ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UNA APLICACIÓN PARA LA IMPLEMENTACIÓN DE CALIDAD DE SERVICIO POR PRIORIZACIÓN DE TRÁFICO SOBRE UNA RED DEFINIDA POR SOFTWARE (SDN)

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

MOSCOSO CLERQUE ESTEBAN MAURICIO
estebanmoscoso09@gmail.com

DIRECTOR: ING. MEJÍA NAVARRETE DAVID M.Sc.
david.mejia@epn.edu.ec

Quito, Marzo 2016

DECLARACIÓN

Yo, Esteban Mauricio Moscoso Clerque, declaro bajo juramente que el trabajo aquí descrito es de mi autoría; que no ha sido presentado previamente para ningún grado o calificación profesional; y que he consultado las referencias bibliográficas que se incluyen en este documento.

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

_____

Esteban Mauricio Moscoso Clerque

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Esteban Moscoso Clerque, bajo mi supervisión.

_____

Ing. David Mejía Navarrete M.Sc.

DIRECTOR DEL PROYECTO

AGRADECIMIENTOS

A mis padres por su inmenso amor y apoyo incondicional en todo momento, por su esfuerzo y sacrificio que me permite hoy culminar mis estudios, por su guía, consejo y ejemplo que fueron fundamentales para tomar las mejores decisiones, por su incansable aliento que siempre estuvo a mi lado.

A mis hermanos que permanentemente están junto a mí como un pilar en mi vida, por su cariño y fortaleza.

A mis amigos y a todos quienes con una palabra de apoyo siempre están presentes.

A la Escuela Politécnica Nacional y sus docentes por el conocimiento que me fue entregado, gracias.

DEDICATORIA

A Jorge y Gabriela Moscoso Clerque, jamás se rindan,

todo es posible en la medida de que

tú creas que es posible.

Esteban Mauricio Moscoso Clerque

CONTENIDO

DECLARACIÓN ..... **I**

CERTIFICACIÓN ..... **II**

AGRADECIMIENTOS ..... **III**

DEDICATORIA ..... **IV**

CONTENIDO ..... **V**

ÍNDICE DE FIGURAS ..... **VIII**

ÍNDICE DE TABLAS ..... **XII**

PRESENTACIÓN ..... **XIII**

RESUMEN ..... **XV**

CAPÍTULO I **1**

FUNDAMENTOS TEÓRICOS **1**

1.1 INTRODUCCIÓN 1

1.2 REDES DEFINIDAS POR SOFTWARE 1

1.2.1 COMPONENTES DE UNA SDN 3

1.2.1.1 Plano de Datos 3

1.2.1.2 Plano de Control 4

1.2.1.2.1 Servidor Controlador 4

1.2.1.3 Plano de Aplicación 4

1.2.2 MODELOS SDN 5

1.2.2.1 Modelo Centralizado 5

1.2.2.2 Modelo Distribuido 6

1.2.2.3 Modelo Híbrido 7

1.3 PROTOCOLO OPENFLOW 9

1.3.1 Características De OpenFlow 10

1.3.2 Versiones de OpenFlow [31] 12

1.3.2.1 OpenFlow 1.0 (diciembre 2009) 12

1.3.2.2 OpenFlow 1.1 (febrero 2011) 13

1.3.2.3 OpenFlow 1.2 (diciembre 2011) 13

1.3.2.4 OpenFlow 1.3 (junio 2012) 14
1.3.2.5 OpenFlow 1.4 (octubre 2013) 14
1.3.3 Proveedores de acuerdo a la versión del protocolo OpenFlow 14
1.3.4 Switch OpenFlow 15
1.3.4.1 Tabla de flujo 15
1.3.4.2 Tabla de grupo 16
1.3.4.3 Canal OpenFlow 16
1.4 CONTROLADOR FLOODLIGHT 16
1.4.1 Arquitectura de Floodlight 17
1.4.1.1 Módulos de Control de Floodlight 18
1.4.1.2 Módulos de Aplicación 19
1.5 OPEN VSWITCH 19
1.6 VIRTUALBOX 20
1.7 ELASTIX 20

CAPÍTULO II 22

DISEÑO E IMPLEMENTACIÓN 22

2.1 INTRODUCCIÓN 22
2.2 DISEÑO E IMPLEMENTACIÓN DE LA SDN 22
2.2.1 Plataforma de Hardware y Software 23
2.2.2 Direccionamiento 24
2.2.3 Configuración de componentes de la red 25
2.2.3.1 Configuración de interfaces de red 25
2.2.3.2 Configuraciones en Open vSwitch 26
2.2.3.2.1 Enlace OVS-Floodlight 26
2.2.3.3 Configuración de VirtualBox 28
2.2.3.4 Configuración de Elastix 29
2.3 DESARROLLO DEL MÓDULO PARA BRINDAR CALIDAD DE SERVICIO 31
2.3.1 DESARROLLO DE LA APLICACIÓN 31
2.3.1.1 Diagrama de flujo 32
2.3.1.2 Diagrama de Clases 34
2.3.1.3 Descripción del código 36

CAPÍTULO III 52

PRUEBAS Y VALIDACIONES 52

3.1 INTRODUCCIÓN 52
3.2 CONECTIVIDAD DE LA RED 52
3.3 VERIFICACIÓN DE OPEN vSWITCH 54
3.4 ARCHIVO DE CONFIGURACIÓN 55
3.5 VALIDACIÓN DEL CONTROLADOR 56
3.5.1 INTERCAMBIO DE MENSAJES OPENFLOW 59
3.6 LLAMADAS TELEFÓNICAS 60
3.6.1 VERIFICACIÓN DEL INTERCAMBIO SIP Y RTP/RTCp 62
3.7 VALIDACIÓN DE ENCOLAMIENTO 65
3.7.1 PRUEBAS DE SATURACIÓN 70
3.8 PRESUPUESTO REFERENCIAL DEL PROTOTIPO 74

CAPÍTULO IV 76

CONCLUSIONES Y RECOMENDACIONES 76

4.1 CONCLUSIONES 76
4.2 RECOMENDACIONES 77

REFERENCIAS BIBLIOGRÁFICAS 81

ANEXOS 90

ÍNDICE DE FIGURAS

CAPÍTULO I

Figura 1.1 Arquitectura de una SDN 2
Figura 1.2 Arquitectura de una SDN con elementos de administración 3
Figura 1.3 Modelo Centralizado 5
Figura 1. 4 Modelo Distribuido..... 7
Figura 1.5 Modelo Híbrido..... 8
Figura 1.6 Modelo *OpenFlow* 11
Figura 1.7 Componentes de OpenFlow..... 12
Figura 1.8 Soporte Multitabla 13
Figura 1.9 Componentes principales de un *switch* OpenFlow..... 15
Figura 1.10 Componentes principales de una entrada de flujo dentro de una tabla 15
Figura 1.11 Componentes principales de una entrada de grupo dentro de una tabla 16
Figura 1.12 Arquitectura de Floodlight 17

CAPÍTULO II

Figura 2.1 Diagrama de la red implementada 22
Figura 2.2 Diagrama lógico de la red 24
Figura 2.3 Creación de interfaces de red 25
Figura 2.4 Interfaces para enlace entre *switches*..... 25
Figura 2.5 Creación de un *switch* virtual 26
Figura 2.6 Configuraciones para el *switch* br0 26
Figura 2.7 Controlador Floodlight en ejecución..... 27
Figura 2.8 Comando para el enlace *switch*-controlador 27
Figura 2.9 Configuración de Open vSwitch 27
Figura 2.10 Floodlight al momento del enlace con los *switches* virtuales 28
Figura 2.11 Configuración de la interfaz de red en Virtualbox..... 28
Figura 2.12 Configuración de interfaces de red..... 29
Figura 2.13 Configuración de la interfaz de red en Centos 29
Figura 2.14 Creación de una extensión telefónica 30

Figura 2.15 Registro de la extensión 30
Figura 2.16 Clave de la extensión 30
Figura 2.17 Diagrama de flujo 33
Figura 2.18 Diagrama de clases 34
Figura 2.19 Librerías de *Floodlight* y Java 37
Figura 2.20 Librerías de *OpenFlow* y slf4j 37
Figura 2.21 Declaración de la clase para QoS 38
Figura 2.22 Método getName 38
Figura 2.23 Métodos para orden de ejecución de módulos 39
Figura 2.24 Funciones getModuleServices y getServiceImpls 39
Figura 2.25 Método getModuleDependencies 40
Figura 2.26 Método init 40
Figura 2. 27 Lectura del archivo de configuración 41
Figura 2. 28 Categorización de parámetros de comparación 41
Figura 2.29 Método startUp 42
Figura 2.30 Método para recepción de mensajes 43
Figura 2.31 Validación capa 3 y 4 44
Figura 2. 32 Validaciones para RTP/RTCP 44
Figura 2.33 Método insertarFlujo 45
Figura 2.34 Obtención de *host* origen y destino 46
Figura 2.35 Creación del *match* 47
Figura 2.36 Creación de acciones 48
Figura 2.37 Creación y envío de la regla de flujo 49
Figura 2.38 Determinación de los puertos de entrada y salida del flujo 49
Figura 2.39 Creación de la regla de flujo1 50
Figura 2.40 Modificación de forwardingbase 51

CAPÍTULO III

Figura 3.1 Interfaces de red 53
Figura 3.2 Conectividad entre *hosts* 54
Figura 3.3 Conectividad con el servidor Elastix 54
Figura 3.4 Puertos de los *switches* virtuales 54
Figura 3.5 Versión de los *switches* virtuales 55
Figura 3.6 Procesos de Open vSwitch en ejecución 55

Figura 3.7 *Dashboard* de Floodlight 56
Figura 3.8 Topología original 57
Figura 3.9 Topología modificada 57
Figura 3.10 Sección *Switches* dentro de Floodlight 57
Figura 3.11 Regla de flujo insertada en el *switch* 58
Figura 3.12 Opción *Host* en la interfaz de usuario de Floodlight 58
Figura 3.13 Consola de *Floodlight* 58
Figura 3.14 Filtro para Openflow en Wireshark 59
Figura 3.15 Mensajes Openflow 59
Figura 3.16 Instalación de Linphone 60
Figura 3.17 Credenciales dentro de Linphone 60
Figura 3.18 Nuevo contacto en Linphone 61
Figura 3.19 Llamada en curso 61
Figura 3.20 Módulo de telefonía en Wireshark 62
Figura 3.21 Intercambio de tramas SIP 62
Figura 3.22 Trama de registro 63
Figura 3.23 Trama RTP 64
Figura 3.24 Paquete de finalización de llamada 64
Figura 3.25 Parámetros para envío de paquetes TCP 65
Figura 3.26 Resultados de mediciones de ancho de banda 66
Figura 3.27 Medición de ancho de banda de una llamada VoIP 67
Figura 3.28 Creación de colas en OVS 67
Figura 3.29 Verificación de colas en el *switch* 68
Figura 3.30 Resultados de ancho de banda 68
Figura 3.31 *Jitter* en cola 0 69
Figura 3.32 Paquetes ICMP en la cola 0 69
Figura 3.33 Paquetes en cola 0 70
Figura 3.34 Paquetes en cola 1 70
Figura 3.35 Configuraciones de ffserver 71
Figura 3.36 Comando para transmisión de video 71
Figura 3.37 Video *streaming* 72
Figura 3.38 *Jitter* con saturación 73
Figura 3.39 Ping con saturación 73
Figura 3.40 *Streaming* con saturación 73

Figura 3.41 Calidad de la llamada telefónica 74

ÍNDICE DE TABLAS

CAPÍTULO I

Tabla 1.1 Comparativa de los modelos SDN..... 9
Tabla 1.2 Proveedores OpenFlow..... 14

CAPÍTULO II

Tabla 2.1 Especificaciones del computador empleado 23
Tabla 2.2 Especificaciones de las máquinas virtuales creadas..... 23
Tabla 2.3 Direccionamiento de red..... 24

CAPÍTULO III

Tabla 3.1 Resumen de resultados..... 74
Tabla 3.2 Presupuesto referencial..... 75

PRESENTACIÓN

Las redes de datos, día a día, incrementan su tamaño y complejidad, demandan mayor rapidez y recursos, se requieren servicios más ágiles y eficientes que permitan aumentar la productividad y alcanzar competitividad dentro del mercado, es por ello que se buscan nuevas tecnologías que permitan cumplir con estos requerimientos.

Actualmente, las Redes Definidas por Software (SDN) se encuentran catalogadas como el futuro de las redes; según la IDC (*International Data Corporation*), el volumen de ingresos por productos SDN como infraestructura de red, aplicaciones, soluciones de plano de control y servicios profesionales asociados para empresas y proveedores de servicios, pasarán de los 360 millones de euros que generaron el 2013 a 3.700 millones en el 2016, lo que demuestra un crecimiento sostenido de esta tecnología.

En las SDN, gracias a la separación del plano de control del plano de datos, las empresas obtienen una capacidad de análisis del estado de la red, con la cual pueden potenciar su negocio y controlar que tráfico circula y cuáles son los elementos que están interactuando; esta tecnología es programable, centralizada y permite acceder a ciertas funciones inteligentes de forma ágil e implementa funcionalidades de red a una mayor velocidad que la ofrecida por una solución tradicional.

Actualmente, redes distintas como telefonía, video, imagen entre otros, se fusionan en una red IP, provocando el constante incremento de la información que circula en las redes de datos y poniendo en riesgo la eficiencia en la transmisión, es por ello que se maneja el término calidad de servicio.

La calidad de servicio (QoS) se define como la garantía de un valor límite mínimo o máximo de un parámetro determinado; dentro de las redes se aplica en función del tiempo que demora la entrega de un paquete a través de la red sin importar el estado o saturación que presente, el porcentaje de información que se pierde en la transmisión, entre otros; estos conceptos son de gran valía debido al contenido que pueda transportar dicho paquete de datos y es una de las claves del éxito para el despliegue de las redes convergentes, pensadas como

plataformas multiservicios y soporte para distintos modelos de productos que sumen valor a la red.

Debido a los argumentos expuestos, se plantea el presente Proyecto de Titulación denominado “DESARROLLO DE UNA APLICACIÓN PARA IMPLEMENTACIÓN DE CALIDAD DE SERVICIO POR PRIORIZACIÓN DE TRÁFICO SOBRE UNA RED DEFINIDA POR SOFTWARE (SDN)”. En él, se busca integrar los beneficios de las SDN, sin dejar de lado la calidad en la transmisión de datos considerados primordiales. Se propone la implementación de un prototipo basado en software utilizando *switches* virtuales creados con la herramienta Open vSwitch, el controlador Floodlight y máquinas virtuales generadas en el hipervisor VirtualBox que actúan como *hosts.*

De acuerdo a la lógica con la que trabaja una Red Defina por Software, cuando a un *switch* le llega un paquete y no existe una regla de reenvío establecida, el dispositivo de red realiza una consulta al controlador para que este le indique como manejar ese paquete.

En el Proyecto se aprovecha esta característica para lograr una transmisión con calidad de servicio, así, se presenta el desarrollo de un módulo dentro del controlador que permita analizar los paquetes de datos y de acuerdo a este análisis enviar las reglas correspondientes a los *switches* y lograr una priorización en el tráfico, brindando una solución alternativa a las redes tradicionales.

RESUMEN

El presente Proyecto de Titulación se encuentra dividido en cuatro capítulos. En el primero se realiza una revisión del fundamento teórico; el segundo capítulo comprende la implementación de una SDN, las configuraciones de la misma y el desarrollo de la aplicación que brinda calidad de servicio; en el tercero se procede con las pruebas de funcionamiento y validaciones de la solución y finalmente en el cuarto, se presentan las conclusiones y recomendaciones del Proyecto.

En primer lugar, se analiza el funcionamiento de una Red Definida por Software, así como su estructura y componentes, como base para el desarrollo del proyecto; una SDN consta de un plano de control manejado por un servidor controlador que permite definir las reglas y políticas de transmisión, y de un plano de datos constituido por los dispositivos de red que reciben las instrucciones del controlador y las ejecutan.

El controlador escogido para el desarrollo del proyecto es Floodlight, el cual está escrito en Java y es de código abierto; adicionalmente, se estudia brevemente las características de OpenFlow, que es un protocolo que permite la comunicación entre el controlador y los *switches*.

Una vez realizado el estudio general de los conceptos y funciones de las Redes Definidas por Software, se realiza el diseño e implementación de una red SDN, la misma que consta de dos *switches*, el servidor controlador para el manejo del tráfico y cinco computadores adicionales que son utilizados para el intercambio de información, permitiendo probar la priorización realizada por la aplicación que brinda calidad de servicio. Luego, se procede con el desarrollo de la aplicación que permite brindar la priorización deseada.

La aplicación se desarrolla en el lenguaje de programación Java, corre sobre el controlador Floodlight y determina si el tráfico debe ser priorizado o no, para ello, realiza la captura, el procesamiento de los paquetes intercambiados dentro de la red y analiza las cabeceras TCP e IP, así como el puerto origen y destino, y de acuerdo al tipo de trama encontrada envía reglas al *switch*, para que este conozca como procesar estos paquetes. Finalmente, se realizan pruebas de funcionamiento utilizando las máquinas virtuales adicionales; se procede con la

transmisión de voz a través de una llamada telefónica, una vez establecida la misma, se emplea una o varias aplicaciones que permitan inyectar múltiples paquetes a la red, como por ejemplo video *streaming*, de esta manera, el enlace entre los *switches* se satura, sin embargo la aplicación que brinda calidad de servicio, permite que la saturación no afecte a la llamada en curso, demostrando así la validez del Proyecto; se realiza capturas de tráfico y mediciones de latencia y *jitter* para validar la efectividad de la solución.

Se incluye un presupuesto referencial del costo de la implementación del Proyecto de Titulación, así como las conclusiones y recomendaciones respectivas.

En la parte final se presentan los Anexos correspondientes a los tutoriales elaborados e información adicional recopilada durante el desarrollo del Proyecto.

Se adjunta el CD correspondiente donde se encuentra el código de la aplicación que brinda calidad de servicio, anexos y los *scripts* utilizados para las diferentes configuraciones de la red.

CAPÍTULO I
FUNDAMENTOS TEÓRICOS

1.1 INTRODUCCIÓN

El presente Capítulo es de carácter teórico, en él se detallarán los conceptos fundamentales sobre Redes Definidas por Software, se describirán sus características y funcionalidades, su arquitectura y componentes como por ejemplo el servidor controlador, que, para el proyecto será Floodlight; de igual manera, se estudiará el protocolo OpenFlow que permite la comunicación entre el controlador y los dispositivos de red; estos conceptos serán utilizados posteriormente en la implementación de calidad de servicio por priorización de tráfico para una SDN.

1.2 REDES DEFINIDAS POR SOFTWARE

Los sistemas de comunicación se encuentran en constante evolución, cada día se generan nuevos conceptos y formas de establecer las comunicaciones; las redes, incrementan su tamaño y complejidad, se demanda mayor rapidez y recursos, se requieren servicios más ágiles y eficientes que permitan aumentar la productividad y alcanzar competitividad dentro del mercado, es en ese contexto que emerge una nueva tecnología, las Redes Definidas por Software (SDN).

El principal objetivo de las SDN es centralizar la administración de la red y mediante software controlar la conectividad y el flujo de datos, con la posibilidad de incluir reglas específicas dentro de estos.

Una SDN plantea una arquitectura de red que se caracteriza por separar el plano de datos del plano de control, siendo este último programable; esta cualidad permite determinar las características de los dispositivos de red mediante la utilización de aplicaciones y servicios, dando paso al manejo de la red como una entidad lógica o virtual [9].

La Figura 1.1 muestra la arquitectura básica de una Red Definida por Software, la cual consta de 3 capas: aplicación, control y datos, así como las interfaces entre ellas. En la parte baja se encuentra la capa de infraestructura que comprende los dispositivos de red, estos prestan sus características y

funcionalidades a la capa de control, que, a través de la interfaz de control de plano de datos, actúa como plano regulador. El plano de datos debe contener las características de control y gestión necesarias para ejecutar las tareas emitidas por el controlador [2].

