227416,"dpid":"00:01:e4:8d:8c:02:66:c7","in_port":0,"dl_src":"00:0
   0:00:00:00:00","dl_dst":"00:00:00:00:00:00","dl_type":2054,"nw_src_prefix":"0.0
   .0.0","nw_src_maskbits":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_pro
   to":0,"tp_src":0,"tp_dst":0,"wildcard_dpid":false,"wildcard_in_port":true,"wild
   card_dl_src":true,"wildcard_dl_dst":true,"wildcard_dl_type":false,"wildcard_nw_
   src":true,"wildcard_nw_dst":true,"wildcard_nw_proto":true,"wildcard_tp_src":tru
   e,"wildcard_tp_dst":true,"priority":0,"action":"ALLOW"}]
```


Figura 31 : Reglas Iniciales.

Las reglas 1 y 2 permiten el tráfico bidireccional en el puerto TCP 8443 utilizada para la conexión con el SA mediante el socket SSL. De manera similar las reglas 3 y 4 permiten el tráfico UDP en los puertos 67 y 68 que posibilitan la asignación de direcciones IP a los clientes, mediante protocolo DHCP. Por último la regla 5 permite el trafico ARP.

Con las reglas iniciales insertadas, la aplicación se encuentra atendiendo las peticiones de conexión, por lo que se procede a conectar los clientes. En primer lugar se autenticó correctamente a dos clientes a los que se les permitió el acceso. Posteriormente el tercer cliente erró la contraseña tres veces por lo que fue bloqueado. En la Tabla 4 se observan los datos de cada uno de los clientes conectados.

Tabla 4: Datos clientes.

Usuario	Dirección IP	MAC	Puerto SW	SO	Autenticación
apenasco	192.168.10.10	00:03:0d:ef:9d:b1	ether7	Debian 8	Correcta
jperez	192.168.10.14	48:5b:39:b73e:04	ether8	Debian 8	Incorrecta
ggonzalez	192.168.10.12	c8:0a:a9:e8:ee:a6	ether9	Windows7	Correcta

En la Figura 32 se observa la asignación de direcciones IP realizada por el servidor DHCP.

```
root@alp-pc:/# cat /var/lib/dhcp/dhcp.leases

lease 192.168.10.10 {
  starts 4 2017/07/24 20:40:25;
  binding state active;
  next binding state free;
  rewind binding state free;
  hardware ethernet 00:03:0d:ef:9d:b1;
  client-hostname "debian";
}
lease 192.168.10.14 {
  starts 4 2017/07/24 20:41:25;
  binding state active;
  next binding state free;
  rewind binding state free;
  hardware ethernet 48:5b:39:b7:3e:04;
  client-hostname "ZEUS";
}
lease 192.168.10.12 {
  starts 4 2017/07/24 20:41:50;
  binding state active;
  next binding state free;
  rewind binding state free;
  hardware ethernet c8:0a:a9:e8:ee:a6;
  client-hostname "PC-ROMINA";
}
```

Figura 32: Asignación de direcciones IP.

Además, se puede ver en la Figura 33 el cambio de estados de puertos registrado en el log del controlador.

```
20:40:25.729 WARN Switch 00:01:e4:8d:8c:02:66:c7 port ether7 changed: UP
20:40:41.729 WARN Switch 00:01:e4:8d:8c:02:66:c7 port ether8 changed: UP
20:41:50.729 WARN Switch 00:01:e4:8d:8c:02:66:c7 port ether9 changed: UP
```

Figura 33: Estado puertos Floodlight.

La Figura 34 muestra la salida de consola del SA, se pueden diferenciar los clientes que han ingresado correctamente el usuario y la contraseña

logrando el acceso a la red, de aquel que ha colocado la contraseña incorrectamente tres veces y ha sido bloqueado.

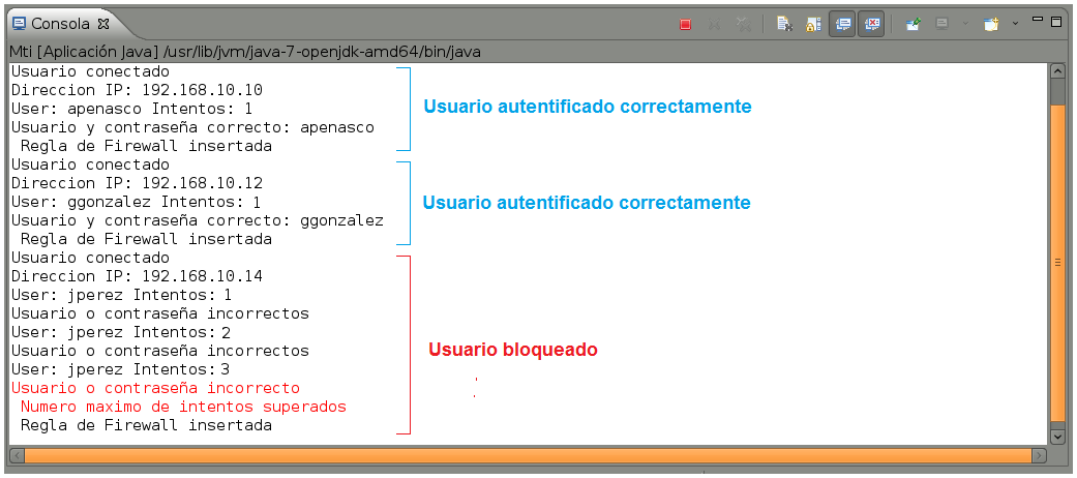

Figura 34: Autenticación de usuarios (consola eclipse).

Como se detalló en la sección 3.9.8, la información de autenticación es registrada en un archivo denominado *auth.log*. En la Figura 35 se puede apreciar cómo se realiza el registro de la información.

```
2018/01/29 20:41:47 INFO: [420235320] Usuario Conectado: Direccion IP:
192.168.10.10
2018/01/29 20:41:49 INFO: [420235320] User: apenasco Intentos: 1
2018/01/29 20:41:49 INFO: [420235320] Usuario y contraseña correcto: apenasco

2018/01/29 20:43:12 INFO: [363205280] Usuario Conectado: Direccion IP:
192.168.10.12
2018/01/29 20:43:30 INFO: [363205280] User: ggonzales Intentos: 1
2018/01/29 20:43:30 INFO: [363205280] Usuario y contraseña correcto: ggonzalez

2018/01/29 20:44:26 INFO: [206280435] Usuario Conectado: Direccion IP:
192.168.10.14
2018/01/29 20:44:43 INFO: [206280435] User: jperez Intentos: 1
2018/01/29 20:44:43 WARN: [206280435] Usuario o contraseña incorrectos
2018/01/29 20:45:22 INFO: [206280435] User: jperez Intentos: 2
2018/01/29 20:45:42 WARN: [206280435] Usuario o contraseña incorrectos
2018/01/29 20:47:31 INFO: [206280435] User: jperez Intentos: 3
2018/01/29 20:47:31 WARN: [206280435] Usuario o contraseña incorrecto/Numero
maximo de intentos superados
```

Figura 35: Log Autenticación de usuarios.

La Figura 36 corresponde a las reglas insertadas para cada cliente.

```
1. {"ruleid":842857164,"dpid":"00:01:e4:8d:8c:02:66:c7","in_port":7,"dl_src":"00:0
   3:0D:EF:9D:B1","dl_dst":"00:00:00:00:00:00","dl_type":2048,"nw_src_prefix":"192
   .168.10.10","nw_src_maskbits":32,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,
   "nw_proto":0,"tp_src":0,"tp_dst":0,"wildcard_dpid":false,"wildcard_in_port":tru
   e,"wildcard_dl_src":false,"wildcard_dl_dst":true,"wildcard_dl_type":false,"wild
   card_nw_src":false,"wildcard_nw_dst":true,"wildcard_nw_proto":true,"wildcard_tp
   _src":true,"wildcard_tp_dst":true,"priority":0,"action":"ALLOW"}

2. {"ruleid":1444981451,"dpid":"00:01:e4:8d:8c:02:66:c7","in_port":8,"dl_src":"C8:
   0A:A9:E8:EE:A6","dl_dst":"00:00:00:00:00:00","dl_type":2048,"nw_src_prefix":"19
   2.168.10.12","nw_src_maskbits":32,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0
   ,"nw_proto":0,"tp_src":0,"tp_dst":0,"wildcard_dpid":false,"wildcard_in_port":tr
   ue,"wildcard_dl_src":false,"wildcard_dl_dst":true,"wildcard_dl_type":false,"wil
   dcard_nw_src":false,"wildcard_nw_dst":true,"wildcard_nw_proto":true,"wildcard_t
   p_src":true,"wildcard_tp_dst":true,"priority":0,"action":"ALLOW"}

3. {""ruleid":220360245,"dpid":"00:01:e4:8d:8c:02:66:c7","in_port":9,"dl_src":"48:
   5B:39:B7:3E:04","dl_dst":"00:00:00:00:00:00","dl_type":0,"nw_src_prefix":"0.0.0
   .0","nw_src_maskbits":0,"nw_dst_prefix":"0.0.0.0","nw_dst_maskbits":0,"nw_proto
   ":0,"tp_src":0,"tp_dst":0,"wildcard_dpid":false,"wildcard_in_port":false,"wildc
   ard_dl_src":false,"wildcard_dl_dst":true,"wildcard_dl_type":true,"wildcard_nw_s
   rc":true,"wildcard_nw_dst":true,"wildcard_nw_proto":true,"wildcard_tp_src":true
   ,"wildcard_tp_dst":true,"priority":0,"action":"DENY"}
```


Figura 36: Reglas clientes.