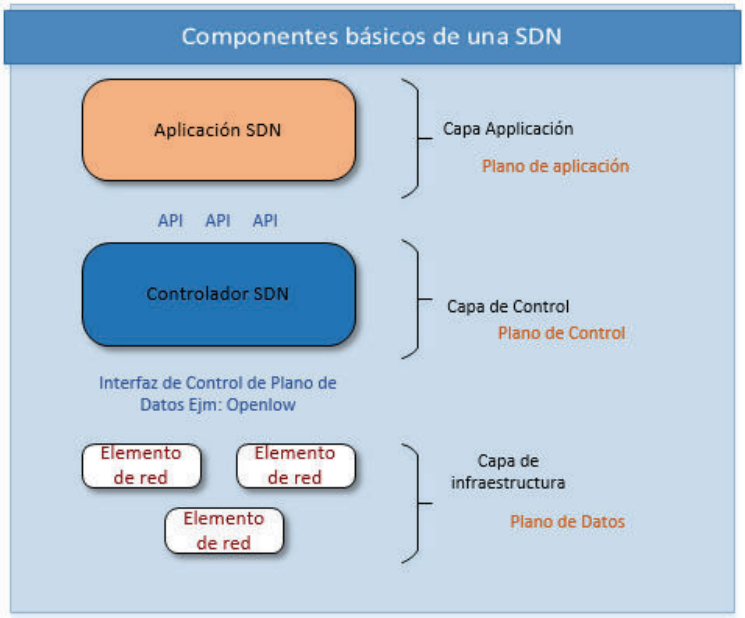


Figura 1.1 Arquitectura de una SDN

En la parte media, el controlador SDN traduce los requerimientos del plano de aplicación y al mismo tiempo ejerce el control de bajo nivel sobre los dispositivos de la red y proporciona información relevante a las aplicaciones. Un controlador SDN responde a las demandas de las aplicaciones de acuerdo a los recursos de red que posee y establece las políticas adecuadas para satisfacer las necesidades requeridas. En la parte superior se encuentra la capa de aplicación, la misma que comunica sus requerimientos de la red hacia el plano de control a través de las interfaces (API)[1].

Posteriormente se profundizará en las características de cada una de las capas de la estructura lógica de las SDN. Muchas de las funciones de administración son realizadas por la interfaz entre el plano de control y el de aplicación ACPI (*Aplication Controller Plane Interface*), sin embargo, ciertas funciones deben ser realizadas en cada una de las capas. En el plano de datos, por ejemplo, se

[1] **API** (*Application Programming Interface*): Formato de mensajes utilizado para comunicar una aplicación con el sistema operativo u otro programa de control [4].

requiere al menos la gestión de la configuración inicial de los elementos de la red. En el plano de control, las funciones de administración definen políticas para establecer el alcance del control que se asignará a la aplicación SDN y para monitoreo de la eficiencia del sistema. Finalmente, en el plano de aplicación, la gestión configura los contratos y acuerdos de nivel de servicio (SLA). En todos los planos, la gestión establece las políticas de seguridad que permiten la comunicación de forma fiable [2]. En la Figura 1.2 se puede apreciar la arquitectura de una SDN con la inclusión de las funciones de administración.

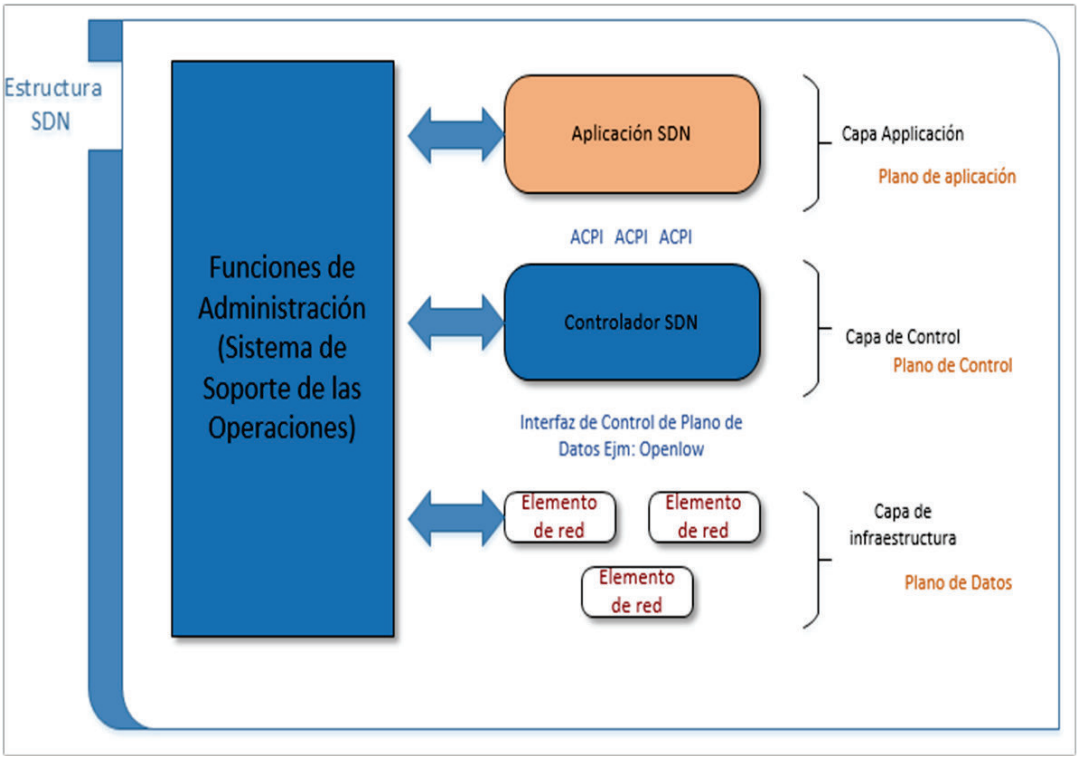


Figura 1.2 Arquitectura de una SDN con elementos de administración

1.2.1 COMPONENTES DE UNA SDN

En esta sección, se describen los componentes de una SDN.

1.2.1.1 Plano de Datos

El plano de datos comprende los elementos o dispositivos de red, maneja los paquetes entrantes y los procesa de acuerdo a reglas establecidas en tablas de flujo. Este plano incorpora los recursos para manipular directamente el tráfico, junto con los medios de apoyo necesarios para garantizar la correcta

implementación de virtualización[2], la conectividad, la seguridad, la disponibilidad y la calidad del servicio. La información entra y sale del plano de datos por puertos físicos y/o virtuales, es dirigida a las funciones de procesamiento o análisis de datos, las mismas que son ejecutadas por el servidor controlador o en su defecto por un mecanismo externo que trabaje en conjunto con este. El plano de datos no toma decisiones de reenvío por sí solo, realiza el proceso de consulta al controlador para realizar esta tarea sin embargo, el controlador puede almacenar órdenes preestablecidas para que el procesamiento sea eficiente, en casos, como por ejemplo, fallos en la red, el plano de datos utilizará dichas órdenes y actuará automáticamente.

1.2.1.2 Plano de Control

A través de las diferentes capas de la SDN se ejercen diversos grados de control, sin embargo, dentro del plano de control, se ejecuta el principal. El plano de control está compuesto por el servidor controlador, el mismo que maneja uno o varios elementos de red y coordina el intercambio de información, a este proceso se denomina "*orchestration*" (orquestación).

1.2.1.2.1 *Servidor Controlador*

El Controlador SDN como su nombre lo indica es implementado para controlar el plano de datos y recibir de la capa aplicación los elementos necesarios para determinar el tipo de control que se va a ejercer. Existen una gran variedad de controladores desarrollados, como Floodlight, OpenDaylight, NOX, entre otros. El controlador contiene módulos que manejan las diferentes funciones necesarias para el correcto funcionamiento de la red, como por ejemplo la determinación de la topología de la red o el cálculo de la mejor ruta de reenvío de paquetes.

1.2.1.3 Plano de Aplicación

Las características de una SDN permiten a las aplicaciones especificar los recursos y comportamientos que requieren de la red, como un acuerdo de reglas y políticas. Una aplicación puede invocar servicios externos y dirigir (orquestar)

[2] **Virtualización:** En una arquitectura SDN, se considera la virtualización como la designación de un conjunto de recursos abstractos para una aplicación o cliente específico.

cualquier número de controladores SDN para lograr sus objetivos. La aplicación SDN debe tener por lo menos el conocimiento a priori de su entorno para poder actuar de la manera esperada. La actividad a través de la interfaz entre el controlador y el plano aplicación (A-CPI), incluye consultas o notificaciones sobre el estado de la red virtual, de igual manera se intercambian comandos para modificar dichos estados. Por ejemplo se puede crear o modificar conectividad dentro de la red o procesar el tráfico incluyendo un ancho de banda específico o implementando calidad de servicio (QoS[3]). La A-CPI[4] puede ser utilizada también por funciones adicionales, por ejemplo como punto de acceso para configurar un servicio o como entrada para las funciones de control de la red virtualizada.

1.2.2 MODELOS SDN

Existen diferentes modelos para implementar una SDN, cada uno tiene sus ventajas y desventajas.

1.2.2.1 Modelo Centralizado

En la Figura 1.3 se muestra la estructura del modelo centralizado.

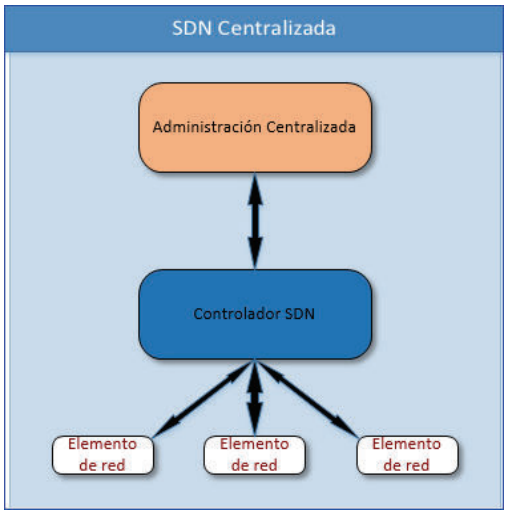


Figura 1.3 Modelo Centralizado

[3] **QoS** (Calidad de servicio): Efecto global de las características de servicio que determinan el grado de satisfacción de un usuario del servicio [17].

[4] **A-CPI** (*Advanced Configuration and Power Interface*): Especificación que proporciona un estándar que los sistemas operativos pueden utilizar para el descubrimiento de hardware, configuración, administración de energía y monitoreo.

Se presenta un solo gestor (administrador) y un solo controlador, que manejan el intercambio de información del plano de datos mediante las definiciones de rutas y flujos. El enfoque centralizado simplifica la administración de flujos complejos que están relacionados con aplicaciones específicas, permite que las colas de prioridad se ajusten alrededor de un solo tipo de información, por ejemplo, datos de voz a través del tráfico de navegación web, independientemente de los nodos de la red [14].

La desventaja que tiene este modelo es la escalabilidad al momento que se presenta un crecimiento exponencial de la red, especialmente en entornos basados en la web o en la nube, el control centralizado requiere poderosos sistemas, los cuales son costosos y utilizan CPU más potentes, mayor almacenamiento y una infraestructura más resistente con menos tolerancia a fallos, lo que convierte al enfoque centralizado en un modelo costoso y complejo.

1.2.2.2 Modelo Distribuido

En este modelo, cada elemento de la red tiene un pequeño controlador también denominado agente, el mismo que cumple determinadas tareas eliminando la carga centralizada, su estructura se visualiza en la Figura 1.4.

Debido a esta característica, el procesamiento y manejo de solicitudes se realiza rápidamente ya que se evita que cada petición sea consultada al controlador central; esta técnica se aplica también en redes inalámbricas, donde el *Access Point* responde a los paquetes que requieren una contestación rápida y los que necesitan mayor procesamiento como los de autenticación, son enviados al control central de la WLAN[5] para su gestión.

Un *switch* SDN, por ejemplo, podría decidir por sí mismo que acción tomar si se produce la caída de un enlace, dentro de la tabla de flujo se encontraría una ruta alternativa por la cual se enviaría la información de forma automática cuando el fallo se produzca, evitando así la consulta al controlador central, solamente se notificaría del error y se continuaría con el reenvío de paquetes [14].

La principal ventaja que poseen las SDN distribuidas es que los elementos de red tienen un mejor rendimiento debido a la presencia del agente por lo que el procesamiento de datos es más rápido, su desventaja se presenta en la

[5] **WLAN** (*Wireless Local Area Network*): Red que conecta dispositivos en un área relativamente pequeña de forma inalámbrica.

flexibilidad ya que las características y funcionalidades del agente varían dependiendo del fabricante.

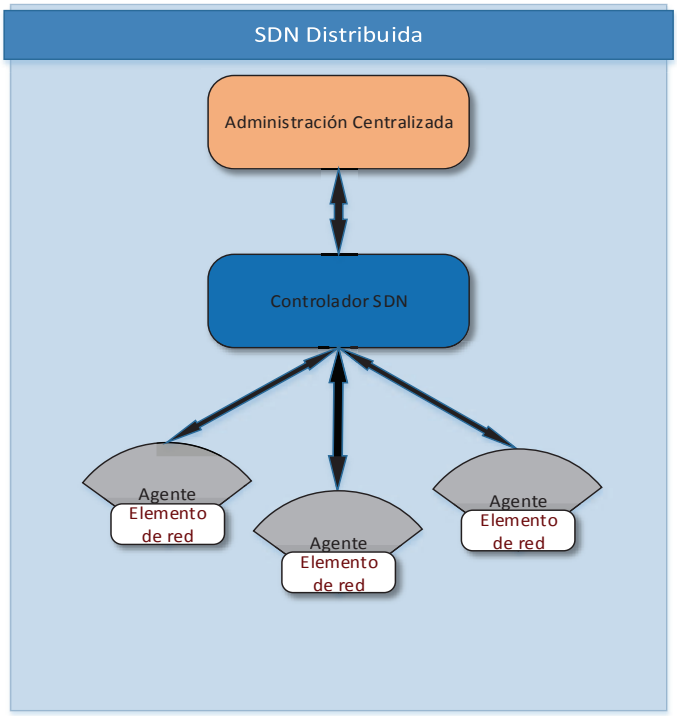


Figura 1. 4 Modelo Distribuido

1.2.2.3 Modelo Híbrido

Cada uno de los modelos presentados posee un valor, es por ello que se crea el modelo híbrido juntando las características de los modelos centralizado y distribuido. Se presenta un gestor centralizado con varios controladores y cada uno de ellos maneja varios elementos de red, aprovechando así el control de datos en los flujos como en el modelo centralizado e incluyendo la escalabilidad y flexibilidad del modelo distribuido. En la Figura 1.5 se muestra la estructura de una SDN híbrida.

Definir políticas es clave para cualquier red, y cumplir con los requisitos de una aplicación a través de una red de gran tamaño, es muy complejo. En una SDN híbrida, el enfoque ideal es tener políticas estrictas respeto a ciertas aplicaciones para un mayor control y políticas flexibles para una mayor agilidad y procesamiento, entre otras.

El enfoque híbrido permite una definición de políticas manejable, se puede empezar a nivel local, y luego a medida que el tamaño de la red se incremente,

aplicar esas políticas a los nuevos componentes de red. Por ejemplo, el tráfico de voz puede tener una política de priorización general, sin embargo, si desea que el tráfico de otra aplicación tome prioridad como el tráfico de video, entonces se puede definir la política de priorización por una ruta específica en la red para su transmisión y no abarcar toda la red en una configuración determinada. Respecto a la seguridad, se puede aplicar el mismo enfoque, es posible que se desee requisitos de seguridad globales para ser manejados de manera distribuida, pero también se pueden aplicar políticas locales de seguridad específicas para una gestión más centralizada.

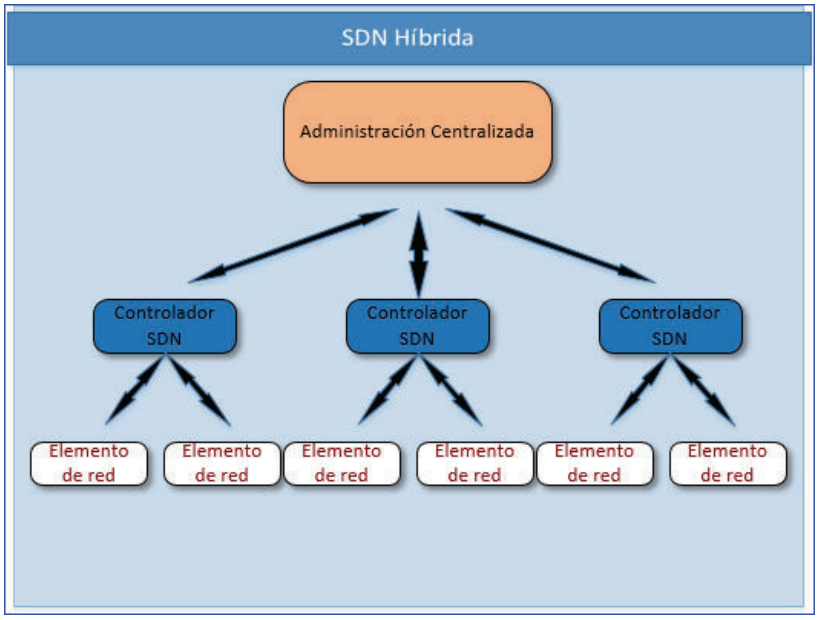


Figura 1.5 Modelo Híbrido

Es mucho más probable que la automatización de la red se realice a nivel del controlador de forma distribuida, mientras que las políticas globales y la configuración de seguridad se producirán a nivel de gestión centralizada.

Otra forma de configuración permitiría distribuir la carga de una mejor manera, un controlador se encarga de la configuración de la red, un segundo gestiona el centro de datos y un tercero de la red inalámbrica, por ejemplo; para ello se requiere el sistema de gestión centralizado que permita la comunicación entre los tres controladores. El flujo de información puede viajar de un controlador para el sistema de gestión y enviarlo después a otro controlador; por lo tanto, los cambios que se hicieron en una parte de la red tienen impacto en otra.

En la Tabla 1.1 se realiza una comparación entre los tres modelos de SDN descritos, incluyendo sus ventajas (azul) y desventajas (rojo):

SDN	Precio	Escalabilidad	Rendimiento	Flexibilidad	Dependencia
Centralizada	Depende de la capacidad	Baja	Bajo	Alto	No depende del fabricante
Distribuida	Hardware costoso	Alta	Alto	Bajo	Depende del fabricante
Hibrida	Alto	Alta	Medio	Alto	No depende del fabricante

Tabla 1.1 Comparativa de los modelos SDN

De acuerdo a las características descritas y tomando en cuenta el objetivo del Proyecto es crear una aplicación en un solo controlador que maneje las reglas de flujo y que se utilizará virtualización en los elementos de red, se utilizará el modelo centralizado.

1.3 PROTOCOLO OPENFLOW

OpenFlow es el primer protocolo de comunicación definido entre las capas de control y reenvío de información o infraestructura en una red SDN. Los orígenes de este protocolo se remontan al año 2006 cuando Martin Casado, un estudiante de la Universidad de Stanford en Silicon Valley, California, desarrolló “Ethane”, una arquitectura de red lógicamente centralizada para la gestión de las políticas de seguridad de las redes empresariales.

Esta idea condujo a lo que actualmente se conoce como OpenFlow, gracias a los trabajos realizados en conjunto por los equipos de la Universidad de Stanford y la Universidad de California. Aunque era una tecnología que daba sus primeros pasos en el mercado, obtuvo una gran acogida en Silicon Valley, las empresas Nicira[6] y Big Switch Networks Inc., invirtieron grandes cantidades de dinero para sacar a flote su producto.

La segunda versión del protocolo denominada 1.1 fue lanzada en febrero de 2011, su tercera versión 1.2 se lanzó en diciembre del mismo año con el visto bueno de la ONF (*Open Networking Foundation*). Esta organización fue creada en 2011 con el objetivo de estandarizar tecnologías emergentes[7], tomando el

[6] **NICIRA**: Compañía enfocada en SDN y redes virtualizadas, creada por Martin Casado en 2007. Fue adquirida por VMware en agosto de 2012 por 1.26 billones de dólares [29].

[7] **Tecnología emergente**: Las tecnologías emergentes son definidas como innovaciones científicas que pueden crear una nueva industria o transformar una existente [30].

software como componente principal en el entorno de las redes y administración de centros de datos, las principales empresas fundadoras son Google, Facebook y Microsoft, actualmente tiene más de 150 miembros. La promoción de las SDN y OpenFlow fue uno de los principales beneficios de la ONF, de acuerdo a Stu Bailey, fundador y CTO de Infoblox [27].

1.3.1 CARACTERÍSTICAS DE OPENFLOW

Es el primer protocolo de comunicación definido entre la capa de control y la capa de datos, permite el acceso directo y la manipulación de los dispositivos de red como *switches* y *routers*, tanto físicos como virtuales, estos últimos basados en un hipervisor[8]. Debido a la ausencia de una interfaz abierta que permita mover el control de la red fuera del entorno lógico, se ha convertido en el estándar al momento de la implementación de las SDN.

En la Figura 1.6 se muestran los componentes principales del modelo OpenFlow, los mismos que se han convertido en parte de la definición de una SDN, la misma que se resume en las siguientes características:

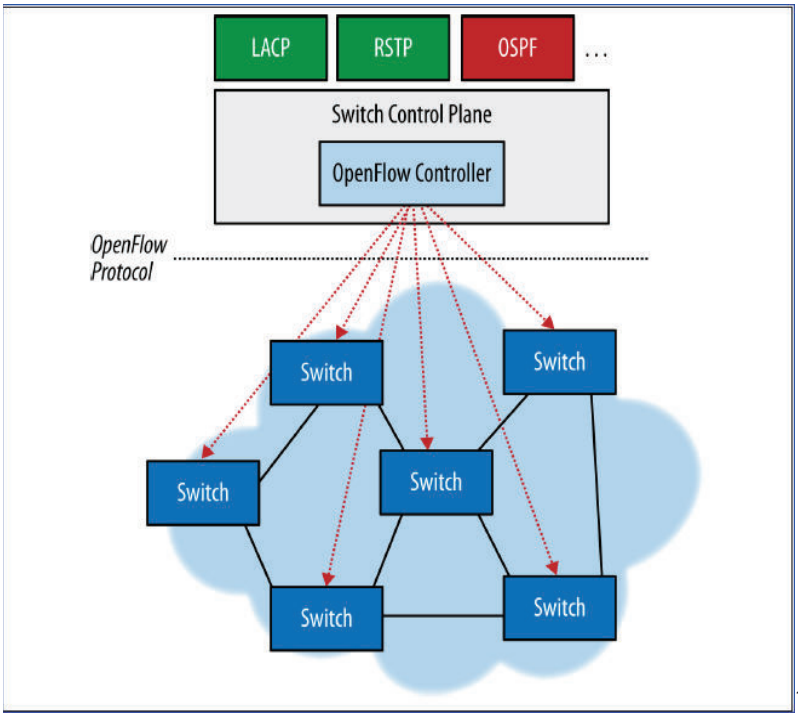
- Separación del plano de datos y de control.
- El uso de un protocolo estandarizado entre el controlador y un agente en la red.
- Proporcionar capacidad de programación de la red desde una visión centralizada a través de una moderna y extensible API.

El protocolo se implementa en ambos lados de la interfaz entre los dispositivos de infraestructura de red y el software de control de la SDN. Utiliza el concepto de flujos para identificar el tráfico basándose en reglas predefinidas por el software del controlador SDN, estas pueden ser estáticas o dinámicas. Adicionalmente, OpenFlow permite definir cómo el tráfico debe fluir a través de los dispositivos de red basados en parámetros tales como los patrones de uso, aplicaciones y recursos de la nube, proporciona un control específico, lo que permite que la red responda a los cambios en tiempo real a nivel de aplicación,

[8] **Hipervisor**: Es un programa que permite a múltiples sistemas operativos compartir un solo procesador [37].

en usuarios y a nivel de sesión con facilidad. OpenFlow está compuesto por una API y por un conjunto de protocolos, los mismos que se dividen en dos partes:

- Un protocolo de conexión (actualmente en la versión 1.3) para establecer una sesión de control, define la estructura del mensaje para intercambiar las modificaciones de los flujos y obtener estadísticas, y con esta información definir la estructura fundamental del *switch* (puertos y tablas). La versión 1.1 añadió la capacidad de soportar múltiples tablas y la ejecución de acciones almacenadas.
- Un protocolo de configuración y administración (actualmente en la versión 1.4) basado en NETCONF[9], para asignar puertos de *switch* físicos a un controlador en específico, define la disponibilidad (activo / en espera) y los comportamientos en el controlador, cuando se produce un fallo en la conexión.



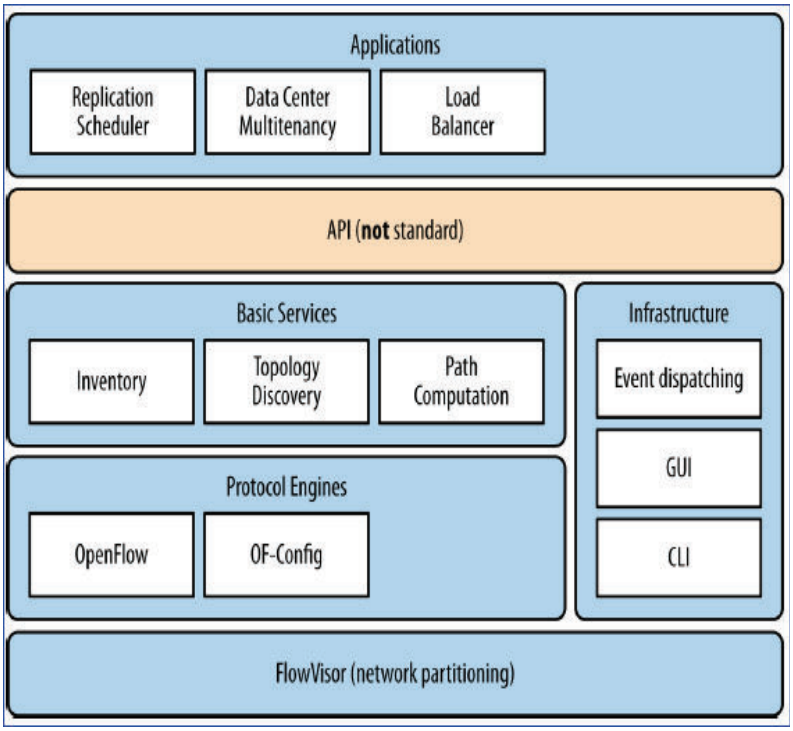
[10]

Fuente: [38]

Figura 1.6 Modelo *OpenFlow*

[9] **NETCONF**: Es un protocolo definido por la IETF (*Internet Engineering Task Force*) para instalar, modificar y eliminar configuraciones en dispositivos de red.

[10] **LACP** (*Link Aggregation Control Protocol*), **RTSP** (*Real Time Streaming Protocol*), **OSPF** (*Open Shortest Path First*): Protocolos de red.



Fuente: [38]

Figura 1.7 Componentes de OpenFlow

Una SDN basada en OpenFlow, puede ser implementada sobre una red tanto física como virtual, los dispositivos de red, pueden soportar las reglas de reenvío del protocolo de igual manera que lo hacen con las configuraciones tradicionales, lo que facilita la migración a esta nueva tecnología incluso si dentro de la red se tienen equipos de diferentes proveedores.

1.3.2 VERSIONES DE OPENFLOW [31]

Desde que fue desarrollado el protocolo se han realizado cambios y mejoras sobre el mismo, se tiene un total de 5 versiones oficiales que son las siguientes:

1.3.2.1 OpenFlow 1.0 (diciembre 2009)

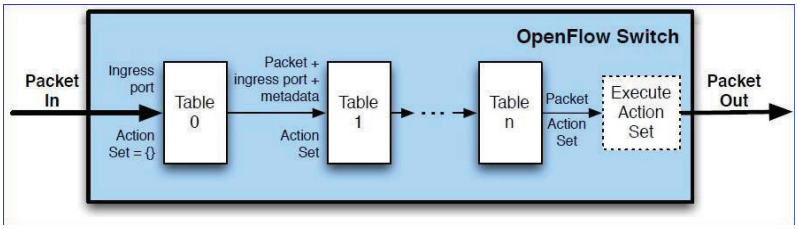
- El controlador se comunica con el switch mediante el protocolo OpenFlow utilizando un canal seguro TLS11.
- La tabla de flujo consta de: campos para emparejar con las cabeceras de los paquetes, contadores y un set de instrucciones.

[11] **TLS** (*Transport Layer Security*): Es un certificado que permite y garantiza el intercambio de datos en un entorno seguro y privado entre dos entes [43].

- Existen tres tipos de mensajes: mensajes del controlador al *switch*, mensajes asíncronos del *switch* al controlador y mensajes para comunicación simétrica entre el controlador y el *switch*.

1.3.2.2 OpenFlow 1.1 (febrero 2011)

- La mayoría de los *switches* tienen diferentes tablas para manejar diferentes funcionalidades como el procesamiento de capa dos, capa 3, calidad de servicio, entre otros. OpenFlow 1.1 introduce el soporte multitabla y la forma para vincular los resultados de las tablas individuales. En la Figura 1.8 se muestra el funcionamiento del soporte multitabla incluido.



Fuente: [31]

Figura 1.8 Soporte Multitabla

- Se añade soporte para VLAN y MPLS.
- Se añade soporte para puertos virtuales.
- Para apoyar funcionalidades como *multipath*[12], *multicast*[13], se introduce la función de agrupación en la cual los grupos se pueden crear con un set de acciones y paquetes.

1.3.2.3 OpenFlow 1.2 (diciembre 2011)

- Soporte para reescritura de paquetes.
- Soporte de IPv6.
- Emparejamiento flexible con criterios definidos por el usuario y *bitmasking*[14].

---

[12] **Multi-path:** Es una técnica para conectar un servidor a un dispositivo de almacenamiento con dos conexiones en lugar de una [44].
[13] **Multicast**: Es un método de envío de paquetes (a nivel de IP) que solo serán recibidos por un determinado grupo de *host* [45].
[14] **Bitmasking:** Técnica para representar una sub red usando un vector de bits [46].

- Soporte para control múltiple.

1.3.2.4 OpenFlow 1.3 (junio 2012)

- Manejo de cabecera extendida para IPv6.
- Mediciones por cada flujo.
- Filtrado por eventos de cada conexión para optimización dentro de un ambiente con varios controladores y switches.
- Soporte de *Provider Backbone Bridges*.

1.3.2.5 OpenFlow 1.4 (octubre 2013)

- El protocolo se vuelve extensible mediante el uso de TLV15.
- Soporte para puertos ópticos.
- Mejora de la interacción entre el controlador y el switch con la adición de un monitoreo por cada flujo.

1.3.3 PROVEEDORES DE ACUERDO A LA VERSIÓN DEL PROTOCOLO OPENFLOW

No todos los proveedores publican la versión del protocolo que utilizan, sin embargo, en la Tabla 1.2, se resume algunos de los distribuidores de esta tecnología con sus productos.

Manufacturer	Models/Series	OpenFlow Version
Arista	7050, 7150, 7200, 7300, and 7500 series	1
Brocade	ICX and VDX Products	1.3
Brocade	MLXe, CER and CES Series	1.3
Brocade	NetIron XMR Series	1.3
Extreme Networks	BlackDiamond 8000, BlackDiamond X8, Summit X670	1
Hewlett Packard	2920 Series, 3500/3500yl, 3800, HP 5400 zl	1.0 and 1.3
IBM	IBM Programmable Network Controller, RackSwitch G8264, RackSwitch G8264T, RackSwitch G8332, RackSwitch G8052, RackSwitch G8316	1
Juniper	EX and MX Series	1
NEC	PF5240, PF5820, PF1000	1
NEC	ProgrammableFlow Network Controller PF 6800	1.0, 1.3
Pica8	P-3290, P-3295, P-3780, P3920	1.4

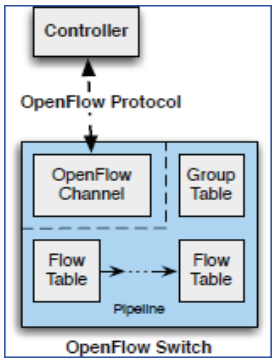
Fuente: [40]

Tabla 1.2 Proveedores OpenFlow

[15] **TLV** (*Type Length Value*): Formato de representar información, de forma que pueda tener presencia opcional y longitud variable.

1.3.4 SWITCH OPENFLOW

Un *switch* OpenFlow consta de un canal para comunicarse con un controlador externo, de una o más tablas de flujo y una tabla de grupo, las mismas que realizan el reenvío y búsqueda de paquetes. En la Figura 1.9 se muestran los principales componentes.



Fuente: [41]

Figura 1.9 Componentes principales de un *switch* OpenFlow

1.3.4.1 Tabla de flujo

Utilizando el protocolo OpenFlow, el controlador puede agregar, actualizar y eliminar entradas de flujo en las tablas, cada tabla de flujo consta de campos para emparejamiento, contadores y un set de instrucciones. En la Figura 1.10 se visualiza una entrada de flujo.

Fuente: [41]

Figura 1.10 Componentes principales de una entrada de flujo dentro de una tabla

- ***Match fields:*** Para emparejar paquetes, consiste en el puerto de ingreso, las cabeceras y, opcionalmente, otros campos.
- ***Priority:*** Marca la prioridad de la entrada de flujo.
- ***Counters:*** Contadores que se actualizan cuando se realiza el emparejamiento de los paquetes.
- ***Instructions:*** Sirven para modificar el procesamiento.
- ***Timeouts:*** Establece el tiempo máximo luego del cual el switch descarta el flujo.

- ***Cookie:*** Valor utilizado por el controlador para filtrar las entradas de flujo.
- ***Flags:*** Para alterar el manejo de las entradas de flujo.

1.3.4.2 Tabla de grupo

Una tabla de grupo está compuesta por entradas de grupo, la misma se aprecia en la Figura 1.11.

Group Identifier	Group Type	Counters	Action Buckets

Fuente: [41]

Figura 1.11 Componentes principales de una entrada de grupo dentro de una tabla

- ***Group identifier:*** Entero de 32 bits que identifica al grupo dentro del switch de forma única.
- ***Group type:*** Para determinar la semántica del grupo.
- ***Counters:*** Contadores que se actualizan cuando los paquetes son procesados por un grupo.
- ***Action buckets:*** Una lista ordenada de acciones para asociar parámetros.

1.3.4.3 Canal OpenFlow

El canal OpenFlow es la interfaz que conecta al *switch* con el controlador; a través de esta, el servidor controlador administra, recibe eventos y envía paquetes al *switch*, el canal debe soportar una o más parejas *switch*-controlador.

1.4 CONTROLADOR FLOODLIGHT

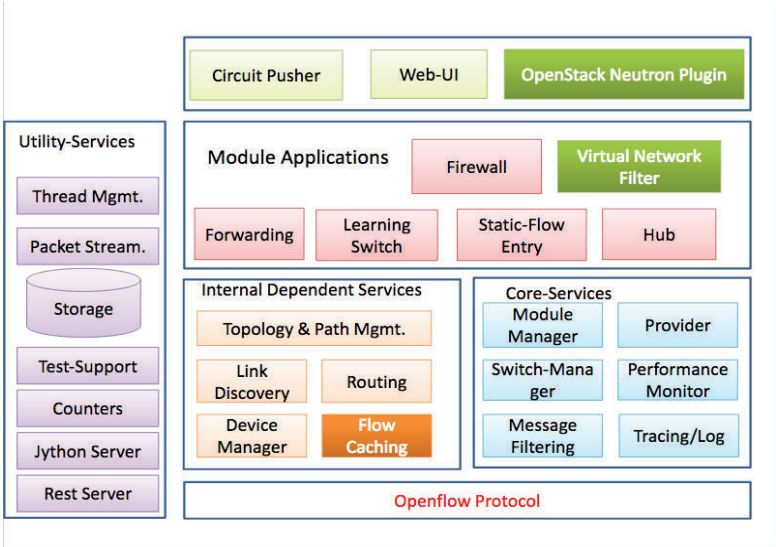
Floodlight es un controlador SDN ofrecido por Big Switch Networks Inc.[16], está diseñado para trabajar con *switches*, *routers*, máquinas virtuales y *access points* que soportan el protocolo OpenFlow, con el objetivo de dirigir el tráfico de datos dentro de una red definida por software. El controlador se originó en la Universidad de Stanford y la Universidad de California en Berkeley y actualmente es soportado por la comunidad de desarrolladores de código abierto, en conjunto con ingenieros especializados en SDN y virtualización. Se encuentra disponible

[16] **Big Switch Networks Inc**.: Compañía estadounidense líder en tecnología SDN.

en descarga gratuita para desarrollo de aplicaciones y fue publicado bajo la licencia de Apache 2.0, permitiendo que el software sea incluido en paquetes comerciales. El controlador se encuentra escrito en Java y tiene la capacidad de adaptarse a diferentes entornos, lo que facilita el desarrollo de aplicaciones, puede trabajar con equipos físicos y virtuales compatibles con OpenFlow; de igual manera, es compatible con Open Stack que es un conjunto de herramientas que ayudan en la construcción y administración de plataformas de Cloud Computing[17].

1.4.1 ARQUITECTURA DE FLOODLIGHT

Floodlight no solo representa a un controlador, sino que también incluye un conjunto de aplicaciones que se encuentran sobre este; para controlar la red OpenFlow ejecuta un set de funcionalidades, mientras que las aplicaciones en la parte superior realizan diferentes acciones para solventar las necesidades de los usuarios de la red. En la Figura 1.12 se aprecia su estructura.



Fuente: [23]

Figura 1.12 Arquitectura de Floodlight

Adopta una arquitectura modular, funcionalmente está compuesto por los módulos de controlador que implementan servicios centrales de la SDN que se

[17] ***Cloud Computing***: Utilización de las instalaciones propias de un servidor web albergadas por un proveedor de Internet para almacenar, desplegar y ejecutar aplicaciones a petición de los usuarios demandantes de las mismas [24].

expondrán a la capa aplicación, y los módulos denominados de aplicación, que implementan soluciones para diferentes propósitos.

1.4.1.1 Módulos de Control de Floodlight

Implementan funciones que comúnmente se utilizan en la mayoría de las aplicaciones:

- Descubrir y exponer los estados y eventos de red (topología, dispositivos, flujos).
- Permite la comunicación del controlador con los dispositivos de la red (protocolo OpenFlow).
- Administrar los módulos Floodlight y compartir recursos como almacenamiento, hilos y pruebas.
- Una interfaz de usuario web y el servidor de depuración (Jython)[18].

A continuación, se enlistan algunos módulos de control:

- ***FloodlightProvider:*** Este módulo se ocupa de las conexiones con los conmutadores y convierte los mensajes OpenFlow en eventos que los otros módulos del controlador pueden interpretar.
- ***DeviceManagerImpl:*** Rastrea como los dispositivos se interconectan a través de la red y define el equipo destino para un nuevo flujo.
- ***RestApiServer:*** *Permite a los módulos exponer la API REST*[19] *mediante HTTP.*
- ***ThreadPool:*** *Es un módulo reflector para el servicio Scheduled Executor Service*[20] *de Java.*
- ***MemoryStorageSource:*** *Proporciona almacenamiento en memoria estilo NoSQL*[21]*.*

---

[18] **Jython**: Es la implementación del lenguaje Python (lenguaje de programación de alto nivel) para la plataforma Java.
[19] **REST** (*Representational State Transfer*): Es un estilo de arquitectura enfocada a la comunicación, usualmente utilizada en el desarrollo de servicios web [48].
[20] **Scheduled Executor Service**: Clase de Java que permite calendarizar un *executor*, que son interfaces que permiten crear hebras dentro de un hilo [49].
[21] **NoSQL:** Clase de sistema de gestión de bases de datos que difieren del modelo clásico en aspectos importantes como que no usan SQL como el principal lenguaje de consultas [50].

- ***TopologyService:*** Este módulo mantiene la información de la topología de la red para el controlador y para la obtención de rutas.
- ***LinkDiscoveryManager:*** Es responsable de descubrir y mantener los estados de los enlaces dentro de la red.

1.4.1.2 Módulos de Aplicación

Implementan soluciones para propósitos específicos, los principales son:

- ***VirtualNetworkFilter*****:** Es la capa 2 del modelo ISO/OSI[22] basado en la virtualización de la red, permite crear múltiples capas 2 lógicas, en un dominio simple.
- ***Forwarding*****:** Permite el reenvío de paquetes entre dispositivos.
- ***Firewall*****:** Controla el cumplimiento de las reglas de denegación o permiso del tráfico.
- ***Port Down Reconciliation*****:** Es implementado para conectar flujos, cuando un determinado puerto o enlace deja de funcionar.

1.5 OPEN VSWITCH

Open vSwitch es un conmutador virtual multicapa, se encuentra licenciado por código abierto bajo Apache 2.0[23], soporta múltiples tecnologías de virtualización basados en Linux, incluyendo Xen / XenServer, KVM y VirtualBox.

La mayor parte del software está escrito en leguaje C y es independiente de la plataforma; además se puede transferir fácilmente a diferentes entornos. Entre las principales características que la versión actual de Open vSwitch están:

- QoS (Calidad de Servicio) configurable y reenvío de alto rendimiento utilizando un módulo del *kernel* Linux 2.6.32 y superiores.
- Protocolos de túnel Múltiples.
- OpenFlow 1.4 además de numerosas extensiones.
- Base de datos de configuración transaccional con C y Python.

---

[22] **ISO/OSI**: Modelo de red creado por la Organización Internacional de Estándares.

[23] **Apache**: Es un servidor web HTTP de código abierto para plataformas Unix-like (BSD, GNU/Linux, etc.), Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. [52]

1.6 VIRTUALBOX

Virtualbox es un hipervisor[24] que permite implementar sistemas operativos en un entorno virtual o también llamado máquina virtual tanto en procesadores de 32 bits (X86) así como de 64 bits (AMD/Intel 64). Entre los sistemas operativos soportados (en modo anfitrión) se encuentran Linux, Mac OS, Windows, y OpenSolaris, y dentro de estos es posible virtualizar los sistemas operativos FreeBSD[25], Linux, Windows, OpenSolaris y muchos otros. [53]

Entre sus principales características están:

- VirtualBox es funcionalmente idéntico en todas las plataformas, ya que utiliza los mismos formatos de archivo e imagen. Esto le permite ejecutar máquinas virtuales creadas en un *host* a otro *host* con un sistema operativo diferente; por ejemplo, puede crear una máquina virtual en Windows y luego ejecutarla en Linux.
- A diferencia de otras soluciones de virtualización, VirtualBox puede ejecutarse en hardware antiguo, donde características como Intel VT-x o AMD-V no están presentes.
- Permite la instalación de herramientas para carpetas compartidas, ventanas integradas, virtualización 3D en los sistemas operativos huéspedes o virtualizados.
- Posee un gran soporte de hardware, permite virtualizar los elementos de una máquina física con facilidad y realizar sus configuraciones de forma gráfica.