Las reglas 1 y 2 corresponden a los clientes que lograron autenticarse correctamente y se les permitió el acceso, es importante destacar el puerto al cual se encuentra conectado el dispositivo, la dirección MAC, la dirección IP, la máscara de red y la acción de la regla.

La regla 3 corresponde al cliente que ha sido bloqueado. Al igual que en la regla 1 y 2 se puede observar la dirección MAC del dispositivo, el puerto al que se encuentra conectado y la acción de la regla. En este caso no se incluye la dirección IP de dispositivo ni la máscara, ya que la dirección IP es dinámica y el bloqueo se hace por dirección MAC del dispositivo y puerto al que está conectado.

En la Figura 37 y la Figura 38 se demuestra la conectividad exitosa entre los clientes que pudieron acceder a la red, en este ejemplo, la PC correspondiente a la dirección IP 192.168.10.10, con la PC correspondiente a la dirección IP 192.168.10.12. Así mismo se demuestra la falta de conectividad de estos dos dispositivos hacia la PC bloqueada con dirección IP 192.168.10.14.

Es importante mencionar que el módulo Firewall por defecto bloquea todo tipo de tráfico, por lo que al momento de conexión y hasta que se logre una autenticación correcta, los clientes no tendrán ningún tipo de comunicación más que el permitido por las reglas iniciales insertadas al inicio de la aplicación.

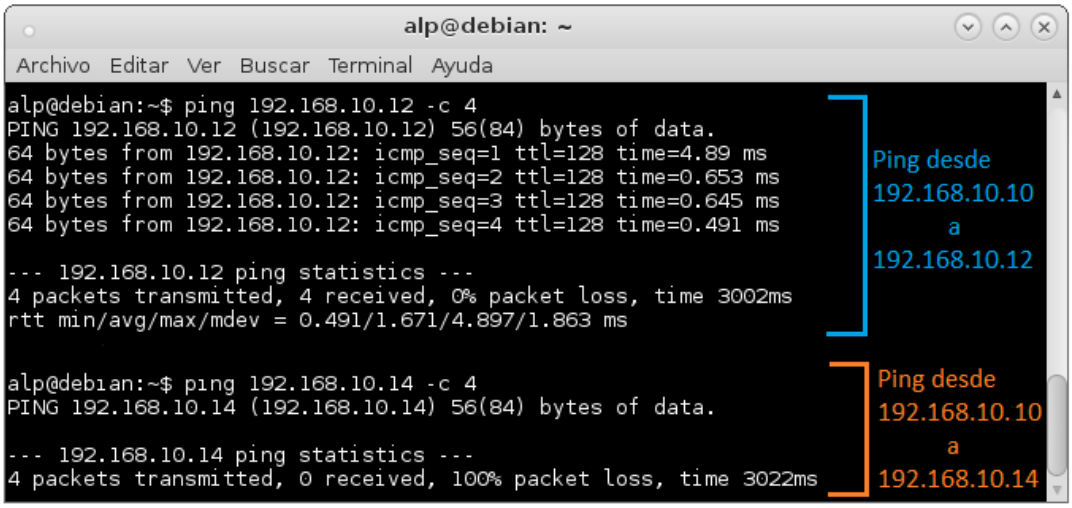

Figura 37 : Conectividad entre equipos con acceso.

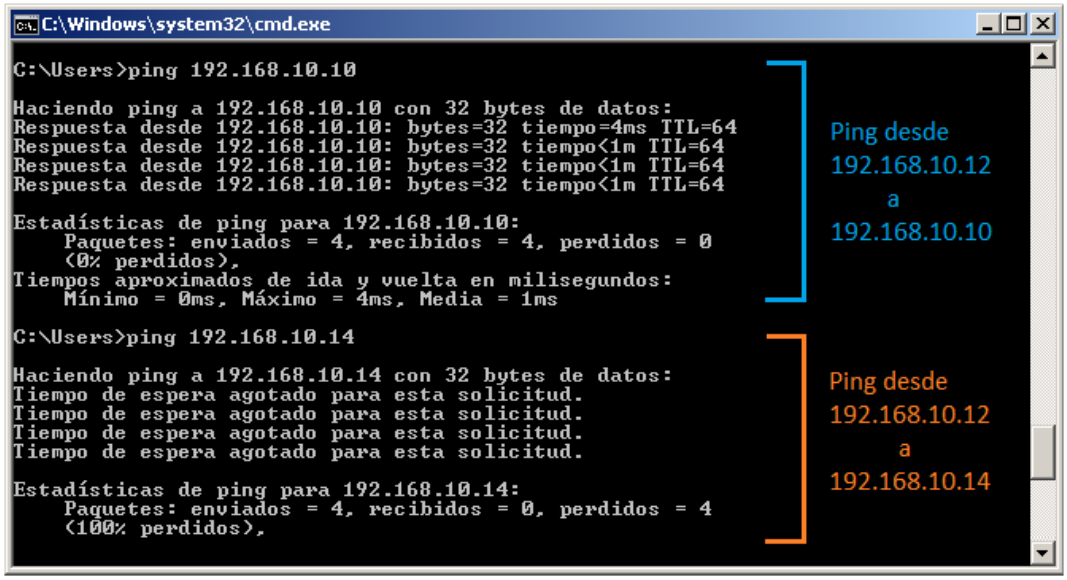

Figura 38 : Conectividad entre equipos con acceso.

La Figura 39 corresponde al equipo bloqueado, se puede apreciar la conectividad fallida hacia los equipos que se encuentran habilitados en la red.

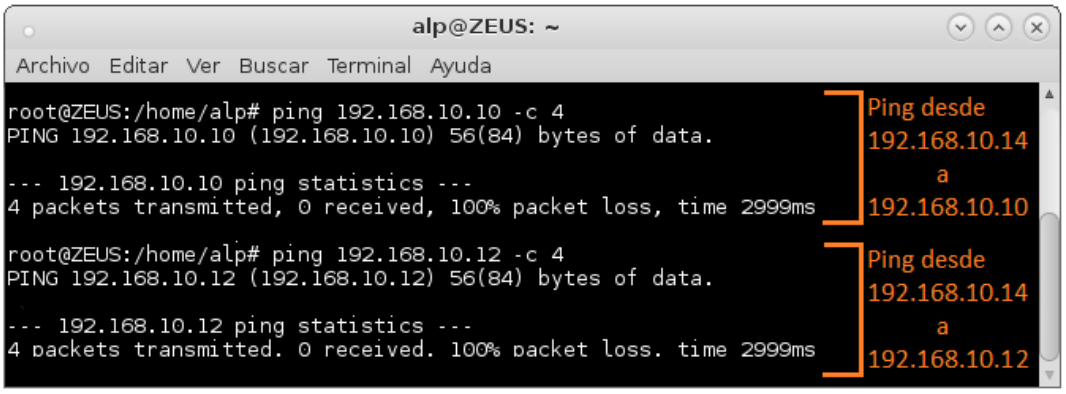

Figura 39: Conectividad equipo bloqueado.

Ahora se procederá a comprobar el borrado de reglas, que como se detalló anteriormente se realiza de diferentes maneras según si son reglas ALLOW o DENY. Inicialmente se desconectaron los dos clientes que accedieron correctamente, es decir con reglas ALLOW. En primer lugar se desconectó el cliente cuya MAC es 00:03:0D:EF:9D:B1, posteriormente de desconectó el cliente cuya MAC es C8:0A:A9:E8:EE:A6. Recordar que estas reglas se borran cuando los clientes de desconectan físicamente del puerto del switch. Para poder apreciar el borrado de las mismas se ha programado para que se muestre en la consola del SA el número de reglas antes y después de que sean eliminadas, como se observa en la Figura 40.

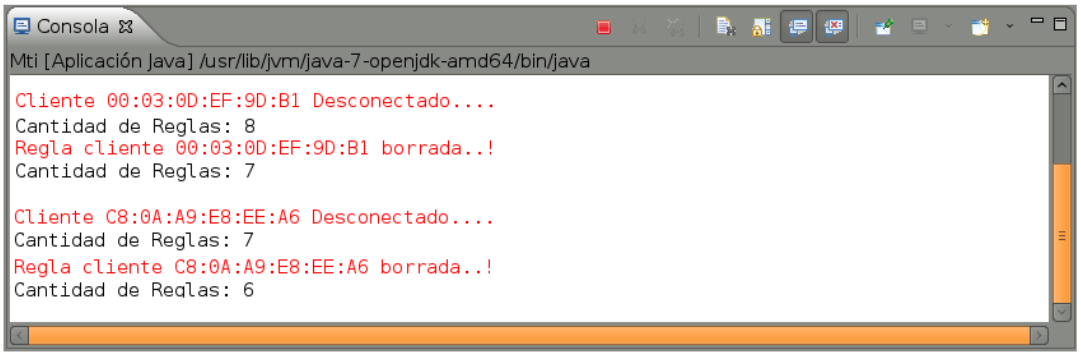

Figura 40 : Borrado de reglas ALLOW.

Transcurridos los 10 minutos desde el bloqueo al cliente que no logro ingresar, se ha borrado automáticamente la regla que le impedía el acceso. La Figura 41 muestra la salida en la consola.