1.7 ELASTIX

Elastix es un software de código abierto para el establecimiento de comunicaciones unificadas. Pensando en este concepto, el objetivo de Elastix es el de incorporar en una única solución todos los medios y alternativas de comunicación existentes en el ámbito empresarial.

---

[24] **Hipervisor**: Software que permite administrar varios sistemas operativos o varias instancias de un mismo sistema en un único sistema informático.

[25] **FreeBSD:** es un avanzado sistema operativo para arquitecturas de 32 y 64 desarrollada en la Universidad de California, Berkeley [57].

Elastix permite integrar locaciones para centralizar las comunicaciones y llevarlas a niveles globales. Entre sus principales características se encuentran:

- Correo de Voz
- Fax-a-email
- Soporte para *softphones*[26]
- Interfaz de configuración web
- Sala de conferencias virtuales
- Grabación de llamadas
- *Roaming*[27] de extensiones
- Interconexión entre centrales telefónicas
- Identificación del llamante
- Generación avanzada de reportes

[26] **Softphone:** Software que transforma la computadora en un teléfono multimedia, con capacidad de voz, datos e imagen [58].
[27] **Roamming:** Servicio consiste en permitir que un usuario que se encuentre en una zona de cobertura de una red móvil diferente a la que le presta el servicio pueda recibir las llamadas hechas hacia su número móvil [59].

CAPÍTULO II
DISEÑO E IMPLEMENTACIÓN

2.1 INTRODUCCIÓN

En este Capítulo se presenta el diseño e implementación de la SDN utilizada en el Proyecto, sus componentes y configuraciones. De igual manera se describe la aplicación desarrollada que permite brindar calidad de servicio por priorización de tráfico y que se ejecuta sobre el servidor controlador Floodlight.

2.2 DISEÑO E IMPLEMENTACIÓN DE LA SDN

Para que la aplicación que brinda calidad de servicio pueda actuar, se requiere la creación de una red SDN a través de la cual se realice el intercambio de información.

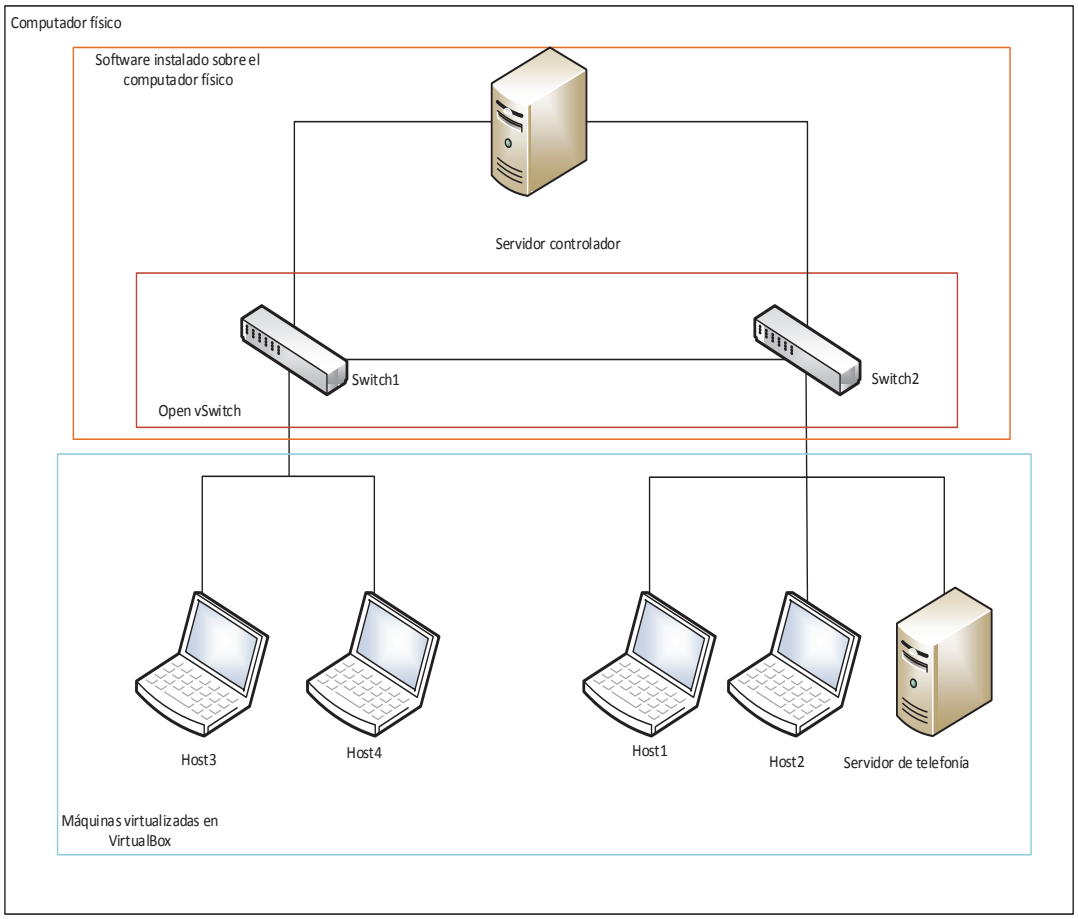


Figura 2.1 Diagrama de la red implementada

La red consta de dos *switches* basados en software implementados con la herramienta Open vSwitch, el controlador Floodlight y cinco máquinas virtuales que corren sobre el hipervisor VirtualBox. De las máquinas virtuales, se emplean cuatro para realizar llamadas telefónicas, envío de diversos paquetes, mediciones, entre otras tareas y uno para el servidor de telefonía Elastix. La SDN se crea sobre un computador físico con sistema operativo Ubuntu en su versión 14.04.3 LTS (*Long-Term Support*). En la Figura 2.1 se visualiza el diagrama de la red implementada.

2.2.1 PLATAFORMA DE HARDWARE Y SOFTWARE

Para la implementación de la red se utiliza un equipo con las características descritas en la Tabla 2.1.

Parámetro	Descripción
Modelo	Toshiba Satélite S55-B5280
Procesador	Intel Core i7 2.6 GHz
Memoria RAM	12,0 GB
Disco Duro	150 GB
Sistema Operativo	Ubuntu 14.04.3 LTS

Tabla 2.1 Especificaciones del computador empleado

Para los *hosts* y el servidor de telefonía, se crean cinco máquinas virtuales con las características descritas en la Tabla 2.2.

El servidor de telefonía tiene como sistema operativo base Centos 6, mientras que para los clientes se utiliza Ubuntu 14.04.3

Parámetro	Descripción
Modelo	Virtualizado
Memoria RAM	1 GB
Disco Duro	15 GB
Sistema Operativo	Ubuntu 14.04.3 LTS/Centos 6

Tabla 2.2 Especificaciones de las máquinas virtuales creadas

2.2.2 DIRECCIONAMIENTO

Dentro de la red se utiliza el direccionamiento descrito en la Tabla 2.3.

Parámetro	Dirección IP
***Host*1**	192.168.2.2
***Host*2**	192.168.2.3
Servidor Elastix	192.168.2.4
***Host*3**	192.168.3.2
***Host*4**	192.168.3.3
Controlador	127.0.0.1:6653

Tabla 2.3 Direccionamiento de red

Por defecto, dentro del controlador se establece la dirección de *loopback* para su interconexión con los dispositivos de red, al ser un sistema totalmente virtualizado, el controlador y los *switches* corren directamente sobre la máquina física, por lo que comparten dicha dirección y no es necesaria su modificación. En la Figura 2.2 se muestra el diagrama con el direccionamiento indicado.

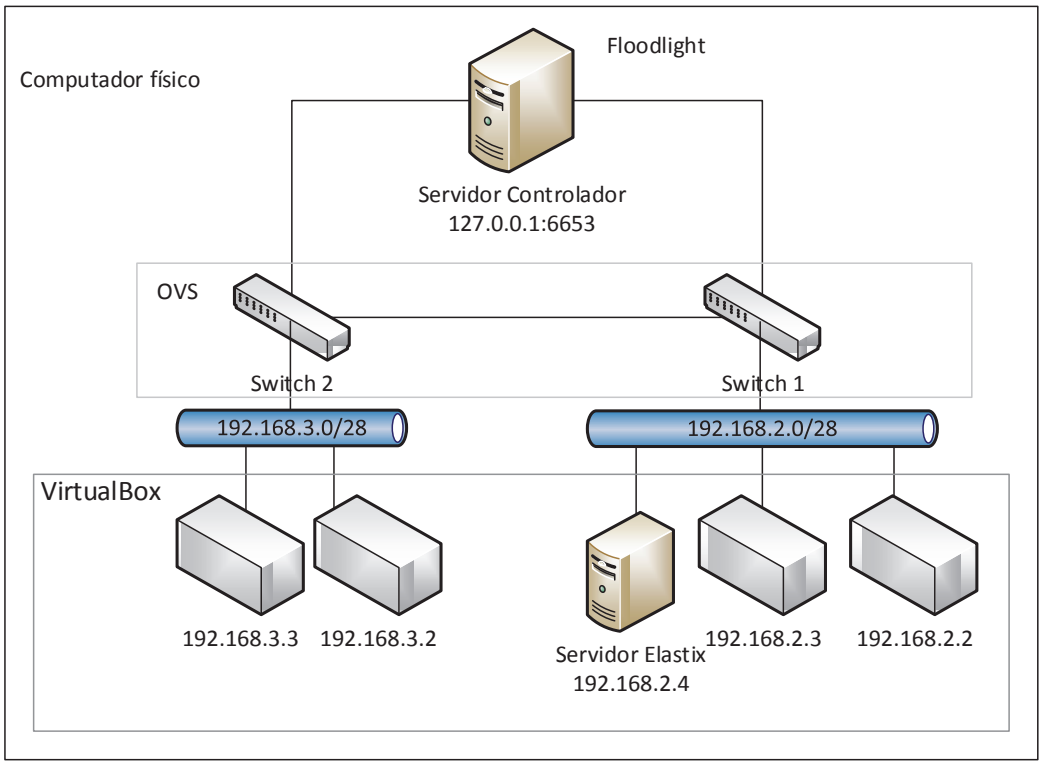


Figura 2.2 Diagrama lógico de la red

2.2.3 CONFIGURACIÓN DE COMPONENTES DE LA RED

2.2.3.1 Configuración de interfaces de red

La configuración de red se realiza de acuerdo al direccionamiento establecido, el computador físico empleado posee una tarjeta de red física alámbrica y una inalámbrica, sin embargo, para la implementación de la SDN, se requieren siete interfaces, es por ello que se utiliza el comando ip tuntap, que permite la creación de interfaces de red virtuales manejadas directamente por el *kernel* de Linux, ante la ausencia de una interfaz de red asociada a una tarjeta de red física.

Se establecen cinco interfaces de red para el enlace entre las máquinas virtuales creadas en VirtualBox y los *switches*, denominadas vnet2, vnet3, vnet4, vnet5 y vnet6, respectivamente, el detalle se muestra en la Figura 2.3.

```
1   ip tuntap add mode tap vnet1
2   ip link set vnet1 up
3   ip tuntap add mode tap vnet2
4   ip link set vnet2 up
5   ip tuntap add mode tap vnet3
6   ip link set vnet3 up
7   ip tuntap add mode tap vnet4
8   ip link set vnet4 up
9   ip tuntap add mode tap vnet5
10  ip link set vnet5 up
11  ip tuntap add mode tap vnet6
12  ip link set vnet6 up
```

Figura 2.3 Creación de interfaces de red

Para el enlace entre los *switches* se utilizan interfaces veth (*virtual ethernet*), que son interfaces virtuales que se crean en pares y están conectadas una con la otra, así el tráfico que es enviado a una interfaz, saldrá por su par correspondiente y viceversa, de esta manera se virtualiza la conexión entre los *switches* posibilitando el envío de tráfico entre estos.

Para crear las interfaces veth se utiliza el comando ip link y se las denomina trunk1 y trunk2, el detalle se muestra en la Figura 2.4.

```
1  sudo ip link add trunk1 type veth peer name trunk2
2  sudo ifconfig trunk1 up
3  sudo ifconfig trunk2 up
```

Figura 2.4 Interfaces para enlace entre *switches*

2.2.3.2 Configuraciones en Open vSwitch

Se realiza la instalación de Open vSwitch, los pasos a seguir se encuentran en el Anexo A. La versión a instalar es la 2.4.0, se escogió esta debido a que es la más reciente y maneja el protocolo OpenFlow en todas sus versiones.

Finalizada la instalación, se procede con las configuraciones respectivas, en primer lugar, se crean dos *switches* br0 y br1, los comandos utilizados se muestran en la Figura 2.5.

```
1  ovs-vsctl add-br br0
2  ovs-vsctl add-br br1
```

Figura 2.5 Creación de un *switch* virtual

Se establecen las características de los *switches* y enlaces respectivos, en la Figura 2.6 se visualizan los comandos utilizados para configurar el *switch* br0. En las líneas uno y dos, se incluye la versión del protocolo que se utiliza, se habilita la opción de encolamiento y se configura el modo a prueba de errores en seguro, el cual indica al *switch* que inicie la tabla OpenFlow sin registros para que esta sea llenada por el controlador, caso contrario el *switch* inicia la tabla con el flujo denominado normal y reenvía los paquetes de forma automática.

En la línea tres, se activa al switch y en las líneas cuatro, cinco y seis se añaden los puertos que se utilizan para conectar las máquinas virtuales al *switch*; por último, en la línea siete se configura la interfaz trunk1.

```
1  ovs-vsctl set bridge br0  fail_mode=secure
2  protocols=OpenFlow13 other_config:in-band-queue=true
3  ifconfig br0 up
4  ovs-vsctl add-port br0 vnet2 -- set interface vnet2 ofport_request=2
5  ovs-vsctl add-port br0 vnet3 -- set interface vnet3 ofport_request=3
6  ovs-vsctl add-port br0 vnet4 -- set interface vnet4 ofport_request=4
7  ovs-vsctl add-port br0 trunk1 -- set interface trunk1 ofport_request=6
```

Figura 2.6 Configuraciones para el *switch* br0

Este proceso se realiza de igual manera para el *switch* br1.

2.2.3.2.1 *Enlace OVS-Floodlight*

Para la instalación del servidor controlador Floodlight, se deben seguir los pasos descritos en el Anexo B. En la Figura 2.7 se aprecia al controlador en ejecución

y a la espera de la conexión de un dispositivo de red, en este caso un *switch* virtual.

```
23:47:42.480 WARN [n.f.c.i.OFSwitchManager:main] SSL disabled. Using unsecure connections between Floodlight and switches.
23:47:42.480 INFO [n.f.c.i.OFSwitchManager:main] Clear switch flow tables on initial handshake as master: TRUE
23:47:42.480 INFO [n.f.c.i.OFSwitchManager:main] Clear switch flow tables on each transition to master: TRUE
23:47:42.480 INFO [n.f.c.i.OFSwitchManager:main] Setting 0x2 as the default max tables to receive table-miss flow
23:47:42.490 INFO [n.f.c.i.OFSwitchManager:main] Setting max tables to receive table-miss flow to 0x1 for DPID 00:00:00:00:00:00:00:01
23:47:42.490 INFO [n.f.c.i.OFSwitchManager:main] Setting max tables to receive table-miss flow to 0x1 for DPID 00:00:00:00:00:00:00:02
23:47:42.962 INFO [n.f.c.i.OFSwitchManager:main] Computed OpenFlow version bitmap as [62]
23:47:42.989 INFO [n.f.c.i.Controller:main] Controller role set to ACTIVE
23:47:43.055 INFO [n.f.l.i.LinkDiscoveryManager:main] Link latency history set to 10 LLDP data points
23:47:43.056 INFO [n.f.l.i.LinkDiscoveryManager:main] Latency update threshold set to +/-0.5 (50.0%) of rolling historical average
23:47:43.134 INFO [n.f.f.Forwarding:main] Default hard timeout not configured. Using 0.
23:47:43.135 INFO [n.f.f.Forwarding:main] Default idle timeout not configured. Using 5.
23:47:43.135 INFO [n.f.f.Forwarding:main] Default priority not configured. Using 1.
23:47:43.135 INFO [n.f.f.Forwarding:main] Default flags will be empty.
23:47:43.135 INFO [n.f.f.Forwarding:main] Default flow matches set to: VLAN=true, MAC=true, IP=true, TPPT=true
23:47:43.135 INFO [n.f.f.Forwarding:main] Not flooding ARP packets. ARP flows will be inserted for known destinations
23:47:43.518 INFO [o.s.s.i.c.FallbackCCProvider:main] Cluster not yet configured; using fallback local configuration
23:47:43.533 INFO [o.s.s.i.SyncManager:main] [32767] Updating sync configuration ClusterConfig [allNodes={32767=Node [hostname=localhost,
23:47:43.762 INFO [o.s.s.i.r.RPCService:main] Listening for internal floodlight RPC on localhost/127.0.0.1:6642
23:47:43.849 INFO [n.f.c.i.OFSwitchManager:main] Listening for switch connections on 0.0.0.0/0.0.0.0:6653
23:47:43.948 INFO [n.f.l.i.LinkDiscoveryManager:main] Setting autoportfast feature to OFF
```

Figura 2.7 Controlador Floodlight en ejecución

Se debe crear el enlace entre el controlador y los *switches*, para ello se utiliza el comando set-controller, como se muestra en la Figura 2.8.

```
1   ovs-vsctl set-controller br0 tcp:127.0.0.1:6653
```

Figura 2.8 Comando para el enlace *switch*-controlador

En la Figura 2.9 se muestra la configuración final del *switch* br0, una vez establecida la conexión con el controlador. La interfaz y puerto br0 son utilizados para asignar una IP al *switch* de ser necesario y se crean de forma automática al momento de la creación del *switch.* El proceso se realiza de igual manera para el *switch* br1.

```
root@tesis:/home/tesis# ovs-vsctl show
cd37ce7b-5e52-4a22-9f53-660f97c982c0
    Bridge "br0"
        Controller "tcp:0.0.0.0:6653"
            is_connected: true
        fail_mode: secure
        Port "vnet2"
            Interface "vnet2"
        Port "br0"
            Interface "br0"
                type: internal
        Port "vnet3"
            Interface "vnet3"
        Port "vnet4"
            Interface "vnet4"
        Port "trunk1"
            Interface "trunk1"
```

Figura 2.9 Configuración de Open vSwitch

Al momento de establecer la conexión entre los *switches* y el controlador, en la consola de Floodlight se muestran los resultados, los cuales se visualizan en la Figura 2.10.

```
21:36:52.599 INFO [n.f.c.i.OFChannelHandler:New I/O worker #19] New switch connection from /127.0.0.1:50130
21:36:52.599 INFO [n.f.c.i.OFChannelHandler:New I/O worker #20] New switch connection from /127.0.0.1:50131
```

Figura 2.10 Floodlight al momento del enlace con los *switches* virtuales

2.2.3.3 Configuración de VirtualBox

Para el intercambio de información se utilizan *hosts* virtuales, para implementarlos se emplea el software VirtualBox, los pasos de su instalación y de la creación de las máquinas virtuales se describen en el Anexo C. Una vez creadas las máquinas virtuales e instalados sus sistemas operativos, se procede con las configuraciones, para lo cual, se establece la interfaz de red en adaptador tipo puente y se escoge la tarjeta de red respectiva para cada máquina virtual; en la Figura 2.11 se aprecia la configuración para el *host* número uno.

Este proceso se repite para cada uno de los *hosts* y el servidor de telefonía. Adicionalmente, dentro de cada *host,* se debe configurar la interfaz de red, se edita el archivo de sistema donde se encuentran las configuraciones de red, con el editor de textos gedit, con el comando: gedit /etc/network/interfaces. En el archivo de texto se debe ingresar la dirección IP y la máscara de red de acuerdo al detalle de la Figura 2.12.

Figura 2.11 Configuración de la interfaz de red en Virtualbox

```
1  auto eth0
2  iface eth0 inet static
3      address 192.168.x.x
4      netmask 255.255.240
```

Figura 2.12 Configuración de interfaces de red

2.2.3.4 Configuración de Elastix

Con el fin de ejecutar las llamadas telefónicas entre los *hosts*, es necesaria la utilización de un servidor de telefonía, para ello se emplea Elastix, cuya instalación se describe en el Anexo D. Finalizada la instalación, se deben realizar ciertas configuraciones dentro del servidor, en primer lugar, se configura la interfaz de red de acuerdo al direccionamiento establecido. Elastix tiene como sistema operativo base una distribución de Centos, para realizar la configuración de red, se edita el archivo /etc/sysconfig/network-scritps con el editor de texto nano que se encuentra preinstalado y se incluye: el dispositivo que se va a configurar, si la configuración será estática o dinámica, la dirección IP, la máscara de red, entre otros. El detalle se aprecia en la Figura 2.13.

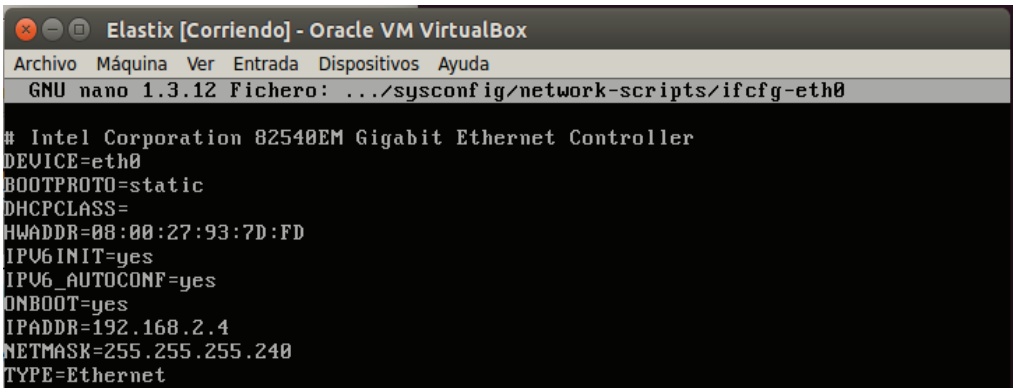


Figura 2.13 Configuración de la interfaz de red en Centos

Adicionalmente se deben crear las extensiones para los *hosts.* En un navegador, se ingresa como URL[28] la dirección IP del servidor de telefonía y se accede a la consola web de Elastix, se ingresa a las opciones: PBX>>Extensions>>Device, como se muestra en la Figura 2.14.
Se escoge el tipo de extensión con un clic en el menú desplegable; para el Proyecto se utiliza dispositivos Generic SIP y se escoge la opción *Submit*.

[28] **URL** (*Uniform Resource Locator*)**:** Ruta que sirve para ubicar de manera precisa en un servidor, cualquier recurso: una imagen, un video o una página web [84].

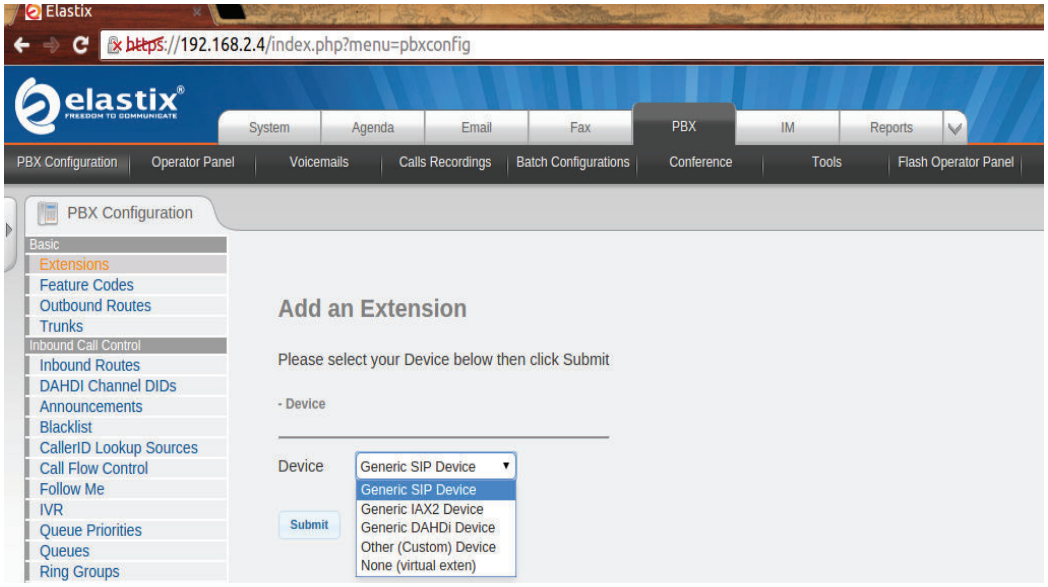


Figura 2.14 Creación de una extensión telefónica

Se desplegarán varias opciones, sin embargo, solo es necesario la configuración de tres:

- Número de extensión y nombre de extensión, que se visualizan en la Figura 2.15.

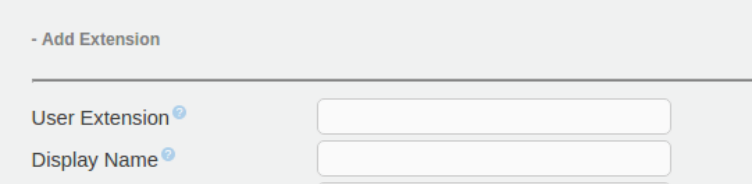


Figura 2.15 Registro de la extensión

- Clave de la extensión, que se muestra en la Figura 2.16.

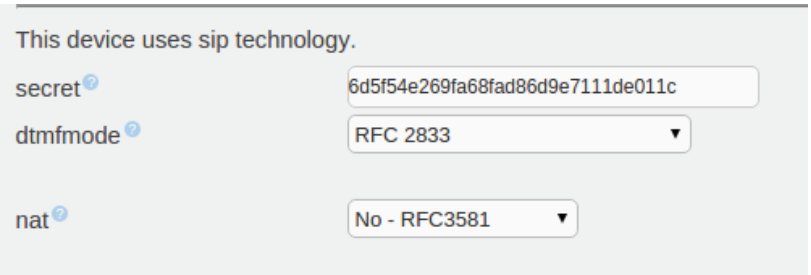


Figura 2.16 Clave de la extensión

Se guardan los cambios y se concluye la configuración.

2.3 DESARROLLO DEL MÓDULO PARA BRINDAR CALIDAD DE SERVICIO

Para el desarrollo de la aplicación, se utiliza Eclipse que es un entorno de desarrollo integrado, de código abierto multiplataforma, permite la programación, desarrollo y compilación en elementos tan variados como sitios web, programas en C++ o aplicaciones Java. Eclipse fue desarrollado originalmente por IBM Canadá como el sucesor de su familia de herramientas para VisualAge[29]. Actualmente es desarrollado por la Fundación Eclipse, una organización independiente sin fines de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. El lenguaje de programación escogido es Java que es orientado a objetos, está basado en el lenguaje C++ con una sintaxis simplificada, fue desarrollado por la compañía Sun Microsystems, con la idea original de usarlo para la creación de páginas web.

2.3.1 DESARROLLO DE LA APLICACIÓN

El módulo a desarrollar dentro del controlador Floodlight, permitirá brindar calidad de servicio durante una llamada telefónica VoIP, para ello, se capturan las tramas enviadas desde los *host*s al servidor y viceversa.

Se realizará un análisis de las cuatro primeras capas del modelo ISO/OSI, de las tramas de establecimiento de la comunicación bajo el protocolo SIP y de las tramas RTP/RTCP que transportan la información de audio y video.

SIP, o Protocolo de Inicio de Sesión, fue desarrollado por el grupo de trabajo MMUSIC[30] del IETF[31] con el fin de ser el estándar en la creación, modificación y finalización de sesiones multimedia con uno o más participantes; se encuentra definido en el RFC[32] 3261.

---

[29] **VisualAge:** es una herramienta de desarrollo rápido de aplicaciones creada por IBM que posee un ambiente de desarrollo integrado de Programación Orientada a Objetos.

[30] **MMUSIC** *(Multiparty Multimedia Session Control):* Grupo de trabajo del IETF.

[31] **IETF** *(The Internet Engineering Task Force):* Es una comunidad internacional de diseñadores de red, operadores, proveedores e investigadores interesados en la evolución de la arquitectura de Internet.

[32] **RFC** *(Request for Comments):* Publicaciones que describen diversos aspectos del funcionamiento de Internet y otras redes de computadoras, protocolos, entre otros.

RTP o Protocolo de Transporte en Tiempo Real define un formato de paquete estándar para el envío de audio y video sobre Internet. Está definido en el RFC 1889.

Fue desarrollado por el grupo de trabajo de transporte de audio y video, y fue publicado por primera vez en 1996. RTP se utiliza ampliamente en los sistemas de comunicación y entretenimiento tales como la telefonía, aplicaciones de videoconferencias, servicios de televisión y web.

RCTP es utilizado para enviar datos de control entre el emisor y receptor de una secuencia RTP. Los paquetes RTCP son enviados aproximadamente cada cinco segundos, y contienen datos que ayudan a verificar las condiciones de transmisión en el extremo remoto, al ser un protocolo complementario a RTP, de igual manera se encuentra definido en el RFC 1889.

Con la información obtenida de los paquetes capturados, se generan reglas que se instalan en los *switches* que conforman la red, estableciendo así las políticas con las que se tratarán a los flujos de audio y video, brindando la prioridad deseada. Como se comentó, el desarrollo del software se realiza sobre el IDE Eclipse, el proceso para importar Floodlight dentro de este y para crear la clase base para la aplicación, se describe en el Anexo E.

2.3.1.1 Diagrama de flujo

En la Figura 2.17 se presenta el diagrama de flujo del módulo creado. El proceso inicia cuando se tiene un paquete generado por el servidor de telefonía o por algún *host* de la red, independientemente del tipo de paquete que sea, el *switch* lo envía al controlador para que este lo analice.

El primer paso es la captura, Floodlight se encuentra en permanente escucha y recibe la información encapsulada en mensajes OpenFlow. Cuando llegue un paquete, el segundo paso entra en marcha y se procede con el análisis de capa dos, donde se verifica que protocolo se encuentra encapsulado dentro de Ethernet. Posteriormente se realiza la primera decisión, si el paquete que se encuentra dentro de Ethernet es IPv4, continuará con el siguiente proceso, caso contrario pasa a una salida, la cual envía el paquete al módulo Forwarding, que se ocupa de la creación del flujo, la inserción del mismo en el *switch* y el posterior reenvío del paquete de regreso al *switch*. Los siguientes procesos y decisiones son similares, se verifica el paquete IPv4 para determinar si el protocolo de

transporte es UDP y posteriormente se analiza la capa de transporte para conocer de qué puerto proviene y hacia qué puerto se dirige la información de igual manera se verifica que información se encuentra dentro del *payload* de UDP, si las condiciones satisfacen lo deseado, el análisis llega al proceso final, se crea el flujo, se realiza la inserción de la regla que proporciona calidad de servicio en el *switch* y se reenvía el paquete original procesado a través del módulo Forwarding. Si las condiciones no se cumplen, la creación de las reglas de envío será manejada directamente por Forwarding de la forma descrita anteriormente.

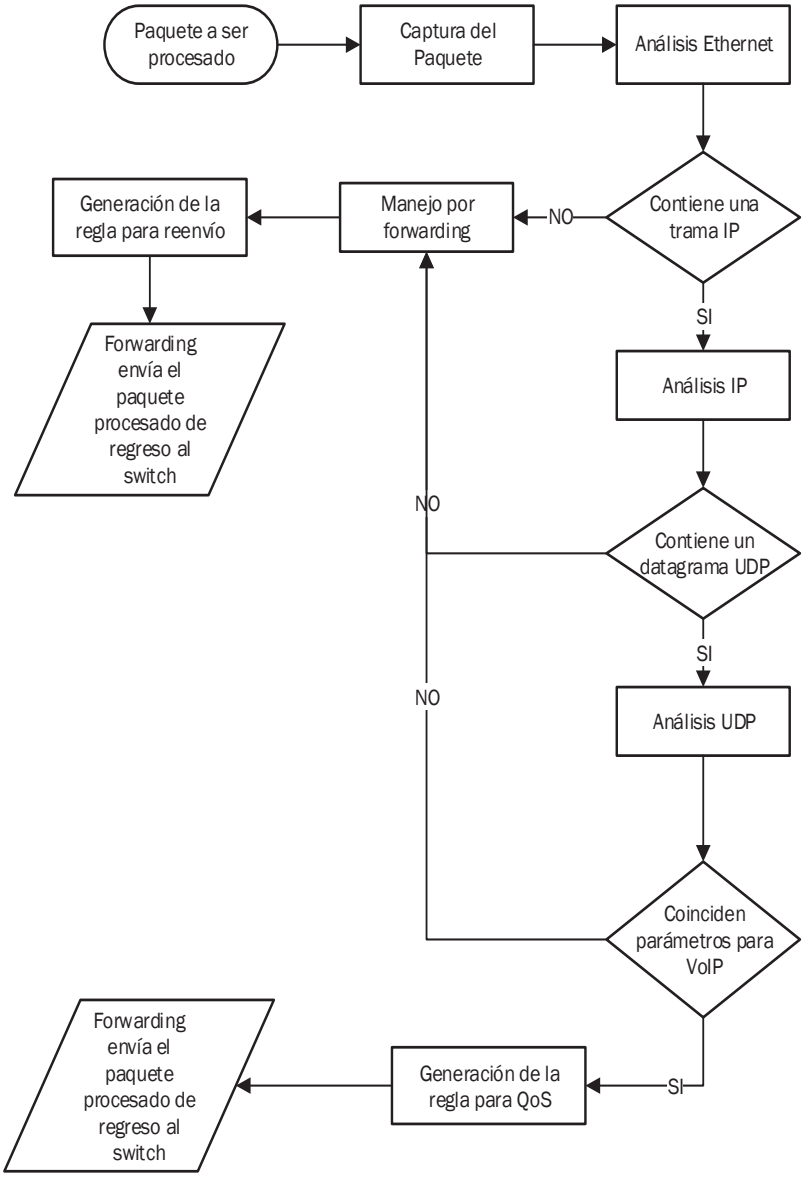


Figura 2.17 Diagrama de flujo

2.3.1.2 Diagrama de Clases

En la Figura 2.18 se presenta el diagrama de clases de la aplicación, en la que se muestra las relaciones del módulo de calidad de servicio, Quality_of_Service.

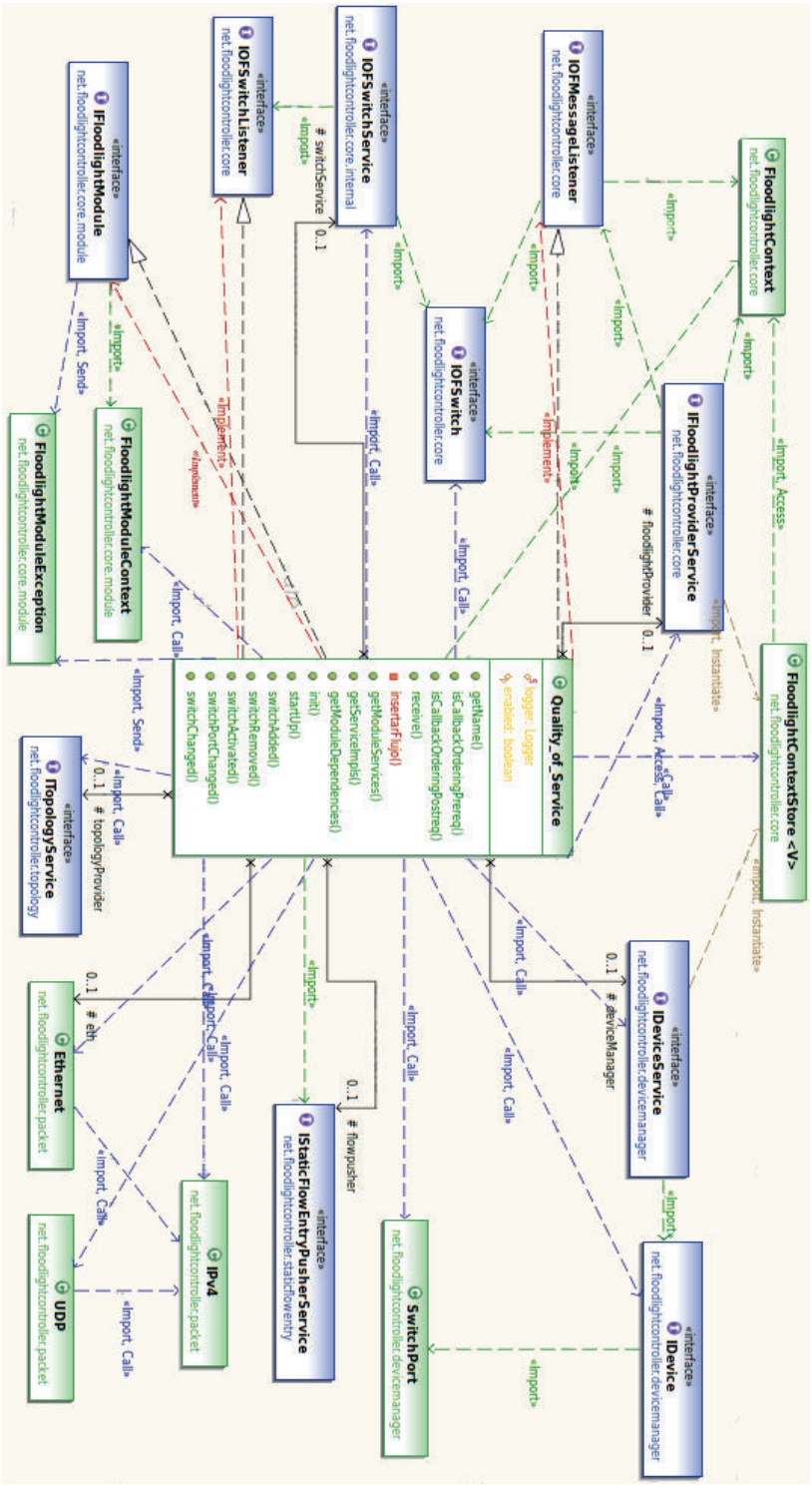


Figura 2.18 Diagrama de clases

A continuación, se describe cada una de las partes del diagrama de clases del módulo Quality_of_Service.

- Las clases pertenecientes al paquete net.floodlightcontroller.deviceManager de Floodlight, son utilizadas para el manejo de los dispositivos. IDeviceService, permite acceder a una colección que contiene los *hosts* que se encuentran conectados a la red; IDevice, por su parte, es utilizado para instanciar objetos que representan a un *host* único dentro de la red, que contiene atributos como direcciones físicas, direcciones de red, el puerto al que está conectado, entre otros, y, finalmente switchPort, a través del cual se puede instanciar objetos del tipo switch DPID-puerto que asocian el identificador único del *switch* con uno de sus puertos.
- Las clases pertenecientes al paquete net.floodlightcontroller.packet que son ethernet, IPv4 y UDP, permiten el manejo de los paquetes a nivel de capa dos, tres y cuatro, respectivamente; son utilizadas para las validaciones descritas en la sección anterior.
- ITopologyService permite el acceso a la topología de la red; dentro del Proyecto se utiliza para conocer en qué puerto de cada *switch* se encuentra el enlace que interconecta a los dos *switches*.
- IStaticFlowEntryPusherService permite enviar las reglas de flujo creadas a los *switches*.
- Las clases IFloodlightService, IFloodlightProviderService e IFloolightModule son utilizadas para el registro de cada uno de los servicios y módulos que se utilizarán, como, por ejemplo, ITopologyService.
- IOFSwitchListener permite la captura del evento que se produce cuando un *switch* se conecta al controlador y facilita el acceso a sus diferentes estados: conectado, desconectado o activo. De igual manera, se utiliza la clase IOFMessageListener para acceder al evento que se produce cuando el controlador recibe un mensaje OpenFlow.
- La clase IOFSwitchService permite obtener una colección de los *switches* que están conectados al controlador; a través de la clase IOFSwitch es posible conocer las características de los *switches*, como, por ejemplo, sus puertos, identificadores únicos, entre otros; la clase PortChangeType

es utilizada para conocer el estado que tiene un puerto, como, por ejemplo, *up* o *down*.

- La clase FloodlightContext es utilizada para que los listeners puedan registrar y, posteriormente, obtener la información asociada con un evento específico; FloodlightModuleContext se utiliza para almacenar información sobre la carga de un módulo, así, cuando el siguiente módulo es cargado, puede obtener la referencia del anterior y acceder a su información; FloodlightContextStore representa un modelo de almacenamiento en caché que guarda y permite obtener información sobre un objeto específico.
- Por su parte, FloodlightModuleException se emplea para la captura de excepciones dentro del código.

2.3.1.3 Descripción del código

Floodlight tiene una estructura modular, cada uno de sus componentes cumple una función específica y está compuesto por una o varias clases; para el módulo de calidad de servicio, solo se crea una clase, sin embargo, se implementan varias de las funcionalidades de los módulos que posee el controlador a través de sus interfaces. En esta sección se describirá el código del módulo Quality_of_Service.

En la Figura 2.19 se presenta la importación de las librerías requeridas para el desarrollo, en la línea uno se encuentra la declaración del paquete qos que contendrá a la clase Quality_of_Service, en las líneas desde la tres a la siete se importan librerías propias de Java que permiten el manejo de arreglos, colecciones, listas y mapas, respectivamente. Finalmente, desde la línea ocho a la veinte y siete se agregan las librerías propias de Floodlight descritas en la sección anterior.

Adicionalmente a las librerías de Floodlight se debe importar librerías para manipular el protocolo OpenFlow que permitirán la interacción con los *switches*, creación e inserción de reglas de flujo e inspección de los paquetes recibidos, de igual manera se incluirán librerías de sfl4j[33] utilizadas para desplegar los *logs* o registros de actividad del módulo; el detalle se visualiza en la Figura 2.20.