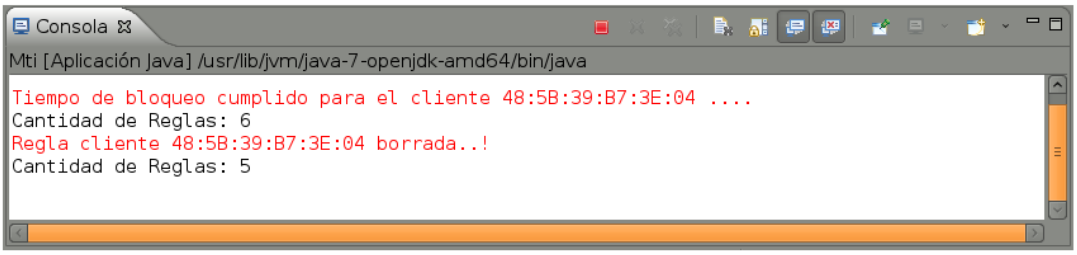

Figura 41 : Borrado de reglas DENY.

Para finalizar, se expone la interfaz web del controlador Floodlight. A continuación se da una breve explicación de la información relevante de cada uno de sus apartados.

La Figura 42 que corresponde al Dashboard de controlador. En ella se puede apreciar los módulos habilitados, información perteneciente al switch Mikrotik y cada uno de los hosts.

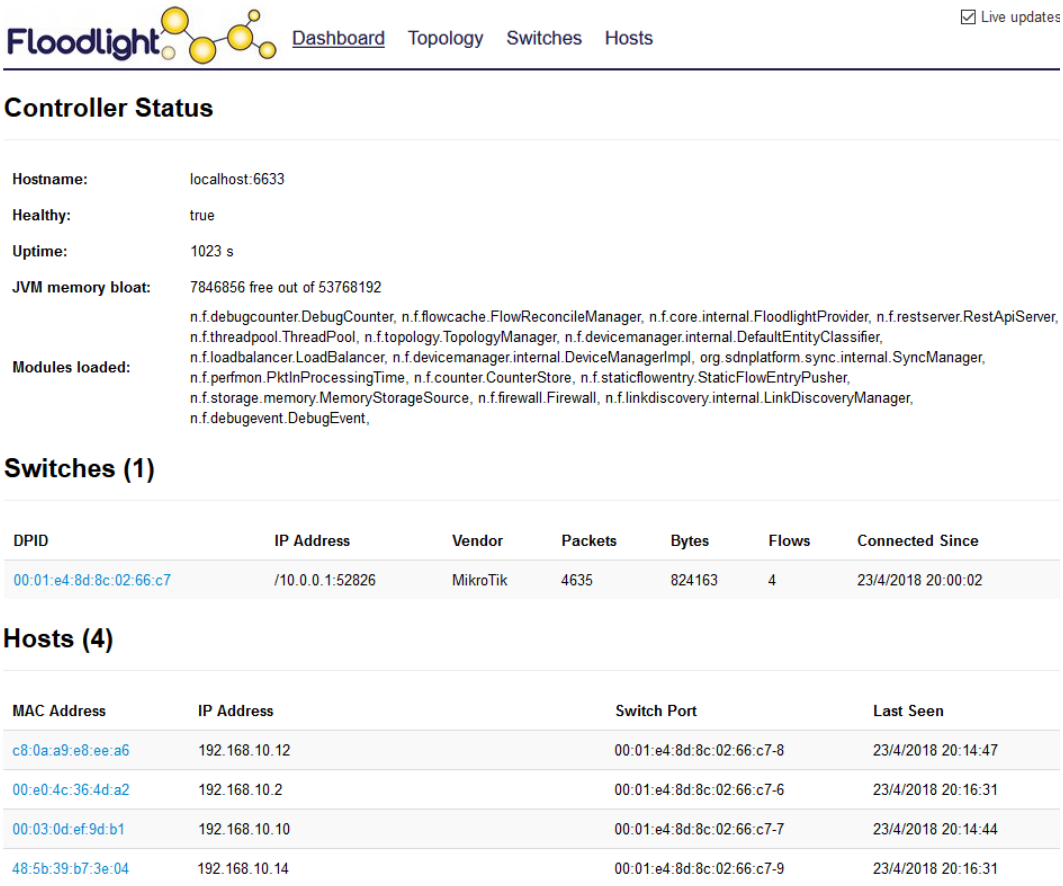

Figura 42: Dashboard Floodlight.

En la Figura 43 se observa el contenido de la pestaña switches. La misma se divide en dos partes. La primera parte muestra información del switch conectado, como el DPID, dirección IP, marca, modelo y versión del sistema operativo. Posteriormente se detalla cada uno de los puertos OpenFlow habilitados. Se puede apreciar el estado “UP” de los puertos 6, 7, 8 y 9. En los cuales se ha conectado al SA y los tres clientes.

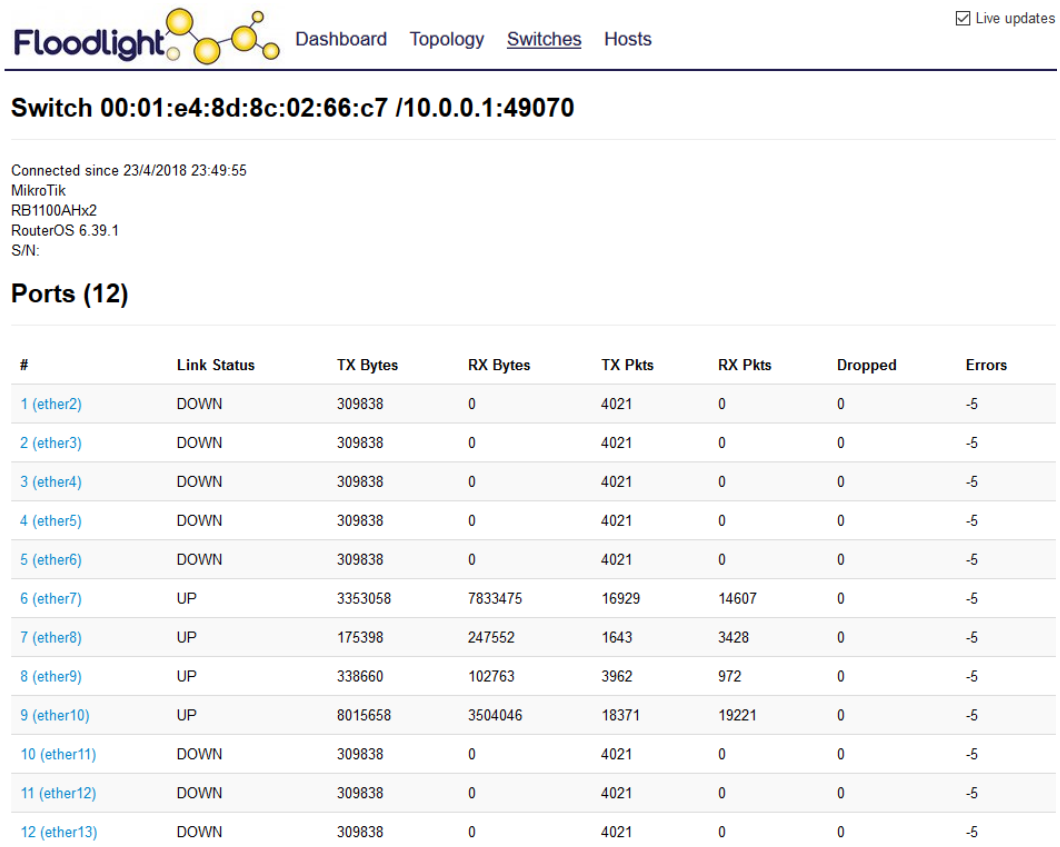

Figura 43: Puertos Floodligth.

La segunda parte de la pestaña switches se puede apreciar en la Figura 44 y Figura 45.

La Figura 44 muestra los flujos de datos ingresados para permitir la comunicación entre los clientes y el SA. Debido a que la comunicación es bidireccional se han enumerado y agrupado de a pares según su correspondencia. Si se observan cada uno de los tres grupos se puede apreciar que la primera regla corresponde a la comunicación desde los clientes al SA. Se detalla dirección IP de origen y destino, dirección MAC de origen y destino, tipo de protocolo IP y puertos de origen y destino. La segunda regla de cada grupo corresponde a la comunicación desde el SA hacia los clientes.

Flows (6)

Cookie	Priority	Match	Action	Packets	Bytes	Age	Timeout	
9007199	0	port=7, src=00:03:0d:ef:9d:b1, dest=00:e0:4c:36:4d:a2, ethertype=0x0800, proto=6, IP src port=-5670, IP dest port=8443, src=192.168.10.10, dest=192.168.10.2	output 6	8	823	2 s	5 s	1
9007199	0	port=6, src=00:e0:4c:36:4d:a2, dest=00:03:0d:ef:9d:b1, ethertype=0x0800, proto=6, IP src port=8443, IP dest port=-5670, src=192.168.10.2, dest=192.168.10.10	output 7	9	2262	2 s	5 s	
9007199	0	port=8, src=c8:0a:a9:e8:ee:a6, dest=00:e0:4c:36:4d:a2, ethertype=0x0800, proto=6, IP src port=-14338, IP dest port=8443, src=192.168.10.12, dest=192.168.10.2	output 6	6	587	1 s	5 s	2
9007199	0	port=6, src=00:e0:4c:36:4d:a2, dest=c8:0a:a9:e8:ee:a6, ethertype=0x0800, proto=6, IP src port=8443, IP dest port=-14338, src=192.168.10.2, dest=192.168.10.12	output 8	9	2045	1 s	5 s	
9007199	0	port=7, src=48:5b:39:b7:3e:04,dest=00:e0:4c:36:4d:a2, ethertype=0x0800, proto=6, IP src port=-5670, IP dest port=8443, src=192.168.10.14, dest=192.168.10.2	output 6	8	823	2 s	5 s	3
9007199	0	port=6, src=00:e0:4c:36:4d:a2, dest=48:5b:39:b7:3e:04,ethertype=0x0800, proto=6, IP src port=8443, IP dest port=-5670, src=192.168.10.2, dest=192.168.10.14	output 9	9	2262	2 s	5 s	

Figura 44: Flujos autenticación clientes.