[33] **SFL4J (***Simple Logging Facade for Java*): API que en tiempo de ejecución permite vincular librerías para el manejo de *logs* o registros.

```
1  package net.floodlightcontroller.qos;
2  //Libraries
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.util.Collection;
6  import java.util.List;
7  import java.util.Map;
8  import net.floodlightcontroller.core.FloodlightContext;
9  import net.floodlightcontroller.core.internal.IOFSwitchService;
10 import net.floodlightcontroller.core.IFloodlightProviderService;
11 import net.floodlightcontroller.core.IOFMessageListener;
12 import net.floodlightcontroller.core.IOFSwitch;
13 import net.floodlightcontroller.core.IOFSwitchListener;
14 import net.floodlightcontroller.core.PortChangeType;
15 import net.floodlightcontroller.core.module.FloodlightModuleContext;
16 import net.floodlightcontroller.core.module.FloodlightModuleException;
17 import net.floodlightcontroller.core.module.IFloodlightModule;
18 import net.floodlightcontroller.core.module.IFloodlightService;
19 import net.floodlightcontroller.devicemanager.IDevice;
20 import net.floodlightcontroller.devicemanager.IDeviceService;
21 import net.floodlightcontroller.devicemanager.SwitchPort;
22 import net.floodlightcontroller.packet.Data;
23 import net.floodlightcontroller.packet.Ethernet;
24 import net.floodlightcontroller.packet.IPv4;
25 import net.floodlightcontroller.packet.UDP;
26 import net.floodlightcontroller.staticflowentry.IStaticFlowEntryPusherService;
27 import net.floodlightcontroller.topology.ITopologyService;
```

Figura 2.19 Librerías de *Floodlight* y Java

```
28 import org.projectfloodlight.openflow.protocol.OFFactory;
29 import org.projectfloodlight.openflow.protocol.OFFlowAdd;
30 import org.projectfloodlight.openflow.protocol.OFMessage;
31 import org.projectfloodlight.openflow.protocol.OFPortDesc;
32 import org.projectfloodlight.openflow.protocol.OFType;
33 import org.projectfloodlight.openflow.protocol.action.OFAction;
34 import org.projectfloodlight.openflow.protocol.action.OFActionOutput;
35 import org.projectfloodlight.openflow.protocol.action.OFActionSetField;
36 import org.projectfloodlight.openflow.protocol.action.OFActionSetQueue;
37 import org.projectfloodlight.openflow.protocol.action.OFActions;
38 import org.projectfloodlight.openflow.protocol.match.Match;
39 import org.projectfloodlight.openflow.protocol.match.MatchField;
40 import org.projectfloodlight.openflow.protocol.oxm.OFOxms;
41 import org.projectfloodlight.openflow.types.DatapathId;
42 import org.projectfloodlight.openflow.types.EthType;
43 import org.projectfloodlight.openflow.types.IPv4Address;
44 import org.projectfloodlight.openflow.types.IpDscp;
45 import org.projectfloodlight.openflow.types.IpProtocol;
46 import org.projectfloodlight.openflow.types.MacAddress;
47 import org.projectfloodlight.openflow.types.OFPort;
48 import org.projectfloodlight.openflow.types.TransportPort;
49 import org.slf4j.Logger;
50 import org.slf4j.LoggerFactory;
51 import java.io.*;
```

Figura 2.20 Librerías de *OpenFlow* y slf4j

En la línea cincuenta y tres y cincuenta y cuatro de la Figura 2.21, se encuentra la declaración de la clase Quality_of_Service derivada de las clases IFloodlightModule, IOFMessageListener e IOFSwitchListener. En las líneas cincuenta y seis a la setenta se encuentran declaradas variables que se utilizarán posteriormente para el manejo de *hosts*, *switches*, puertos, paquetes, entre otros.

```
52  //Declaración de la clase
53  public class Quality_of_Service implements IFloodlightModule,
54  IOFMessageListener,IOFSwitchListener{
55      //Declaración de variables
56  protected IStaticFlowEntryPusherService flowpusher;
57  protected IFloodlightProviderService floodlightProvider;
58  protected static Logger logger;
59  protected boolean enabled;
60  protected File Configuracion;
61  protected List<String> origenDatos;
62  protected List<String> listaPuertosOrigen;
63  protected List<String> listaPuertosDestino;
64  protected List<String> listaRangoPuertos;
65  protected List<String> listaOctetosValor;
66  protected IOFSwitchService switchService;
67  protected IDeviceService deviceManager;
68  protected OFPort n;
69  protected  Ethernet eth;
70  protected ITopologyService topologyProvider;
```

Figura 2.21 Declaración de la clase para QoS

El primer método a analizar es getName, el cual devuelve el nombre de la clase como un objeto, el mismo que se utiliza para identificar de forma unívoca a OFMessage listener, ya que este se utiliza en varios módulos dentro del controlador. La implementación se muestra en la Figura 2.22.

```
71  //Método getName
72  @Override
73  public String getName() {
74  return Quality_of_Service.class.getSimpleName();
75  }
```

Figura 2.22 Método getName

Para controlar el orden en el cual los diferentes módulos de Floodlight se ejecutan ante un evento específico, se definen los métodos isCallbackOrderingPrereq e isCallbackOrderingPostreq; estos trabajan con un retorno condicional, si la condición es verdadera, la acción de ordenamiento se ejecuta caso contrario no.

Para el Proyecto se utiliza isCallbackOrderingPostreq; al momento que el controlador llama a este método, envía como parámetros de entrada, el evento (OFType) y el nombre del módulo respectivo (name), se verifica si estos valores concuerdan con la recepción de un mensaje PACKET_IN y con el módulo Forwarding, respectivamente; si el resultado es verdadero, se indica a Floodlight

que el módulo Quality_of_Service se debe ejecutar antes que el módulo Forwarding, ya que de lo contrario, este crearía las reglas de reenvío de información y al no contar con una discriminación por tipo de tráfico, no se aplicaría la priorización deseada.

IsCallbackOrderingPrereq no se modifica y se deja el retorno en falso, ya que con excepción de Forwarding cualquier módulo se puede ejecutar antes que Quality_of_Service y el procesamiento de los paquetes y configuración de las reglas de calidad de servicio, no se verán afectadas. El código se visualiza en la Figura 2.23.

```
76  //Metodos para orden de ejecuciónd e módulos
77  @Override
78  public boolean isCallbackOrderingPrereq(OFType type, String name) {
79  // TODO Auto-generated method stub
80  return false;
81  }
82  @Override
83  public boolean isCallbackOrderingPostreq(OFType type, String name) {
84      return (type.equals(OFType.PACKET_IN) && name.equals("forwarding"));
85  }
```

Figura 2.23 Métodos para orden de ejecución de módulos

El método getModuleServices devuelve una colección de las interfaces que el módulo Quality_of_Service implementa, mientras que la función getServiceImpls instancia y devuelve los objetos que cada uno de los servicios utilizados por el módulo proveen, estos métodos se aprecian en la Figura 2.24; para el Proyecto no se utiliza esta información por lo que el retorno se mantiene en nulo, sin embargo, estas funciones son parte de la implementación de la clase y deben ser implementadas. El método getModuleDependencies devuelve al controlador una colección que contiene los módulos de los que Quality_of_Service depende para su funcionamiento, así el controlador verifica que cada una de las dependencias se encuentren correctamente implementadas antes de proceder con la carga.

```
86  //Métodos de servicio para el controlador
87  @Override
88  public Collection<Class<? extends IFloodlightService>> getModuleServices() {
89  return null;
90  }
91  @Override
92  public Map<Class<? extends IFloodlightService>, IFloodlightService> getServiceImpls() {
93  // TODO Auto-generated method stub
94  return null;
95  }
```

Figura 2.24 Funciones getModuleServices y getServiceImpls

Para ello se crea una colección del tipo IFloodligthService, como se aprecia en las líneas noventa y ocho y noventa y nueve de la Figura 2.25. Desde la línea ciento uno a la ciento cuatro, se agregan uno por uno los módulos que requiere Quality_of_Service y finalmente en la ciento cinco se retorna la colección.

```
96  //Registro de dependencias
97  @Override
98  public Collection<Class<? extends IFloodlightService>> getModuleDependencies() {
99  Collection<Class<? extends IFloodlightService>> l =
100         new ArrayList<Class<? extends IFloodlightService>>();
101 l.add(IFloodlightProviderService.class);
102 l.add(IOFSwitchService.class);
103 l.add(IDeviceService.class);
104 l.add(ITopologyService.class);
105 return l;
106 }
```

Figura 2.25 Método getModuleDependencies

Una vez enlazadas las dependencias se las debe inicializar, para ello se usa el método init en el cual se establecen las dependencias y variables que se desee inicializar al momento de la carga del módulo, particularmente, se encuentra la variable enabled que se la describirá más adelante.

Para establecer referencias a los objetos que Quality_of_Service utiliza, se emplea un objeto del tipo FloodlightModuleContext, que a través de su método getServiceImpls, obtiene los objetos y las propiedades que cada uno de los módulos proveen y se almacenan en las variables correspondientes, como se visualiza en la Figura 2.26.

```
107 //Inicialización de dependencias
108 @Override
109 public void init(FloodlightModuleContext context)throws FloodlightModuleException {
110 flowpusher = context.getServiceImpl(IStaticFlowEntryPusherService.class);
111 floodlightProvider = context.getServiceImpl(IFloodlightProviderService.class);
112 switchService= context.getServiceImpl(IOFSwitchService.class);
113 logger = LoggerFactory.getLogger(Quality_of_Service.class);
114 deviceManager = context.getServiceImpl(IDeviceService.class);
115 topologyProvider = context.getServiceImpl(ITopologyService.class);
116 enabled=true;
117 listaPuertosOrigen = new ArrayList<String>();
118 listaPuertosDestino = new ArrayList<String>();
119 listaRangoPuertos = new ArrayList<String>();
120 listaOctetosValor = new ArrayList<String>();
121 Configuracion=new File("/home/tesis/ConfigQoS.txt");
```

Figura 2.26 Método init

Adicionalmente, ya que se realiza la verificación del puerto origen y destino a nivel de capa transporte y del tipo de información que viaja dentro de UDP, se deben establecer parámetros de compasión, los mismos que se obtienen de un archivo de configuración utilizando un objeto del tipo FileReader y almacenando

la información en un arreglo; la implementación se realiza dentro del método init y su detalle se muestra en la figura 2.27.

```
122  try{
123          FileReader r= new FileReader(Configuracion);
124          BufferedReader br= new BufferedReader(r);
125          String temp="";
126          String linea;
127          while((linea=br.readLine())!=null)
128          {
129           temp=temp+linea;
130          }
131          br.close();
132          origenDatos=Arrays.asList(temp.split("=|;|,"));
133      }catch(IOException e){};
```

Figura 2. 27 Lectura del archivo de configuración

Una vez se obtienen los parámetros, se realiza una categorización de los mismos, donde es posible separar un puerto origen o destino en específico, un rango de puertos y para la validación del *payload* de UDP, se obtienen el número de octetos que se van a verificar y el valor que estos tienen, todos los datos son almacenados en arreglos para su posterior utilización. La implementación se detalla en la Figura 2.28. Un ejemplo del archivo de configuración, se adjunta en el Anexo J.

```
134      if (origenDatos != null){
135       for (String a : origenDatos)
136          {
137             switch (a)
138             {
139             case  "PuertoOrigen":
140              int i= origenDatos.indexOf(a);
141              listaPuertosOrigen.add(origenDatos.get(i+1));
142              break;
143             case "PuertoDestino":
144              int n= origenDatos.indexOf(a);
145              listaPuertosDestino.add(origenDatos.get(n+1));
146              break;
147             case "RangoPuertos":
148              int l= origenDatos.indexOf(a);
149              listaRangoPuertos.add(origenDatos.get(l+1));
150              listaRangoPuertos.add(origenDatos.get(l+2));
151              break;
152             case "OctetosValor":
153              int g= origenDatos.indexOf(a);
154              listaOctetosValor.add(origenDatos.get(g+1));
155              listaOctetosValor.add(origenDatos.get(g+2));
156              break;
157             }
158          }
159      }
160  }
```

Figura 2. 28 Categorización de parámetros de comparación

Para que Quality_of_Service pueda acceder a los eventos de conexión de los *switches* y a la recepción de mensajes OpenFlow del tipo PACKET_IN, se agregan en el método startUP dos listeners: OFMessageListener, y OFSwitchListener, que fueron descritos en la sección 2.3.1.2; el detalle se muestra en la Figura 2.29.

```
161   //Declaración de listeners
162   @Override
163   public void startUp(FloodlightModuleContext context) {
164
165   floodlightProvider.addOFMessageListener(OFType.PACKET_IN, this);
166   switchService.addOFSwitchListener(this);
167   }
```

Figura 2.29 Método startUp

Los métodos descritos son la base de cualquier módulo creado en Floodlight, y la implementación de algunos de ellos la realiza Eclipse por defecto al momento de la creación de la clase, de acuerdo a l proceso descrito en el Anexo E.

Una vez estructurado el módulo, se debe crear un método para el manejo de los paquetes y la posterior inserción de la regla de flujo. En la línea ciento setenta y ciento setenta y uno de la Figura 2.30, se define el método para recepción de mensajes, el cual establece tres objetos como parámetros de entrada: un *switch*, un mensaje OpenFlow y el contexto que se utiliza para que el messageListener pueda registrar y posteriormente obtener la información asociada con la recepción del mensaje PACKET_IN.

La primera validación que se realiza es determinar si el módulo se encuentra habilitado, mediante la variable enable que se incluyó en el método init antes descrito. Si el módulo está habilitado (true), se procede con el análisis de la información, caso contrario se envía el paquete al módulo Forwarding mediante la sentencia Command.Continue. En la línea ciento setenta y siete, se define un condicional de selección "switch", para analizar el tipo de paquetes que ingresan al controlador, aunque solo se maneja un caso para los mensajes del tipo PACKET_IN.

Una vez realizada la primera validación, se crea un objeto del tipo Ethernet, en el cual se almacena el *payload* del mensaje enviado al *switch*; de igual manera se crean dos objetos del tipo MacAddress para almacenar la dirección física origen y destino. La segunda validación se realiza en la línea ciento ochenta y seis, donde se verifica si dentro del paquete Ethernet se encuentra un paquete IPv4,

de no ser así se envía la trama al módulo Forwarding. Si es un paquete IPv4, se crea un objeto del tipo IPv4 y se almacena el *payload* del objeto Ethernet antes descrito, adicionalmente se crean dos objetos IPv4Address para almacenar la dirección IP origen y destino. Como se mencionó en la Sección 2.3.1, para establecer una llamada telefónica se utiliza el protocolo SIP y para la transmisión de la voz el protocolo RTP/RTCP, estos son transportados sobre UDP, es por ello que se requiere una validación para determinar si en el paquete IP capturado se encuentra una trama UDP.

```
168   //Método para captura de mensajes OpenFlow
169   @Override
170   public net.floodlightcontroller.core.IListener.Command receive(
171       IOFSwitch sw, OFMessage msg, FloodlightContext cntx) {
172       //Validaciónd de activación de módulo
173       if (!this.enabled)
174       {
175       return Command.CONTINUE;
176       }
177       switch (msg.getType())
178       {
179       case PACKET_IN:
180       //Manejo capa 2
181          Ethernet eth=IFloodlightProviderService.bcStore.get
182                       (cntx,IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
183          MacAddress sourceMac = eth.getSourceMACAddress();
184          MacAddress destMac= eth.getDestinationMACAddress();
185
186       if (eth.getEtherType().equals(EthType.IPv4))
187          {
188       //Manejo capa 3
189          IPv4 ipv4 = (IPv4) eth.getPayload();
190          IPv4Address dstIp = ipv4.getDestinationAddress();
191          IPv4Address sourceIP= ipv4.getSourceAddress();
```

Figura 2.30 Método para recepción de mensajes

En la línea ciento noventa y dos presentada en la Figura 2.31 se encuentra esta validación; si es una trama UDP, se crea un objeto del tipo UDP y en él se almacena el *payload* del objeto IPv4 antes descrito. Adicionalmente, se crean dos objetos del tipo TransportPort en los cuales se almacenan el puerto origen y destino, respectivamente. Ya que sobre el protocolo UDP pueden transportarse distintas aplicaciones, se debe incluir validaciones adicionales, el protocolo SIP para establecer la comunicación utiliza el puerto 5060 de acuerdo al RFC 3261 [26] y RTP/RTCP para el envío de video y voz utiliza un puerto aleatorio entre el número diez mil y el veinte mil de acuerdo a la documentación de Asterisk utilizado para implementar Elastix [27], de acuerdo a esta información se incluyó estos valores en el archivo de configuración. Las validaciones para un puerto

origen y destino específicos considerada para el análisis de SIP se visualizan desde la línea ciento noventa y siete a la doscientos cuatro.

```
192             if (ipv4.getProtocol().equals(IpProtocol.UDP)) {
193         //Manejo capa 4
194             UDP udp = (UDP) ipv4.getPayload();
195             TransportPort desPort = udp.getDestinationPort();
196             TransportPort sourcePort = udp.getSourcePort();
197             if (listaPuertosOrigen != null){
198             for (String a : listaPuertosOrigen){
199             if (Integer.parseInt(a)==(sourcePort.getPort()))
200                {
201                     if (listaPuertosDestino != null){
202                        for (String dp : listaPuertosDestino)
203                        {
204                           if (Integer.parseInt(dp)==(desPort.getPort()))
205                           {
206                               insertarFlujo(sourceMac,destMac,
207                                       sourceIP,dstIp,sourcePort,desPort);
208                           }
209                        }
210
211                     }
212                }
213              }
214             }
```

Figura 2.31 Validación capa 3 y 4

Ya que para el protocolo RTP/RTCP, no se tiene un puerto específico definido si no un rango, se realiza la validación del contenido del *payload* de UDP obteniendo el número de octetos a verificar y su respectivo valor. En la Figura 2.32 se visualiza en primero lugar la validación por rango de puerto y posteriormente la validación del contenido.

```
215         if (listaRangoPuertos != null){
216               for (int i=0; i<listaRangoPuertos.size(); i++){
217                     if (i%2==0){
218                     if(desPort.getPort()>=Integer.parseInt(listaRangoPuertos.get(i)) &
219                      sourcePort.getPort()<Integer.parseInt(listaRangoPuertos.get(i+1)))
220                        {
221                        if (listaOctetosValor != null)
222                         {
223                             for (int j=0; j<listaOctetosValor.size(); j++){
224                                   if (j%2==0){
225                                        Data dataPkt=(Data) udp.getPayload();
226                                         byte[] arr= dataPkt.getData();
227                                           StringBuilder sb = new StringBuilder();
228                                           for (byte b : arr) {
229                                               sb.append(String.format("%02X ", b));
230                                           }
231                                           String m = sb.substring(0,3*Integer.parseInt
232                                                   (listaOctetosValor.get(i)));
233                                           if(m.equals(listaOctetosValor.get(j+1))){
234                                               insertarFlujo(sourceMac,destMac,sourceIP
235                                                   ,dstIp,sourcePort,desPort);
236                                           }
237                                     }
238                                   }
239                         } else
240                         {
241                        insertarFlujo(sourceMac,destMac,sourceIP,dstIp,sourcePort,desPort);
```

Figura 2. 32 Validaciones para RTP/RTCP

Si las condiciones descritas se cumplen, se procede con la inserción de la regla de flujo dentro de los *switches*, para ello se crea un método adicional llamado InsertarFlujo que acepta como parámetros de entrada: direcciones físicas, direcciones IP y puertos tanto de origen como de destino del mensaje PACKET_IN recibido. En primer lugar, se deben crear variables que se utilizan a lo largo del método, estas se visualizan en la Figura 2.33.

```
257   //Método para inserción del flujo
258   private void insertarFlujo(MacAddress sourceMac,MacAddress
259       destMac,IPv4Address sourceIP, IPv4Address destIP,TransportPort
260           sourcePort,TransportPort desPort)
261   {
262   //Declaración de variables
263       SwitchPort[] sp=null;
264       DatapathId sourceDPID=null;
265       DatapathId destinationDPID=null;
266       OFPort patchPortOrigen = null;
267       OFPort patchPortDestino = null;
268       OFPort puertoSwitchOrigen = null;
269       OFPort puertoSwitchDestino = null;
270       long q1=1;
```

Figura 2.33 Método insertarFlujo

En la red se planteó la utilización de dos *switches*, por lo que se presentan dos casos diferentes en la transmisión de la información, el primero cuando los paquetes se envían entre dispositivos conectados a un mismo *switch* y el segundo cuando la transmisión se realiza entre equipos conectados a *switches* diferentes, es por ello que una vez creadas las variables, el siguiente paso a seguir es determinar el dispositivo de donde proviene el paquete, así como su destino, con el objetivo de identificar en cuál de los dos *switches* se encuentra conectado cada host. Mediante el objeto deviceManager que proporciona Floodlight, se almacenan todos los *hosts* que se encuentran conectados a la red en una colección, este proceso se visualiza en la línea dos cientos setenta y dos de la Figura 2.34.

En la línea dos cientos setenta y tres, se recorre la colección creada mediante un lazo for y para cada uno de los elementos, se obtiene su dirección IP a la cual se le compara con la dirección IP destino y origen que se establecieron como parámetros de entrada del método insertarFlujo, las sentencias utilizadas para la comparación, se visualizan en las líneas dos cientos setenta y ocho y dos cientos ochenta y cinco respectivamente, de esta manera se determina que *host* es el que transmite y qué *host* recibe la información. Si el resultado de la comparación es verdadero, con los métodos de la clase IDevice, se obtiene el

DPID (*DataPath ID*) que es un identificador único de 64 bits del *switch* y el puerto al cual está conectado cada *host*.

```
271  //Obtención de puerto del switch origen y destino
272      Iterable<? extends IDevice> hosts=deviceManager.getAllDevices();
273      for (IDevice device : hosts) {
274          if (device != null){
275          IPv4Address[] a = device.getIPv4Addresses();
276          for(IPv4Address n : a)
277          {
278              if (n.equals(sourceIP))
279              {
280                  sp=device.getAttachmentPoints();
281                  for (SwitchPort x:sp){
282                  sourceDPID= x.getSwitchDPID();
283                  puertoSwitchOrigen=x.getPort();}
284              }
285               if (n.equals(destIP))
286              {
287                  sp=device.getAttachmentPoints();
288                  for (SwitchPort x:sp){
289                  destinationDPID=x.getSwitchDPID();
290                  puertoSwitchDestino=x.getPort();}
291              }
292
293           }
294          }
295      }
```

Figura 2.34 Obtención de *host* origen y destino

Para el primer caso de transmisión, si el DPID origen obtenido es igual al destino, implica que los dispositivos están conectados al mismo *switch*, esta validación se visualiza en la línea dos cientos noventa y nueve de la Figura 2.35. Una vez realizada la comparación, se crea un objeto del tipo OFFactory, que permite crear *matches*, acciones y reglas de flujo, e insertarlas en los *switches*.

La versión de la fábrica o Factory creada, se obtiene del propio *switch* que envía el PACKET_IN al controlador, es decir si el *switch* trabaja por ejemplo con una versión de OpenFlow 1.1 la fábrica creada, será para dicha versión. De acuerdo a las configuraciones realizadas en Open vSwitch, la fábrica permitirá trabajar con la versión 1.3 de OpenFlow.

Para la creación de una regla de flujo, en primer lugar se establece un *match*, este le indica al *switch* las características que debe cumplir un paquete para que una acción posterior sea tomada, en este caso se crea un objeto del tipo Match con la ayuda de la fábrica instanciada, lo cual se aprecia en la línea tres cientos tres.

Se utiliza el método setExact para incluir un parámetro de comparación en el *match*. Cada parámetro de comparación tiene un tipo y un valor, como por ejemplo la IP destino (IPV4_DST) y su respectivo valor almacenado en la variable

destIP; el detalle de la implementación se observa desde la línea tres cientos cuatro a la tres cientos doce, con los diferentes campos establecidos.

```
297  //Inserción de los flujos
298  ////Condicional si el flujo se encuentra en el mismo switch
299     if(sourceDPID.equals(destinationDPID))
300      {
301         OFFactory myFactory = switchService.getSwitch(DatapathId.of(sourceDPID.toString())).getOFFactory();
302         //Creación del match
303         Match myMatch = myFactory.buildMatch()
304                         .setExact(MatchField.IN_PORT, puertoSwitchOrigen)
305                         .setExact(MatchField.ETH_TYPE, EthType.IPv4)
306                         .setExact(MatchField.ETH_DST,destMac)
307                         .setExact(MatchField.ETH_SRC,sourceMac)
308                         .setExact(MatchField.IPV4_SRC, sourceIP)
309                         .setExact(MatchField.IPV4_DST, destIP)
310                         .setExact(MatchField.IP_PROTO, IpProtocol.UDP)
311                         .setExact(MatchField.UDP_DST,desPort)
312                         .setExact(MatchField.UDP_SRC,sourcePort)
313                         .build();
```

Figura 2.35 Creación del *match*

Si un paquete cumple con las condiciones, el *switch* lo reconoce y toma acciones sobre este, así el siguiente paso es configurar las acciones indicadas, las cuales se muestran en la Figura 2.36.

Se establecen tres acciones en base al puerto destino, el encolamiento y la configuración del parámetro DSCP en su valor más alto de acuerdo a la tabla que rige estos valores, la misma se encuentra en el Anexo F; así si dicho paquete viaja fuera de la SDN a un *switch* tradicional, tendrá calidad de servicio siempre y cuando este lo soporte.

Cada una de las acciones se implementa de forma individual, se crean tres objetos OFActionsetOutput, OFActionsetQueue y OFActionsetField, que son instanciados por los métodos de los objetos actions buildOutput, buildSetQueue, y buildSetField respectivamente. Estos métodos, reciben como parámetros de entrada los valores puerto destino, ID de la cola y valor DSCP. Para el caso específico de la configuración de DSCP, se emplea la librería OFOxms que Floodlight incluye para la versión 1.3 del protocolo OpenFlow.

Una cola se define como una línea de espera en un sistema, dentro de las redes este concepto permite priorizar el tráfico mediante la asignación de un mayor o menor ancho de banda dentro del enlace; para el Proyecto, se considera que un tipo específico de paquetes, en este caso de voz, tenga una prioridad mayor ante los demás, por ello se crean dos colas directamente en los *switches*, este

proceso se describirá posteriormente, sin embargo, dentro del módulo Quality_of_Service se debe especificar a qué cola se envía la voz sobre IP, así dentro de las variables definidas en el método se incluyó la variable q1 y se le asignó el valor “1”, que es el identificador de la cola que tendrá mayor prioridad, en la línea tres cientos veinte y ocho se aprecia la configuración de la cola.

na vez creadas las acciones, estas son agregadas a un arreglo (actionList) definido en la línea tres cientos catorce, que posteriormente será empleado en la regla de flujo.

Al tener listos los parámetros de comparación y las acciones a tomar, se debe enviar esa información al *switch*. Para esto, se crea un objeto del tipo OFFlowAdd, se le incluye el *match*, las acciones creadas y se establece la prioridad que tendrá la regla. La prioridad se utiliza para evitar duplicidad en las reglas de flujo, por ejemplo, si el módulo Forwarding crea una regla de flujo con el mismo *match* que el módulo de Quality_of_Service, el *switch* obedecerá al que tenga mayor prioridad. Por defecto, los flujos creados por Forwarding tienen prioridad uno, por esta razón se establece la prioridad en ocho. El detalle se aprecia en la Figura 2.37.

```
314         ArrayList<OFAction> actionList = new ArrayList<OFAction>();
315         OFActions actions = myFactory.actions();
316         OFOxms oxms = myFactory.oxms();
317     //Seteo de la prioridad del flujo
318         OFActionSetField setPriority = actions.buildSetField()
319                 .setField(
320                         oxms.buildIpDscp()
321                         .setValue(IpDscp.DSCP_46)
322                         .build()
323                     )
324                 .build();
325         actionList.add(setPriority);
326     //Seteo de la cola para QoS
327         OFActionSetQueue setQueue = actions.buildSetQueue()
328                 .setQueueId(q1)
329                 .build();
330         actionList.add(setQueue);
331     //Seteo del destino
332         OFActionOutput output = actions.buildOutput()
333                .setPort(puertoSwitchDestino)
334                .build();
335         actionList.add(output);
```

Figura 2.36 Creación de acciones

Finalmente, se crea un objeto IOFSwitch que posee el método write utilizado para el envío de la regla de flujo al *switch* y se lo instancia con la ayuda del método getSwitch de switchService, el cual recibe como parámetro de entrada el DPID del *switch* destino. El detalle se observa en las líneas tres cientos cuarenta y cuatro y tres cientos cuarenta y cinco respectivamente.

Para el segundo caso, en el cual el DPID origen es diferente al destino, se realizan ciertas variantes en las reglas de flujo, ya que al tener un intercambio de información entre dos *switches*, se debe insertar dos reglas de flujo simultáneamente.

```
337     //Creación del flujo
338         OFFlowAdd flowAdd = myFactory.buildFlowAdd()
339                     .setPriority(8)
340                     .setMatch(myMatch)
341                     .setActions(actionList)
342                     .build();
343     //Envío del flujo al switch
344       IOFSwitch mySwitch = switchService.getSwitch(DatapathId.of(sourceDPID.toString()));
345       mySwitch.write(flowAdd);
```

Figura 2.37 Creación y envío de la regla de flujo

Por ejemplo, si el *host* fuente se encuentra en el *switch1*, la regla de flujo a insertar sobre este, indicará que el puerto destino es el enlace de salida (trunk1) y a su vez la regla de flujo ingresada en el *switch2* le indicará que el puerto origen es el enlace de entrada (trunk2). Si el envío de información se realiza de forma contraria, el proceso se repite tomando al *switch*2 como origen.

Para proceder con lo descrito, de los métodos del objeto topologyProvider se obtiene un arreglo con los enlaces que unen los *switches* y se los almacena en la colección linksDestino, se recorre la misma y se extrae el puerto correspondiente, ya que solo existe un enlace, no es necesaria ninguna validación adicional. En la Figura 2.38, se visualiza la obtención del puerto destino (trunk1) para el *switch1* (fuente), el mismo que se almacena en la variable patchPortDestino.

```
402     Collection<OFPort> linksOrigen=topologyProvider.getPortsWithLinks(destinationDPID);
403     for(OFPort a: linksOrigen)
404       {
405           patchPortOrigen=a;
406       }
```

Figura 2.38 Determinación de los puertos de entrada y salida del flujo

Al igual que el proceso seguido para la inserción de la regla de flujo cuando el dispositivo origen y destino se encuentran conectados al mismo *switch*, se debe crear la fábrica y a través de esta establecer el *match*, crear el arreglo con las acciones a tomar y finalmente insertar los flujos en los *switches.* En la Figura 2.39 se aprecia el código para la creación e inserción de la regla de flujo en el *switch* origen, el mismo proceso se repite para la regla de flujo en el destino.

```
357         OFFactory myFactory =switchService.getSwitch(sourceDPID).getOFFactory();
358         ArrayList<OFAction> actionList = new ArrayList<OFAction>();
359         OFActions actions = myFactory.actions();
360         OFOxms oxms = myFactory.oxms();
361       //Seteo de la prioridad del flujo
362         OFActionSetField setPriority = actions.buildSetField()
363                 .setField(
364                         oxms.buildIpDscp()
365                         .setValue(IpDscp.DSCP_46)
366                         .build()
367                     )
368                 .build();
369         actionList.add(setPriority);
370       //Seteo de la cola para QoS
371         OFActionSetQueue setQueue = actions.buildSetQueue()
372                 .setQueueId(q1)
373                 .build();
374         actionList.add(setQueue);
375       //Seteo del destino
376         OFActionOutput output = actions.buildOutput()
377                .setPort(patchPortDestino)
378               .build();
379         actionList.add(output);
380       //Creación del match
381         Match myMatch = myFactory.buildMatch()
382                     .setExact(MatchField.IN_PORT, puertoSwitchOrigen)
383                     .setExact(MatchField.ETH_TYPE, EthType.IPv4)
384                     .setExact(MatchField.ETH_DST,destMac)
385                     .setExact(MatchField.ETH_SRC,sourceMac)
386                     .setExact(MatchField.IPV4_SRC, sourceIP)
387                     .setExact(MatchField.IPV4_DST, destIP)
388                     .setExact(MatchField.IP_PROTO, IpProtocol.UDP)
389                     .setExact(MatchField.UDP_DST,desPort)
390                     .setExact(MatchField.UDP_SRC,sourcePort)
391                     .build();
392       //Creación del flujo
393          OFFlowAdd flowAdd = myFactory.buildFlowAdd()
394                     .setPriority(8)
395                     .setMatch(myMatch)
396                     .setActions(actionList)
397                     .build();
398       //Envío del flujo al switch
399         IOFSwitch mySwitch = switchService.getSwitch(DatapathId.of(sourceDPID.toString()));
400         mySwitch.write(flowAdd);
```

Figura 2.39 Creación de la regla de flujo1

De acuerdo al diagrama de flujo descrito en la sección 2.3.1.1, los paquetes que no cumplan con las condiciones establecidas, es decir que no pertenezcan a una llamada telefónica, serán manejados por Forwarding; la función de encolamiento

no existe dentro de este módulo, por lo que se modificó la clase forwardingbase y se incluyó el encolamiento utilizando el comando setQueue con un valor de “0” que es el identificador de la cola que tendrá menor prioridad, el código se encuentra resaltado en la Figura 2.40.

```
239             mb.setExact(MatchField.IN_PORT, inPort);
240             actions.add( sw.getOFFactory().actions().setQueue(0));
241             aob.setPort(outPort);
242             aob.setMaxLen(Integer.MAX_VALUE);
243             actions.add(aob.build());
```

Figura 2.40 Modificación de forwardingbase

El módulo Forwardingbase emplea el match y las acciones para crear una regla de flujo, al igual que en Quality_of_Service. En la línea dos cientos treinta y nueve, se incluye el parámetro de comparación IN_PORT al *match* (mb) y en las líneas siguientes, se configuran las acciones.

El código completo de la aplicación se encuentra en el Anexo G.

CAPÍTULO III
PRUEBAS Y VALIDACIONES

3.1 INTRODUCCIÓN

Una vez concluido el desarrollo de la aplicación, en este capítulo se procede con las pruebas de funcionamiento y el análisis de los resultados. Se comprobará la conectividad entre los dispositivos de red, la calidad en las llamadas telefónicas, el funcionamiento del módulo desarrollado dentro del controlador, así como la validez de la solución, realizando mediciones de latencia[34], *jitter*[35] y capturas de tráfico. Adicionalmente, se incluirá un presupuesto referencial del costo de la implementación del Proyecto.

3.2 CONECTIVIDAD DE LA RED

En primer lugar, se procede con las validaciones de las interfaces de red creadas, para ello se ejecuta el comando ifconfig dentro del terminal de Ubuntu en la máquina física.

```
root@tesis:/home/tesis# ifconfig
eth0      Link encap:Ethernet  direcciónHW c4:54:44:cc:22:26
          Direc. inet:192.168.0.104  Difus.:192.168.0.255  Másc:255.255.255.0
          Dirección inet6: fe80::c654:44ff:fecc:2226/64 Alcance:Enlace
          ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:78 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:124 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:52513 (52.5 KB)  TX bytes:14356 (14.3 KB)

lo        Link encap:Bucle local
          Direc. inet:127.0.0.1  Másc:255.0.0.0
          Dirección inet6: ::1/128 Alcance:Anfitrión
          ACTIVO BUCLE FUNCIONANDO  MTU:65536  Métrica:1
          Paquetes RX:197 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:197 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:0
          Bytes RX:16502 (16.5 KB)  TX bytes:16502 (16.5 KB)

trunk1    Link encap:Ethernet  direcciónHW f6:1f:3c:ed:1a:54
          Dirección inet6: fe80::f41f:3cff:feed:1a54/64 Alcance:Enlace
          ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:18 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:18 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:3015 (3.0 KB)  TX bytes:2851 (2.8 KB)

trunk2    Link encap:Ethernet  direcciónHW 5a:e7:84:0f:eb:04
          Dirección inet6: fe80::58e7:84ff:fe0f:eb04/64 Alcance:Enlace
          ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:18 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:18 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:2851 (2.8 KB)  TX bytes:3015 (3.0 KB)
```

Figura 3.1 Interfaces de red (Continúa)

[34] **Latencia:** Tiempo que tarda un dato en estar disponible desde que se realiza su petición [93].
[35] **Jitter:** Cualquier desviación en los pulsos de una señal digital de alta frecuencia [92].

```
vnet2     Link encap:Ethernet  direcciónHW 9e:37:f9:32:58:7d
          ACTIVO DIFUSIÓN MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:500
          Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)

vnet3     Link encap:Ethernet  direcciónHW fa:5a:5f:36:33:10
          ACTIVO DIFUSIÓN MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:500
          Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)

vnet4     Link encap:Ethernet  direcciónHW aa:0e:ad:a9:11:b7
          ACTIVO DIFUSIÓN MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:500
          Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)

vnet5     Link encap:Ethernet  direcciónHW 02:89:88:19:63:fd
          ACTIVO DIFUSIÓN MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:500
          Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)

vnet6     Link encap:Ethernet  direcciónHW 92:20:48:c7:b3:21
          ACTIVO DIFUSIÓN MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:500
          Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  direcciónHW 30:3a:64:b7:a4:da
          ACTIVO DIFUSIÓN MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figura 3.2 Interfaces de red

Las interfaces de red se configuraron de forma correcta y de acuerdo a lo esperado, en la Figura 3.1 se aprecian las cinco interfaces de red utilizadas para la conexión de las máquinas virtuales con los *switches* así como las dos interfaces veth utilizadas para la conexión entre los *switches*. Adicionalmente, se encuentran las interfaces eth0, lo y wlan0 propias de la máquina física. Con las interfaces de red creadas, se continúa con las pruebas de conectividad, se pone en funcionamiento todos los dispositivos, los *switches*, *hosts* y servidores. Se envían paquetes ICMP desde un *host* al servidor y a los tres *host*s adicionales, el equipo origen es el número uno con dirección IP 192.168.2.2. El detalle se muestra en la Figuras 3.2 y en la Figura 3.3 respectivamente.

```
root@host1:/home/host1# ping 192.168.2.3
PING 192.168.2.3 (192.168.2.3) 56(84) bytes of data.
64 bytes from 192.168.2.3: icmp_seq=1 ttl=64 time=177 ms
64 bytes from 192.168.2.3: icmp_seq=2 ttl=64 time=0.593 ms
64 bytes from 192.168.2.3: icmp_seq=3 ttl=64 time=0.306 ms
```

Figura 3.3 Conectividad entre *hosts* (Continúa)

```
root@host1:/home/host1# ping 192.168.3.3
PING 192.168.3.3 (192.168.3.3) 56(84) bytes of data.
64 bytes from 192.168.3.3: icmp_seq=1 ttl=64 time=3.19 ms
64 bytes from 192.168.3.3: icmp_seq=2 ttl=64 time=0.400 ms
64 bytes from 192.168.3.3: icmp_seq=3 ttl=64 time=0.279 ms
64 bytes from 192.168.3.3: icmp_seq=4 ttl=64 time=0.350 ms
```

Figura 3.4 Conectividad entre *hosts*

```
root@host1:/home/host1# ping 192.168.2.4
PING 192.168.2.4 (192.168.2.4) 56(84) bytes of data.
64 bytes from 192.168.2.4: icmp_seq=1 ttl=64 time=5.87 ms
64 bytes from 192.168.2.4: icmp_seq=2 ttl=64 time=0.434 ms
64 bytes from 192.168.2.4: icmp_seq=3 ttl=64 time=0.246 ms
```

Figura 3.5 Conectividad con el servidor Elastix

Una vez realizada la prueba, se verifica que cada uno de los *switches*, *hosts* y el servidor tiene conectividad con cualquier equipo de la red.

3.3 VERIFICACIÓN DE OPEN vSWITCH

Para comprobar que Open vSwitch (OvS) se encuentra funcionando correctamente, una vez inicializado, se ejecutan diferentes comandos para validar su comportamiento.

- **ovs-vsctl show:** Muestra los puertos del *switch* virtual y su estado. El resultado de muestra en la Figura 3.4.