La Figura 45 datalla los flujos de datos que permite la comunicación ICMP entre los clientes autenticados. Se puede apreciar las direcciones IP, direcciones MAC, puertos OpenFlow de entrada y salida (campo "*Action*") y tipo de protocolo, en este caso ICMP.

Flows (8)

Cookie	Priority	Match	Action	Packets	Bytes	Age	Time
9007199	0	port=7, src=00:03:0d:ef:9d:b1, dest=c8:0a:a9:e8:ee:a6, ethertype=0x0800, proto=1, dest=192.168.10.12	output 8	22	1628	23 s	5 s
9007199	0	port=8, src=c8:0a:a9:e8:ee:a6, dest=00:03:0d:ef:9d:b1, ethertype=0x0800, proto=1, dest=192.168.10.10	output 7	130	11180	65 s	5 s

Figura 45: Flujos de datos ICMP clientes autenticados.

Es importante recordar que como se mencionó en el desarrollo de la tesis, los flujos de datos son ingresado de forma reactiva por el módulo *Forwarding* de Floodlight, y no tienen relación directa con las reglas de firewall que inserta la aplicación desarrollada para controlar el acceso. Por más que existan flujos de datos que permitan la comunicación entre

distintos hosts, dicha comunicación no será posible si hay una regla de firewall que lo prohíba.

Por último, se muestra la Figura 46 la cual corresponde la pestaña Hosts. Se puede apreciar información de cada dispositivo conectado como la dirección MAC, dirección IP, Puerto del switch a cual se encuentra conectado y la hora de última conexión.

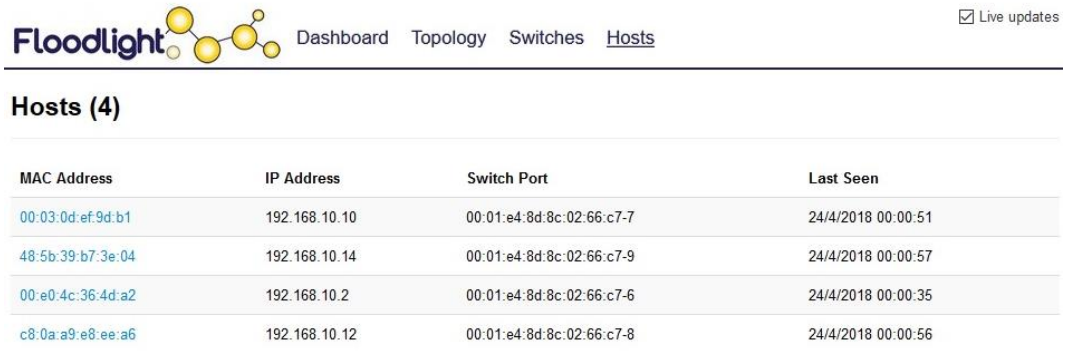

Figura 46: Hosts Floodligth.

CAPÍTULO 4: CONCLUSIONES Y TRABAJO FUTURO

4.1 CONCLUSIONES

La utilización de Redes Definidas por Software abre la posibilidad al desarrollo de nuevas aplicaciones más personalizadas o diseñadas para usos específicos, así mismo permite la integración de nuevos dispositivos y variables a las redes de datos, convirtiéndolas en infraestructuras adaptables y dinámicas. Sin embargo, no están libres de tener inconvenientes de seguridad. Las necesidades actuales de acceder a las redes corporativas utilizando distintos dispositivos, genera la aparición de innumerables puntos débiles. Los que deberán ser abordados en su totalidad antes de poder migrar las tecnologías de red tradicionales a un esquema de red como este.

En este trabajo se llevó a cabo la implementación de una SDN, utilizando equipos físicos y el simulador de red Mininet. Demostrando que se puede configurar y gestionar de forma centralizada los dispositivos SDN según las necesidades requeridas.

Se evaluaron diferentes controladores SDN, eligiendo a FloodLight para la implementación del prototipo desarrollado. La elección se justifica debido a que provee módulos realmente útiles para el desarrollo de aplicaciones, además de ofrecer una API que maneja (JSON/REST) y proporciona una fácil lectura de las acciones de HTTP GET, POST y DELETE. Estas características facilitaron la interacción de la aplicación desarrollada con el controlador. También posee excelente documentación, buena integración con el dispositivo utilizado y soporte por parte de la comunidad de desarrolladores del proyecto.

Se desarrolló un prototipo de aplicación para el control de acceso de usuarios a una arquitectura SDN. Logrando controlar el ingreso de los equipos de usuarios a la red mediante el uso de flujos manejados por el controlador SDN. El método desarrollado es completamente funcional y podría utilizarse sin inconvenientes en un entorno de producción.

Mediante la aplicación creada ha sido posible demostrar que el desarrollo de aplicaciones que gestionan el plano de control de una SDN, pueden ofrecer soluciones completas capaces de solucionar los problemas o debilidades de las redes tradicionales. No solo referentes a la seguridad, sino también a otros aspectos como calidad de servicio, movilidad, ingeniería de tráfico, entre otras.

Para finalizar, es importante destacar que además de desarrollar un método de acceso a una SDN, también se realizó un análisis de diferentes tipos de soluciones disponibles, incluyendo controladores y tipos de switches, como así también la comunicación y configuración de los mismos a través del estándar abierto OpenFlow en sus diferentes versiones, demostrando que SDN es una tecnología que ha llegado para quedarse.

4.2 TRABAJO FUTURO

En esta sección se enumeran posibles mejoras o líneas de investigación a futuro del prototipo desarrollado.

1. Mejoras en el almacenamiento y gestión de las credenciales de usuario: En la aplicación desarrollada las credenciales del usuario se almacenan cifradas en un archivo de texto. Esta forma de almacenar los datos se vuelve difícil de gestionar cuando la cantidad de usuarios aumenta. Se podría pensar en implementar una base de datos que almacene no solo las credenciales, sino también información que pueda

ser utilizada por la aplicación. Por ejemplo, a que VLAN o recursos de red debe tener acceso. Es importante aclarar que en el prototipo desarrollado se realizó el almacenamiento de datos de esa forma ya que no significaba un aporte a los objetivos del proyecto.

2. Automatización del inicio de sesión por parte del usuario: Podría plantearse la ejecución de la aplicación cliente como un servicio. De esta forma el usuario debería proporcionar las credenciales e información del servidor solo la primera vez que se conecte, y que en los inicios de sesión posteriores la conexión sea transparente para el usuario.

3. Inserción de reglas en más de un switch SDN: En aplicación desarrollada los métodos para insertar y borrar reglas trabajan sobre un solo switch. Por lo que al momento de poner esta aplicación en producción en un escenario con más de un dispositivo, es necesaria la modificación de dichos métodos.

4. Agregar un controlador SDN redundante: Debido a que el controlador es un componente vital en la arquitectura SDN, podría ser necesaria la incorporación de un controlador secundario. Con el objetivo de mejorar la disponibilidad de los servicios y facilitar tareas de mantenimiento.

BIBLIOGRAFÍA

Firewall REST API. (Octubre de 2015). Recuperado el Noviembre de 2016, de https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Firewall+REST+API

Floodlight v0.91. (Marzo de 2015). Recuperado el Abril de 2017, de https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Floodlight+v0.91

Forwarding (Dev). (Junio de 2015). Recuperado el Febrero de 2017, de https://floodlight.atlassian.net/wiki/pages/viewpage.action?pageId=1343630

Amer, T., & Izard, R. (marzo de 2015). *Firewall (Dev)*. Recuperado el Enero de 2017, de https://floodlight.atlassian.net/wiki/pages/viewpage.action?pageId=1343599

Chinchilla, J. (Diciembre de 2015). *JAVA Encriptar y Desencriptar AES ECB 128/192/256 bits*. Recuperado el 11 de junio de 2017, de https://bit502.wordpress.com/2016/06/23/java-encriptar-y-desencriptar-aes-ecb-128192256-bits/

Coulouris, G., Dollimore, J., & Kindberg, T. (2011). Distributed Systems: Concepts and Design. En J. D. George Coulouris. Addison Wesley.

Dangovas, V., & Kuliesius, F. (2014). *SDN-Driven Authentication and Access Control System.* Recuperado el 2016, de http://sdiwc.net/digital-library/web-admin/upload-pdf/00001098.pdf

Dangovas, V., & Kuliesius, F. (2016). *SDN Enhanced Campus Network Authentication and Access Control System.* Recuperado el 10 de Septiembre de 2016, de http://ieeexplore.ieee.org/document/7536925/

De León, O., & LACNIC. (2014). *Análisis detallado de la información cuantitativa relevante relativa a la transición hacia una red IPv6.* Recuperado el Marzo de 2016, de http://portalipv6.lacnic.net/caf-lacnic/anexo-iii/

Dietz, T. (Marzo de 2012). *Trema Tutorial.* Recuperado el Abril de 2017, de http://www.fp7-ofelia.eu/assets/Uploads/201203xx-TremaTutorial.pdf

Eclipse Foundatio. (2018). *Eclipse Fundation*. Recuperado el Febrero de 2018, de https://www.eclipse.org/

Erickson, D. (29 de Junio de 2012). *Basic Guides.* Recuperado el Abril de 2017, de https://openflow.stanford.edu/display/Beacon/Guides

Erickson, D. (Marzo de 2012). *The Beacon OpenFlow Controller.* Recuperado el Abril de 2017, de http://yuba.stanford.edu/~derickso/docs/hotsdn15-erickson.pdf

Esmoris, D. O. (2010). *Control de Acceso a Redes.* Recuperado el 2 de Noviembre de 2017, de http://sedici.unlp.edu.ar/bitstream/handle/10915/4182/Documento_completo.pdf?sequence=1

Ferro, G. (Noviembre de 2012). *Network Computing*. Recuperado el Diciembre de 2016, de http://www.networkcomputing.com/networking/sdn-business-openflow-technology/53316220

Gomezcoello Yépez, A. (Febrero de 2017). *Multitenencia Cloud: Una Revisión Sistemática de la Literatura.* Recuperado el Enero de 2018, de http://oa.upm.es/44935/3/TFM_ADAN_GOMEZCOELLO_YEPEZ.pdf

IETF. (Agosto de 2001). *The Secure Sockets Layer (SSL) Protocol Version 3.0.* Recuperado el 17 de Junio de 2017, de https://tools.ietf.org/html/rfc6101

IETF. (Julio de 2003). *Advanced Encryption Standard (AES) Encryption.* Recuperado el Junio de 2017, de https://tools.ietf.org/html/rfc3565

IETF. (Agosto de 2008). *TLS (Transport Layer Security Protocol).* Recuperado el Enero de 2018, de https://tools.ietf.org/html/rfc5246

IETF. (Marzo de 2014). *Software-Defined Networking: A Perspective from within a Service Provider Environment.* Recuperado el 25 de Enero de 2018, de https://tools.ietf.org/html/rfc7149

Internet Systems Consortium. (Abril de 2017). *ISC DHCP*. Recuperado el Mayo de 2017, de https://www.isc.org/downloads/dhcp/

ISO. (2014). *Big Data: Preliminary Report 2014.* Recuperado el septiembre de 2016, de http://www.iso.org/iso/big_data_report-jtc1.pdf

Kreutz, D., Ramos, F., Verissimo, P., Rothenberg, C., Azodolmolky, S., & Uhlig, S. (2015). *Software-Defined Networking:A Comprehensive Survey.* IEEE.

KVM. (2016). *Kernel Virtual Machine*. Obtenido de http://www.linux-kvm.org/page/Main_Page

Marist College. (2013). *What is Avior?* Recuperado el Febrero de 2017, de http://openflow.marist.edu/avior

McKeown, N., Anderson, T., Balakrishnan, H., Guru, P., & Peterson, L. (2008). *Openflow.* Recuperado el septiembre de 2016, de http://archive.openflow.org/documents/openflow-wp-latest.pdf

MikroTik. (Junio de 2002). *MikroTik RouterOS Software Package Installation and Upgrading*. Recuperado el Octube de 2016, de https://www.mikrotik.com/documentation/manual_2.5/Basic/Packages.html

Mikrotik. (Agosto de 2015). *RB1100AHx2*. Recuperado el Mayo de 2017, de https://routerboard.com/RB1100AHx2

Mikrotik. (Septiembre de 2016). *Manual:Upgrading RouterOS*. Recuperado el Diciembre de 2016, de https://wiki.mikrotik.com/wiki/Manual:Upgrading_RouterOS

Mininet. (2016). *Mininet*. Obtenido de http://mininet.org/

MIT. (1988). *The MIT License*. Recuperado el Febrero de 2017, de https://opensource.org/licenses/MIT

Morgan, T. (Abril de 2013). *Theregister*. Recuperado el Mayo de 2017, de https://www.theregister.co.uk/2013/04/08/opendaylight_open_sdn_project/

NIST. (2011). *National Institute of Standards and Technology.* Obtenido de http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

Open Networking Foundation. (2012). *Software-Defined Networking:The New Norm for Networks*. Recuperado el Agosto de 2016, de https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

Open Networking Foundation. (19 de Diciembre de 2014). *OpenFlow Switch Specification.* Recuperado el Enero de 2017, de

https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

Open Networking Foundation. (2016). *Software-Defined Networking (SDN) Definition*. Recuperado el 18 de Agosto de 2016, de https://www.opennetworking.org/sdn-resources/sdn-definition

Open Networking Fundation. (Diciembre de 2009). *OpenFlow swuitch specificaction - Version 1.0.0.* Recuperado el Noviembre de 2017, de https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf

Oracle. (2010). *KeyStores and TrustStores*. Recuperado el Octubre de 2017, de https://docs.oracle.com/cd/E19509-01/820-3503/ggffo/index.html

Oracle. (2011). *Java Logging Overview.* Recuperado el 20 de Febrero de 2018, de https://docs.oracle.com/javase/8/docs/technotes/guides/logging/overview.html

Oracle. (Abril de 2016). *Java Secure Socket Extension (JSSE) Reference Guide*. Recuperado el 12 de Abril de 2017, de https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html#RelatedDocs

Oracle. (Abril de 2016). *JDK Tools and Utilities*. Recuperado el 3 de Mayo de 2017, de https://docs.oracle.com/javase/8/docs/technotes/tools/index.html#security

Oracle. (2017). *Class SimpleFormatter.* Recuperado el 21 de Febrero de 2018, de https://docs.oracle.com/javase/7/docs/api/java/util/logging/SimpleFormatter.html

Oracle. (2018). *Java*. Recuperado el Enero de 2018, de https://www.java.com/es/

Parraga, J. (2013). *Floodlight Network Management and Testing tool*. Recuperado el Enero de 2017, de https://github.com/Sovietaced/Avior

Peñasco, A. (2017). Sistema de Acceso y Autenticación en Redes Definidas por Software. *WICC2017* (págs. 197-201). Buenos Aires: ITBA.

Project Floodlight. (5 de Octubre de 2012). *Application Modules*. Recuperado el Diciembre de 2016, de https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343502/Application+Modules

Project Floodlight. (noviembre de 2016). *Floodlight*. Recuperado el Enero de 2017, de http://www.projectfloodlight.org/floodlight/

Project Floodlight. (Octubre de 2016). *Floodlight REST API*. Recuperado el Noviembre de 2016, de https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343539/Floodlight+REST+API#FloodlightRESTAPI-ModuleRESTAPIDocumentation

Project Floodligth. (Octubre de 2012). *Supported Topologies*. Recuperado el Noviembre de 2016, de https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Supported+Topologies

Project Floodligth. (31 de Mayo de 2017). *Controller Modules*. Recuperado el Mayo de 2017, de https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343500/Controller+Modules

Ryan, I. (Marzo de 2017). *ACL (Access Control List) (Dev)*. Recuperado el Diciembre de 2016, de https://floodlight.atlassian.net/wiki/pages/viewpage.action?pageId=4882434

Stallings, W. (2006). *Data and computer communications* (8° ed.). Prentice Hall.

The Linux Fundation Projects. (2017). *Opendaylight*. Recuperado el Enero de 2017, de https://www.opendaylight.org/

Willens, S., Rigney, C., Rubens, A., & Simpson, W. (Junio de 2000). *Remote Authentication Dial In User Service.* Recuperado el Diciembre de 2016, de https://tools.ietf.org/html/rfc2865

ANEXO I

A continuación se detalla el procedimiento para la generación de un keystore en el SA con su respectivo certificado. Posteriormente se exportó dicho certificado al trutstore de los clientes.

1. Creación Keystore y certificado del SA.

Para la creación del certificado se utilizó el siguiente comando:

```
keytool -genkey -v -alias    servidorAuth -keyalg    RSA -keystore
./server_ks -storepass server -keypass 123123
```

Dónde:

- *Keytool:* como se mencionó anteriormente es la herramienta para la creación de certificados proporcionada por JAVA.
- *-genkey:* indica que genere un certificado.
- *-keyalg RSA*: indica cifrado con el algoritmo RSA.
- *-alias servidorAuth*: indica el nombre con el que luego podremos identificar a este certificado en concreto dentro del keystore.
- *-keystore server_ks:* indica el fichero que hará de almacén de certificados. Si no existe se crea, si ya existe se añade el certificado con el alias que se haya indicado.
- *-storepass:* server indica la contraseña de acceso al keystore. Si el almacén no existe, se crea usando esta contraseña. Si ya existe, debemos proporcionar la contraseña que tuviera ese almacén.
- *-keypass 123123*: indica la clave del certificado.

Normalmente un certificado va asociado a alguna persona u organización, por lo que cuando ejecutemos el comando se nos solicitarán. En este caso, está asociado al servidor de Autenticación.

2. Exportar el certificado del keystore creado.

Como el certificado del servidor generado se encuentra dentro de un almacén, tenemos que exportarlo a un fichero. Se utilizó el siguiente comando:

```
keytool -export -keystore server_ks -alias servidorAuth -file ServidorAuth.cer
```

Dónde:

- *-export*: indica la exportación de un certificado.
- *-keystore serverk_ks*: indica el nombre del almacén en donde está el certificado que queremos exportar.
- *-alias servidorAuth*: es el identificador del certificado dentro del almacén. Es el mismo alias que se utilizó cuando lo creamos.
- *-file ServidorAuth.cer*: es el nombre del fichero donde se va a guardar el certificado que extraído.

Al ejecutar este comando se solicita la contraseña del almacén/certificado que proporcionamos en la creación del mismo.

3. Creación del trutstore del cliente e importación del certificado

Una vez obtenido el fichero ServidorAuth.cer con el certificado, fue necesario crear un almacén de certificados de confianza e introducirlo, para que posteriormente sea utilizado por los usuarios. El comando utilizado es:

```
Keytool -import -alias serverAuth -file ServidorAuth.cer -keystore client_ts -keypass 456456 -storepass clientpass
```

Dónde:

- *-import*: indica que se va a introducir un certificado existente en un almacén.
- *-alias serverAuth*: es el identificador del certificado que vamos a importar dentro del almacén de certificados de confianza del cliente.
- *-file ServidorAuth.cer*: es el certificado que a importar.
- *-keystore client_ts*: es el almacén de certificados de confianza del cliente, el cual se creará si no existe.
- *-keypass 456456*: Indica la clave para proteger el certificado dentro del almacén.
- *storepass clientpass*: Clave para el almacén, si el almacén existe debe ser la introducimos en el momento de crearlo. Si no existe, el almacén se creará, protegido con esta clave.

ANEXO II

En este apartado se realiza la captura de paquetes del protocolo OpenFlow utilizando la herramienta Wireshark. La descripción de cada uno de los mensajes fue extraída de la documentación brindada por la ONF para la versión 1.0 de OpenFLow (Open Networking Fundation, 2009). La captura de paquetes se realizó en el escenario descrito en la Figura 28, sobre la interfaz vmbr0 del Servidor de Autenticación.

HANDSHAKE

En la Figura 47 se observa la secuencia de mensajes enviados entre el controlador y el switch OpenFlow en la comunicación inicial. El controlador SDN se encuentra en constante verificación de conexiones entrantes desde los dispositivos SDN (a través del puerto TCP 6633) y la comunicación se inicia mediante el protocolo TCP (intercambiando mensajes SYN / (SYN / ACK) / ACK), seguido por el protocolo OpenFlow.

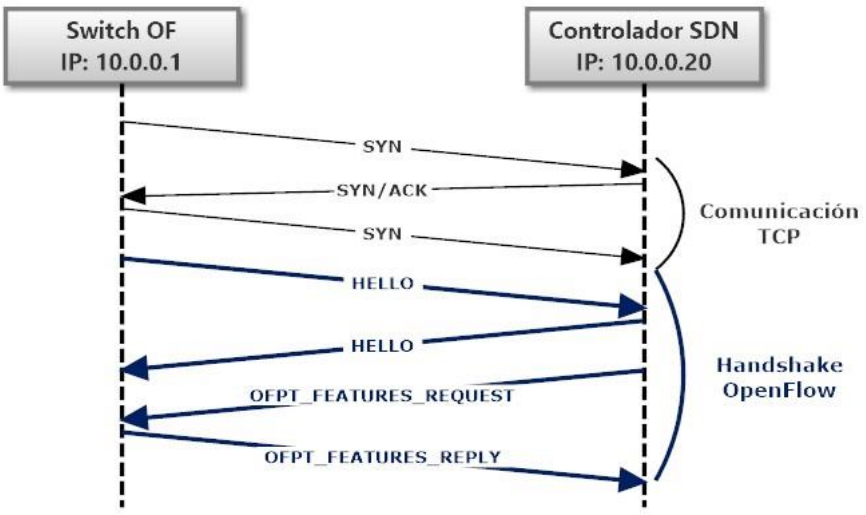

Figura 47: Secuencia de mensajes handshake.

La Figura 48 presenta la captura de paquetes coincidente con el intercambio de mensajes descrito en la Figura 47. El primer paquete OpenFlow enviado contiene el mensaje OFPT_HELLO, es un mensaje de

saludo y no tiene cuerpo; es decir, consiste sólo en un encabezado OpenFlow. En versiones posteriores del protocolo se emplea para negociar la versión de OpenFlow a utilizar.

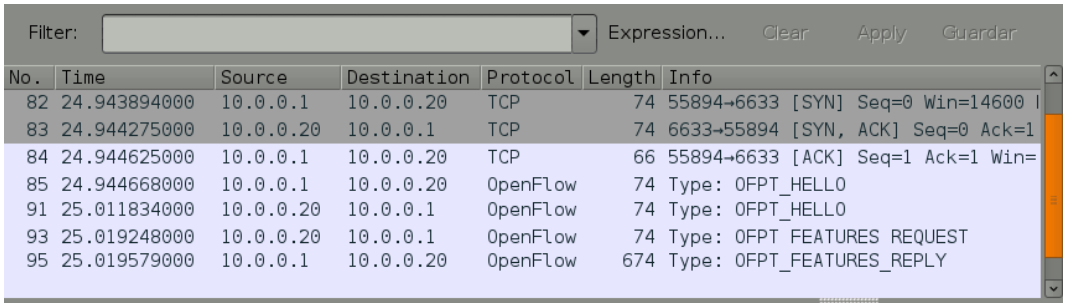

Figura 48: Captura de paquetes handshake.

Posteriormente el controlador envía un mensaje OFPT_FEATURES_REQUEST, que al igual que el mensaje OFPT_HELLO sólo contiene el encabezado OpenFlow y el objetivo es solicitar las características del switch, el cual las informa mediante un mensaje OFPT_FEATURES_REPLY. En la Figura 49 se observa la captura de paquetes correspondiente a esta comunicación.

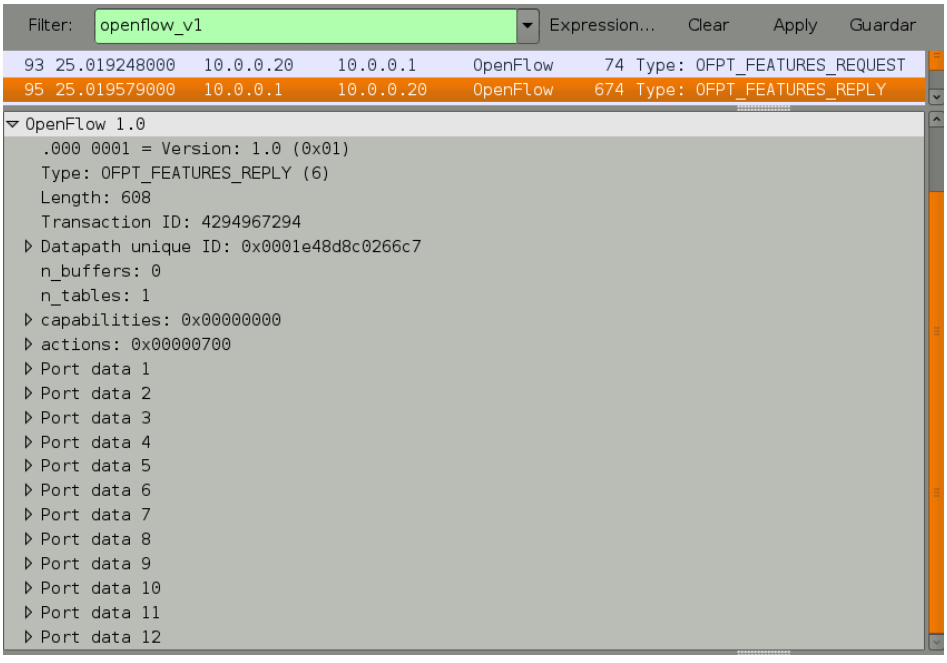

Figura 49: Mensaje OFPT_FEATURES_REPLY.

El campo *datapath_id* identifica de manera única al switch. Los primeros 16 bits son configurables por el operador de la red y los 48 restantes corresponden a la dirección MAC del dispositivo.

El campo *n_tables* describe la cantidad de tablas admitidas por el switch, y cada una de ellas puede tener un número de entradas diferente. Cuando el controlador y el switch se comunican por primera vez, el controlador preguntará cuántas tablas admite el switch. Si desea comprender el tamaño, los tipos y el orden en que se consultan las tablas, el controlador enviará un mensaje de solicitud de estadísticas OFPST_TABLE.

El campo *capabilities* define distintos indicadores que se pueden observar en la Figura 51.

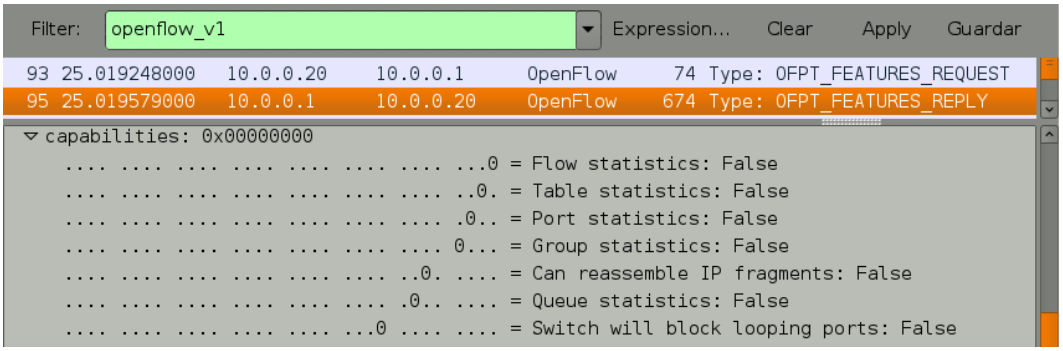

Figura 50: Campo capacidades (OFPT_FEATURES_REPLY).

El campo *actions* describe las acciones admitidas por el switch que se pueden observar en la Figura 51.

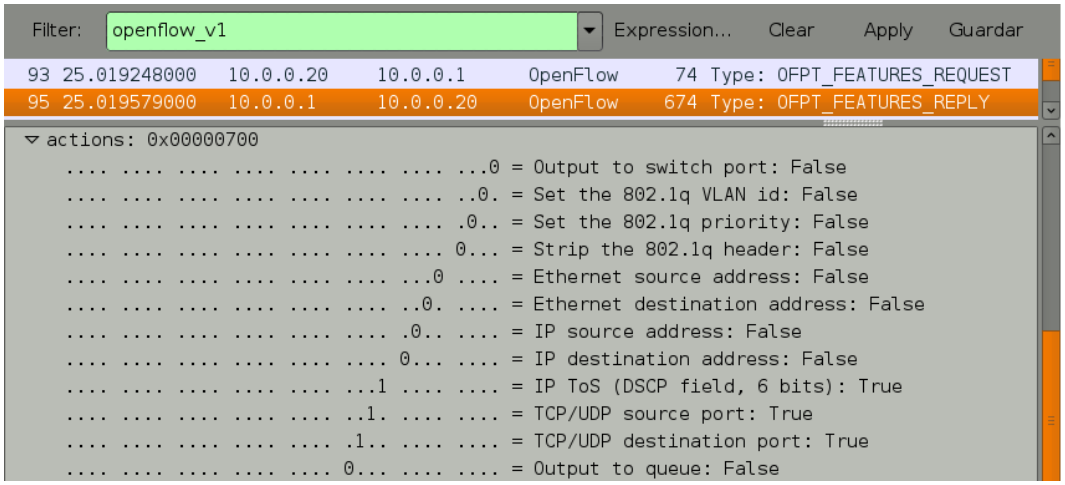

Figura 51 : Campo acciones (OFPT_FEATURES_REPLY).

El campo *ports* describe cada uno de los puertos físicos del sistema que admiten OpenFlow. A modo de ejemplo en la Figura 52 se observa la información que contiene el puerto 1.

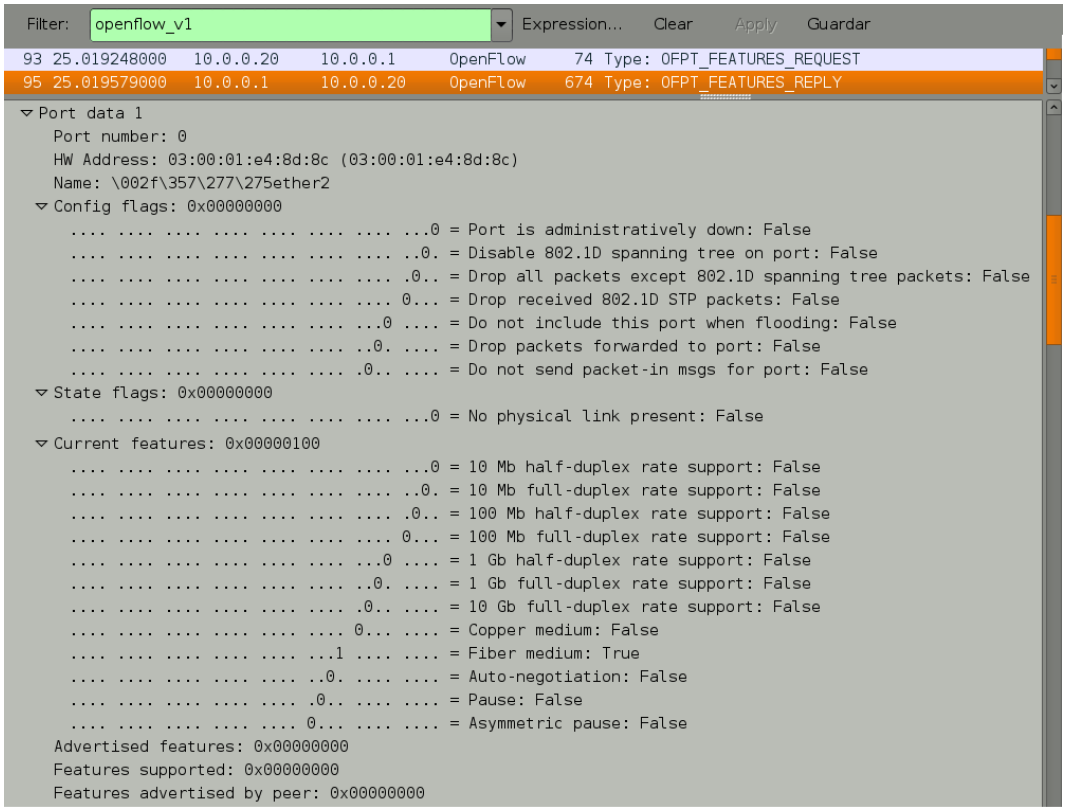

Figura 52: Campo puertos (OFPT_FEATURES_REPLY).

MENSAJES DE CONFIGURACIÓN DEL SWITCH

Una vez realizada la comunicación inicial, el controlador puede consultar y establecer parámetros de configuración del switch. Para ello, envía paquetes con mensajes OFPT_GET_CONFIG_REQUEST y OFPT_SET_CONFIG respectivamente. En este caso particular el controlador envía al switch un mensaje OFPT_GET_CONFIG_REQUEST, al que el switch responde con un mensaje OFTP_GET_CONFIG_REPLY.

El mensaje OFPT_GET_CONFIG_REQUEST enviado por el controlador, sólo contiene un encabezado OpenFlow. A diferencia del mensaje OFPT_GET_CONFIG_REPLY, que en este caso contiene dentro del campo *config_flag* el código OFPC_FRAG_NORMAL indicando que los paquetes fragmentados deben ser tratar normalmente. Es decir, deben intentar pasar a través de las tablas de OpenFlow. Si algún campo no está presente (por ejemplo, los puertos TCP/UDP), entonces el paquete no debería coincidir con ninguna entrada de las tablas de flujo que tengan ese campo establecido. En la Figura 53 se puede observar los paquetes capturados con mensajes OFTP_GET_CONFIG_REQUEST y OFTP_GET_CONFIG_REPLY.

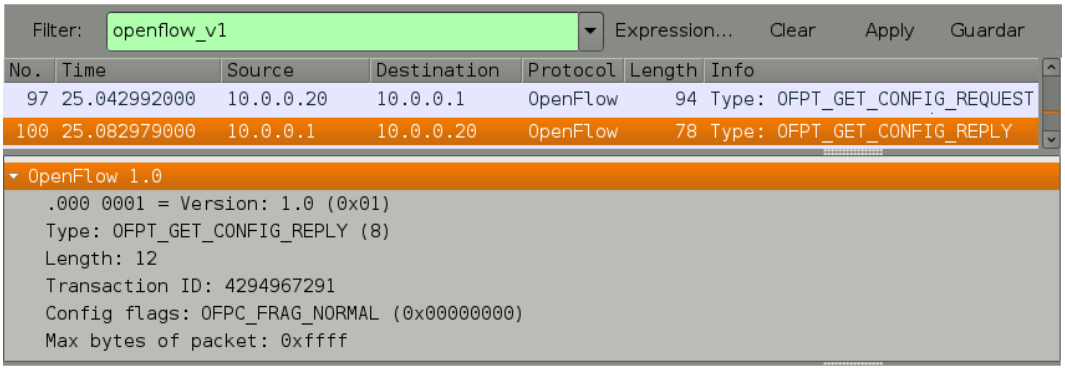

Figura 53: Mensajes OFPT_GET_CONFIG

M ENSAJES DE ESTADO

El controlador puede consultar en cualquier momento el estado del switch, para ello utiliza mensajes OFPT_STATS_REQUEST. En la Figura 54 se observa el mensaje OFPT_STATS_REQUEST enviado por el controlador. Es importante destacar el campo *stats_type*, el cual contiene el código OFPST_DESC y tiene como objetivo solicitar al switch la descripción del mismo.

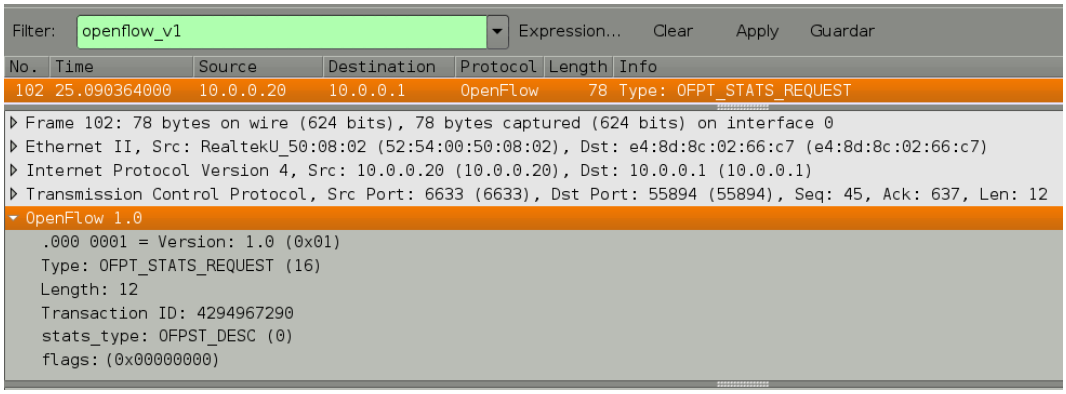

Figura 54: Mensaje OFPT_STATS_REQUEST.

La Figura 55 corresponde al mensaje OFPT_STATS_REPLY enviado por el switch al controlador en respuesta al mensaje OFPT_STATS_REQUEST. Se puede observar la descripción del dispositivo conectado (marca, modelo y versión de sistema operativo). Otro valor a destacar es el ID de transacción, el mismo se encuentra en el campo *Transaction ID* y debe coincidir con el ID del paquete que solicitó la consulta.

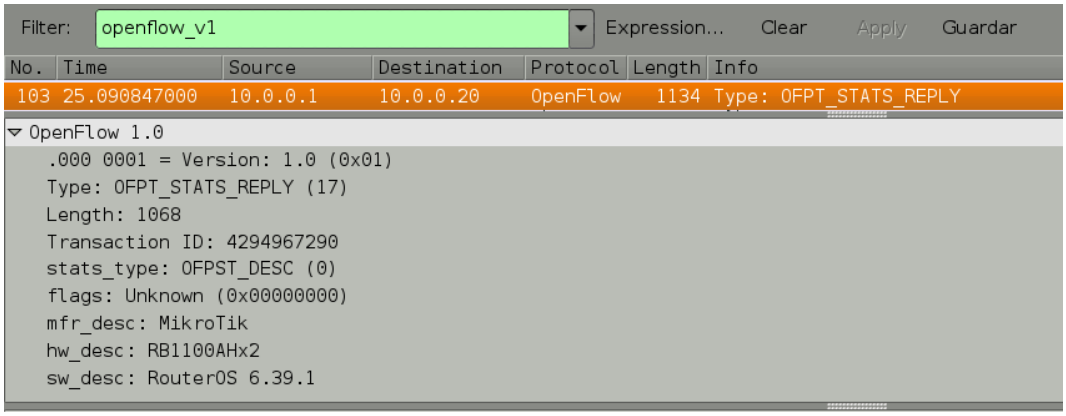

Figura 55: Mensaje OFPT_STATS_REPLY.

MENSAJES DE ESTADOS DE PUERTOS

OpenFlow permite agregar, modificar y eliminar puertos físicos del switch, cada cambio realizado es necesario informarlo al controlador. Para ello utiliza el mensaje OFPT_PORT_STATUS.

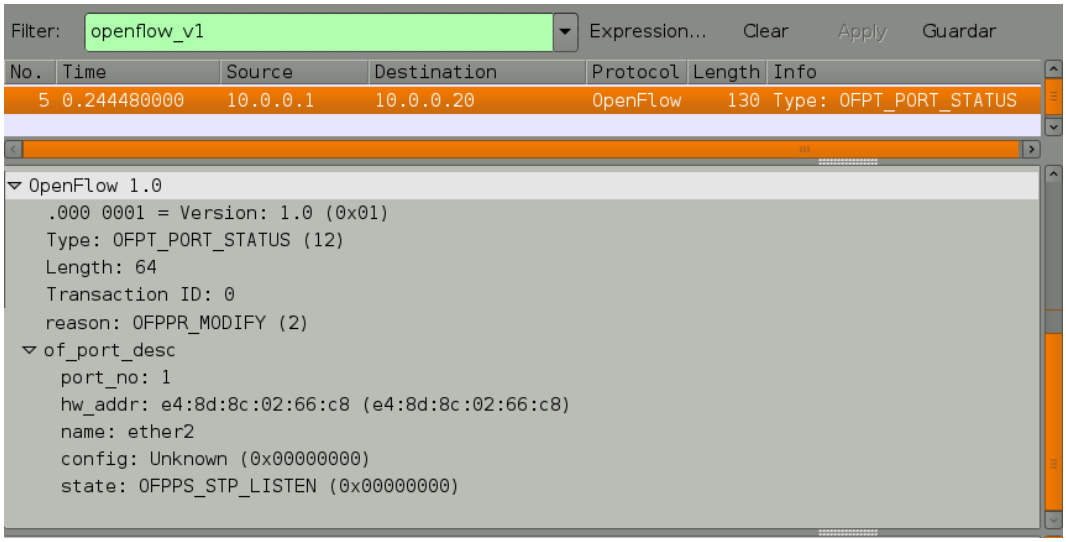

Figura 56: Mensaje OFPPS_STP_LISTEN.

En la Figura 56 se observa un mensaje OFTP_PORT_STATUS enviado desde el switch al controlador. En este caso, el mensaje corresponde a la conexión física del Servidor de Autenticación en el puerto Openflow número 1 con nombre ether2. Es importate descatar el campo *state*, el

cual contiene el código OFPPS_STP_LISTEN indicando que el puerto se encuentra activo.

A diferencia de lo descrito en el párrafo anterior, la Figura 57 corresponde a la desconexión del Sevidor de autenticación. Se puede observar el envío de un paquete desde el switch al controlador con el mensaje OFTP_PORT_STATUS en donde el campo *state* incluye el código OFPPS_LINK_DOWN, indicando que el puerto se encuentra desconectado.

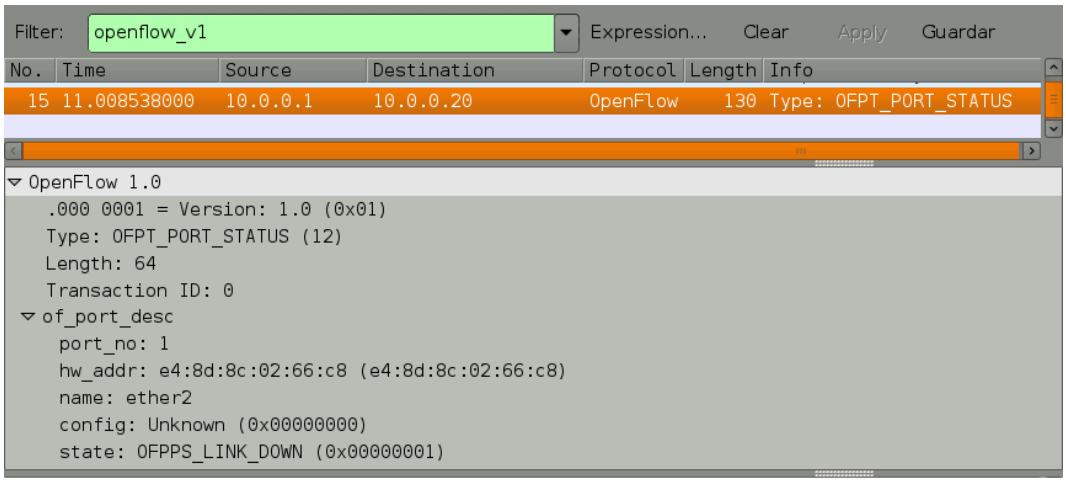

Figura 57: OFPPS_LINK_DOWN.

MENSAJES ECHO REQUEST Y ECHO REPLY

Los mensajes OFPT_ECHO_REQUEST y OFPT_ECHO_REPLY son enviados del switch al controlador o viceversa. Un mensaje OFPT_ECHO_REQUEST consiste en un encabezado OpenFlow más un campo de datos de longitud arbitraria. El campo de datos puede ser:

- Una marca de tiempo, con el objetivo de verificar la latencia
- De diferentes tamaños, para medir el ancho de banda.
- De tamaño cero, para verificar sólo la conectividad entre el switch y el controlador.

La Figura 58 muestra un ejemplo de una solicitud enviada del switch al controlador.

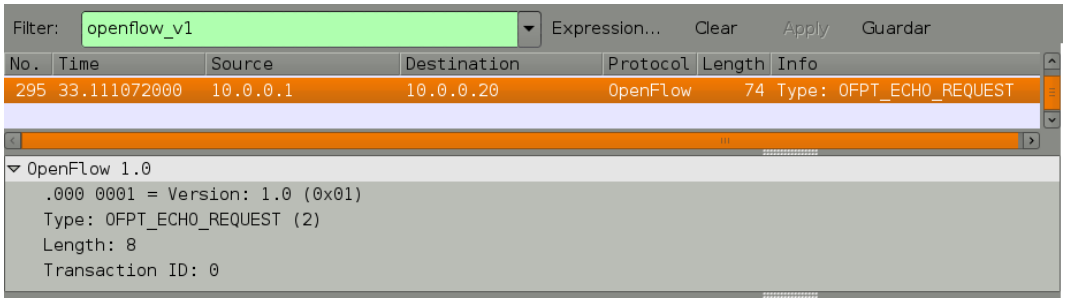

Figura 58: OFPT_ECHO_REQUEST.

Un mensaje de respuesta OFPT_ECHO_REPLY consiste en un encabezado de OpenFlow más el campo de datos no modificado de un mensaje de solicitud OFPT_ECHO_REQUEST. En la Figura 59 se puede observar un mensaje OFPT_ECHO_REPLY enviado por el controlador al switch en respuesta a la solicitud realizada por éste.

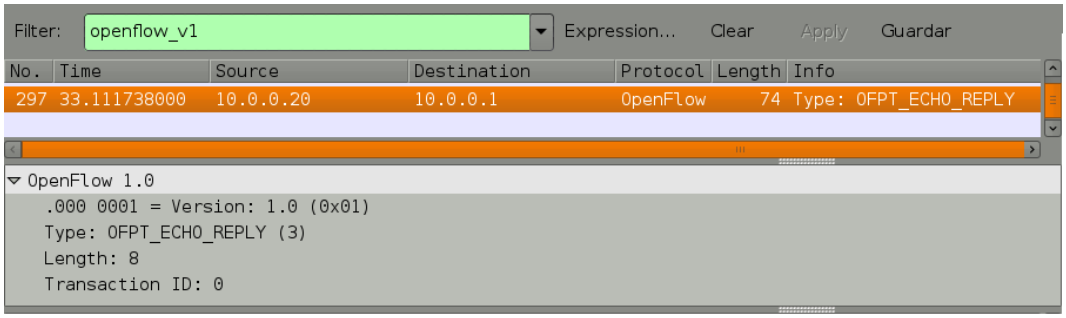

Figura 59: Mensaje OFPT_ECHO_REPLY.