```
root@tesis:/home/tesis# ovs-vsctl show
300c31d8-ee6c-43e5-9332-a657e312fc86
    Bridge "br0"
        fail_mode: secure
        Port "vnet3"
            Interface "vnet3"
        Port "vnet2"
            Interface "vnet2"
        Port "br0"
            Interface "br0"
                type: internal
        Port "vnet4"
            Interface "vnet4"
        Port "trunk1"
            Interface "trunk1"
    Bridge "br1"
        fail_mode: secure
        Port "br1"
            Interface "br1"
                type: internal
        Port "vnet6"
            Interface "vnet6"
        Port "vnet5"
            Interface "vnet5"
        Port "trunk2"
            Interface "trunk2"
```

Figura 3.6 Puertos de los *switches* virtuales

- **ovs-vsctl --version:** Muestra la versión instalada de Open vSwitch. El resultado de muestra en la Figura 3.5.

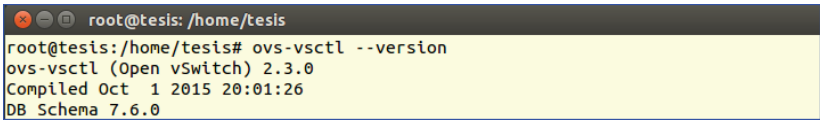


Figura 3.7 Versión de los *switches* virtuales

- **ps -ea | grep ovs:** Muestra los procesos de OvS en ejecución. El resultado de muestra en la Figura 3.6.

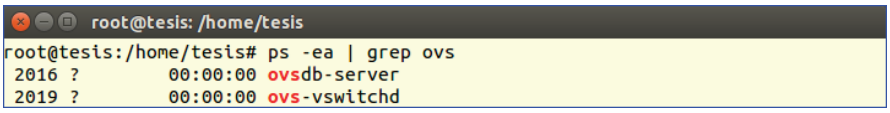


Figura 3.8 Procesos de Open vSwitch en ejecución

3.4 ARCHIVO DE CONFIGURACIÓN

De acuerdo a lo indicado, se utiliza un archivo de configuración para determinar que puerto y que información dentro del *payload* de UDP se va a priorizar. La estructura del archivo se muestra en la Figura 3.7.

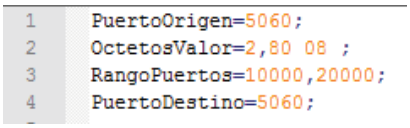


Figura 3.7 Archivo de configuración

En la línea uno se incluye el puerto origen y en la línea cuatro el puerto destino, esta configuración es utilizada para priorizar las tramas SIP, las cuales de acuerdo al RFC 3261 de la IETF, utiliza el puerto 5060.

El rango de puertos que se encuentra en la línea tres, es utilizado para establecer la priorización para RTP/RTCP, para este protocolo no existe un puerto específico relacionado, por lo que se toma en consideración la configuración por defecto de Elastix, que utiliza un puerto aleatorio entre 10000 y 20000.

Ya que el rango de puertos es grande se incluye una validación adicional a nivel del *payload* de UDP, en la línea dos se establece el número de bytes que se van a verificar y el valor que estos contienen; para RTP se chequean los dos

primeros, de acuerdo al RFC 3550 de la IETF, estos contienen la versión del protocolo 2 bits, *padding* 1 bit, *extension* 1 bit, *CSRC* 4 bits y la codificación de audio 8 bits.

Cada una de las opciones: puerto destino, origen, rango de puerto y octetos-valor, se pueden repetir las veces que se desee, con el propósito de incluir más valores y priorizar otro tipo de tramas que viajen sobre UDP.

3.5 VALIDACIÓN DEL CONTROLADOR

Se accede a la interfaz web del controlador para verificar el funcionamiento, dentro de un navegador, se ingresa a la URL http://localhost:8080/ui/index.html. En la primera pantalla denominada *Dashboard*, se observa todos los módulos que fueron ejecutados al momento de la carga de Floodlight, entre estos el módulo de calidad de servicio. De igual manera se visualiza el *uptime*, el *hostname* y los dispositivos que se encuentran en la red. En la Figura 3.7 se aprecia lo descrito.

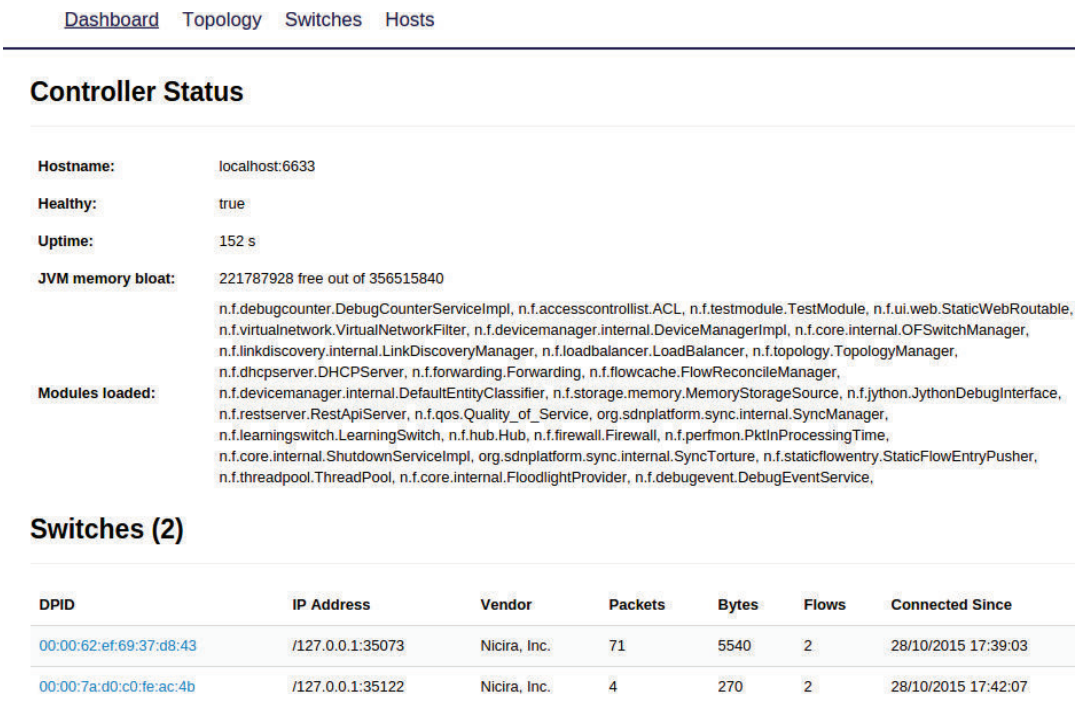


Figura 3.9 *Dashboard* de Floodlight

En la opción *Topology*, se observa la topología de la red; se realizó cambios en los gráficos que representan a los *switches* y a los *hosts* para una mejor

presentación. En la Figura 3.8 se muestra la topología creada por Floodlight y en la Figura 3.9, los gráficos modificados.

Figura 3.10 Topología original

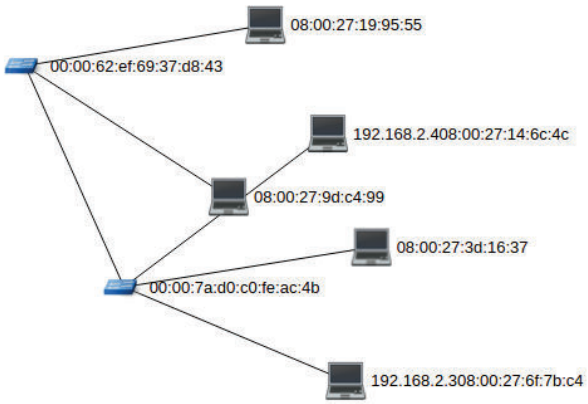


Figura 3.11 Topología modificada

Dentro del enlace *Switches*, se encuentra toda la información referente a los equipos de la red. En la Figura 3.9 se aprecia los dos *switches* con el número de paquetes que han intercambiado y el conteo de flujos que poseen.

Dashboard Topology Switches Hosts

Switches (2)

DPID	IP Address	Vendor	Packets	Bytes	Flows	Connected Since
00:00:62:ef:69:37:d8:43	/127.0.0.1:35073	Nicira, Inc.	71	5540	2	28/10/2015 17:39:03
00:00:7a:d0:c0:fe:ac:4b	/127.0.0.1:35126	Nicira, Inc.	31	13300	11	28/10/2015 17:42:59

Figura 3.12 Sección *Switches* dentro de Floodlight

Al dar clic sobre un *switch*, se visualizan los flujos insertados por el módulo de calidad de servicio, consta del *match*, la prioridad y las acciones a tomar. Uno de ellos se muestra en la Figura 3.11.

Priority	Match	Apply Actions
8	in_port=3 eth_dst=08:00:27:14:6c:4c eth_src=08:00:27:9d:c4:99 eth_type=0x0x800 ip_proto=0x11 ipv4_src=192.168.3.3 ipv4_dst=192.168.2.4 udp_src=5060 udp_dst=5060	actions:ip_dscp=46,output=5,set_queue=1

Figura 3.13 Regla de flujo insertada en el *switch*

Dentro de las acciones claramente se observa que los paquetes que cumplan con el *match*, serán configurados con la prioridad más alta del DSCP e incluidos en la cola con ID 1. Finalmente, en la opción *Host*, se pueden observar las máquinas virtuales conectadas: el servidor Elastix y los cuatro computadores con sus direcciones físicas; para la visualización de las direcciones IP, se debe esperar que se produzca intercambio de información entre los *hosts* para que el controlador pueda detectarlas. El detalle se muestra en la Figura 3.12,

MAC Address	IP Address	Switch Port	Last Seen
08:00:27:34:69:d1		00:00:96:3a:ca:d5:0c:4d-2	8/12/2015 23:17:50
08:00:27:0c:0c:f0		00:00:26:19:52:77:d9:4a-2	8/12/2015 23:18:01
08:00:27:15:62:23		00:00:96:3a:ca:d5:0c:4d-3	8/12/2015 23:18:14
08:00:27:d0:96:16		00:00:26:19:52:77:d9:4a-4	8/12/2015 23:18:20
08:00:27:8a:fb:7a		00:00:26:19:52:77:d9:4a-3	8/12/2015 23:22:05

Figura 3.14 Opción *Host* en la interfaz de usuario de Floodlight

Si se visualiza la consola de Floodlight, se observa todos los procesos que corren dentro del controlador, como por ejemplo en la Figura 3.13 se muestran los registros de la navegación realizada en la interfaz web.

```
tp://localhost:8080/wm/core/switch/00:00:62:ef:69:37:d8:43/desc/json"
17:41:03.876 DEBUG [LogService:Dispatcher: Thread-41] Processing request to: "ht
tp://localhost:8080/wm/core/switch/00:00:62:ef:69:37:d8:43/aggregate/json"
17:41:03.877 INFO [LogService:Dispatcher: Thread-39] 2015-10-28 17:41:03       1
27.0.0.1        -       -       8080    GET     /wm/core/switch/00:00:62:ef:69:3
7:d8:43/desc/json       -       200     -       0       2       http://localhost
:8080   Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/46.0.2490.71 Safari/537.36        http://localhost:8080/
17:41:03.881 INFO [LogService:Dispatcher: Thread-41] 2015-10-28 17:41:03       1
27.0.0.1        -       -       8080    GET     /wm/core/switch/00:00:62:ef:69:3
7:d8:43/aggregate/json  -       200     -       0       5       http://localhost
:8080   Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) C
```

Figura 3.15 Consola de *Floodlight*

3.5.1 INTERCAMBIO DE MENSAJES OPENFLOW

Para la validación del intercambio de tramas entre los dispositivos, se utiliza Wireshark; los detalles de su instalación se muestran en el Anexo G.

Luego de ejecutar el controlador y los *switches,* se captura el tráfico con Wireshark en todas las interfaces de red disponibles y en el filtro se escoge el protocolo Openflow y se espera por el intercambio de información, en la Figura 3.14 se encuentra el filtro seleccionado.

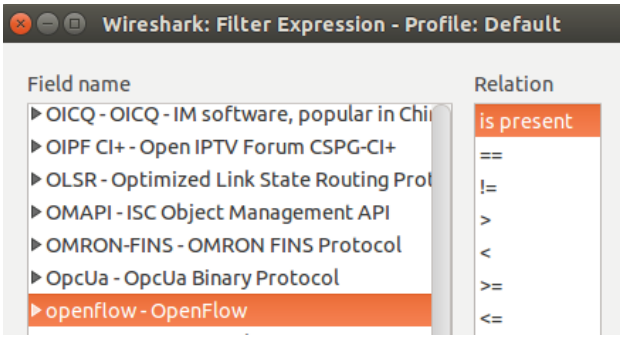


Figura 3.16 Filtro para Openflow en Wireshark

Se visualiza que el intercambio de mensajes OpenFlow es el adecuado y el controlador está en correcta sincronización y funcionamiento. En la Figura 3.15 se muestra el resultado de la captura de mensajes OpenFlow.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
2	0.000000000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
3	0.000112000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
5	0.000184000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
7	2.099862000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
8	2.099862000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
9	2.099979000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
11	2.100040000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
19	2.903952000	127.0.0.1	127.0.0.1	OpenFlow	379	Type: OFPT_PACKET_IN
21	2.903975000	127.0.0.1	127.0.0.1	OpenFlow	356	Type: OFPT_PACKET_IN
23	2.905134000	127.0.0.1	127.0.0.1	OpenFlow	393	Type: OFPT_PACKET_OUT

Figura 3.17 Mensajes Openflow

Los paquetes OFTP_ECHO_REQUEST y OFTP_ECHO_REPLY, son utilizados para el intercambio de información sobre latencia, ancho de banda y estado de la conexión entre el controlador y los dispositivos de red, mientras que OFTP_PACKET_IN y OFTP_PACKET_OUT, son utilizados para el intercambio de paquetes capturados, como, por ejemplo, un paquete IPv4.

3.6 LLAMADAS TELEFÓNICAS

Se procede con la realización de una llamada telefónica entre dos *host*s, para ello se utilizan *softphones* Linphone, que son teléfonos de voz sobre IP que hacen posible comunicarse libremente con la gente a través de Internet, con voz, vídeo y mensajería instantánea. Linphone utiliza el protocolo SIP y se puede utilizar con cualquier operador SIP VoIP, puesto en marcha en 2001; fue la primera aplicación de código abierto utilizando el protocolo SIP en Linux. Durante más de 10 años, una gran cantidad de mejoras se han hecho y Linphone ha sido portado a las principales plataformas de escritorio, móviles y web [85].

Su instalación es simple y se realiza a través del centro de software de Ubuntu, en el cuadro de búsqueda se ingresa la palabra *softphone* y dentro de las opciones desplegadas se escoge Linphone. En la Figura 3.16 se visualiza el cuadro de dialogo para la instalación.

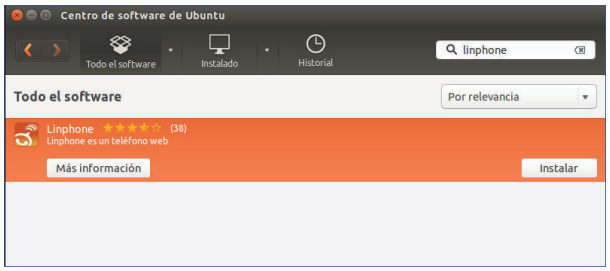


Figura 3.18 Instalación de Linphone

Para poder realizar una llamada, se debe configurar las credenciales de la extensión a la cual se accede, este proceso se denomina registro. Se ingresa el número de la extensión y la dirección IP del servidor de telefonía. Se envía la solicitud de conexión y si las configuraciones tanto de la extensión como del registro son correctas se enlazará el teléfono con el servidor; en la Figura 3.17 se muestra el ingreso de las credenciales para el registro del *softphone*.

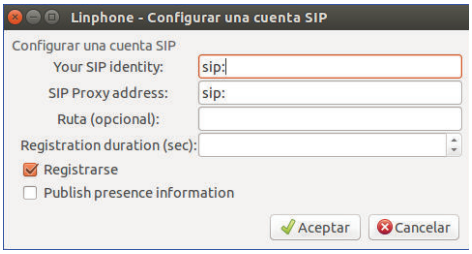


Figura 3.19 Credenciales dentro de Linphone

Adicional al registro se agrega el contacto de Linphone, con el cual se desea realizar la llamada; para ello, dentro de la pantalla principal en la parte inferior derecha, se encuentra la opción para agregar el dispositivo destino. Se despliega un cuadro con la información requerida, el nombre del nuevo registro y la dirección de la extensión seguida de la dirección IP del servidor por ejemplo, 101@192.168.2.4. En la Figura 3.18 se visualiza el cuadro de diálogo para incluir un nuevo contacto.

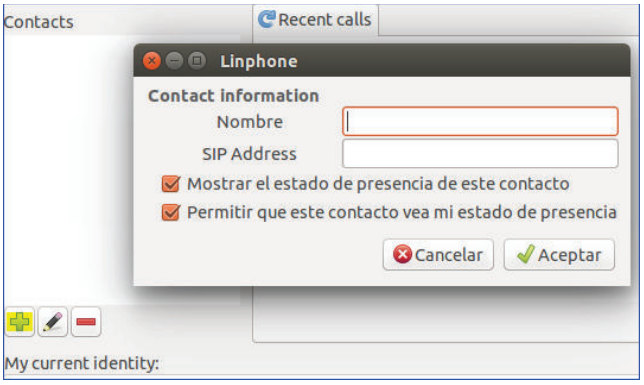


Figura 3.20 Nuevo contacto en Linphone

Una vez creado el contacto se realiza la llamada, esta inicia y los paquetes SIP establecen la conexión entre los teléfonos y, posteriormente, la información se envía sobre RTP. En la Figura 3.19 se observa una llamada en ejecución de la extensión 101@192.168.2.4 a la extensión 104@192.168.2.4; dentro de la red cada uno de los *hosts* puede realizar llamadas telefónicas a los demás.

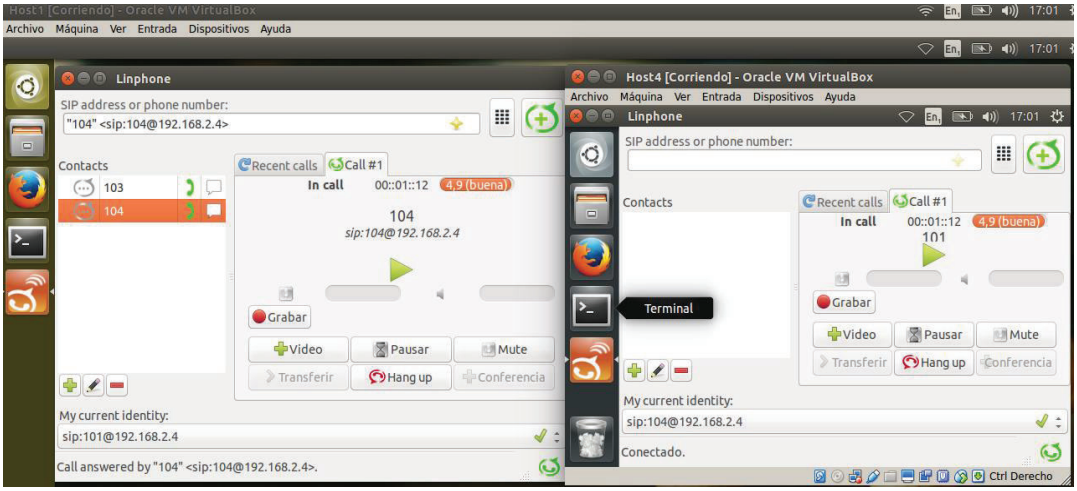


Figura 3.21 Llamada en curso

3.6.1 VERIFICACIÓN DEL INTERCAMBIO SIP Y RTP/RTCP

Para la validación del intercambio de tramas entre los dispositivos durante una llamada, se utiliza el módulo de telefonía de Wireshark, el cual se aprecia en la Figura 3.20.

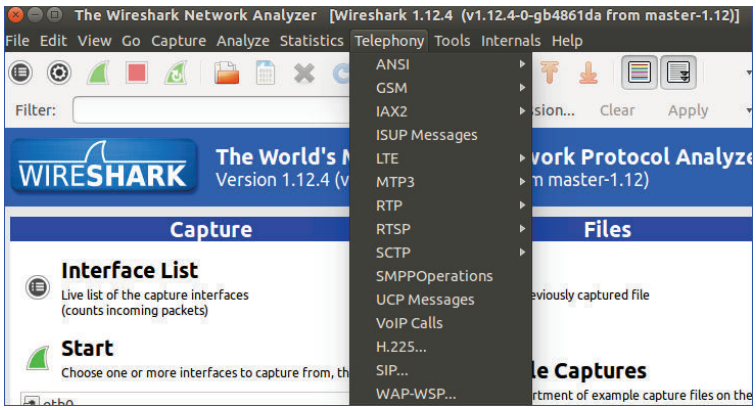
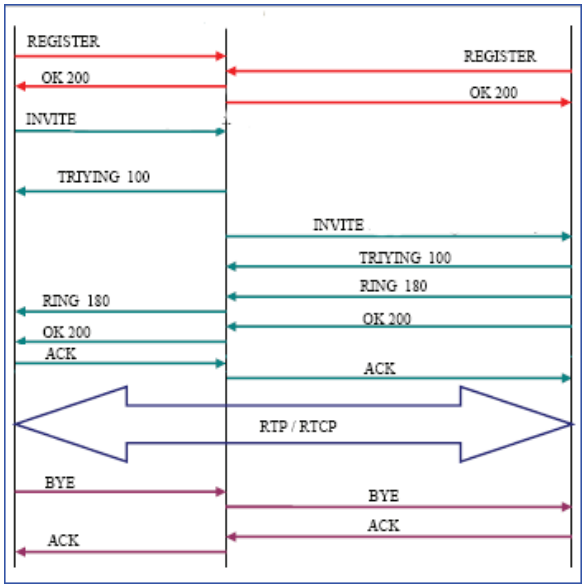


Figura 3.22 Módulo de telefonía en Wireshark

Mediante esta herramienta, se capturan los paquetes que envían los *hosts* y el servidor VoIP; en primer lugar, se verifica el establecimiento de la conexión mediante el protocolo SIP, el intercambio de tramas se lo realiza de acuerdo a la Figura 3.21.



Fuente: [2]

Figura 3.23 Intercambio de tramas SIP

De acuerdo al gráfico, la línea central representa al servidor y las laterales a los terminales. La primera trama es la de registro (*REGISTER*), al ejecutar Linphone dentro de uno de los *hosts*, esta se envía de forma automática al servidor. En la Figura 3.22 se visualiza la trama de registro enviada por un *host*.

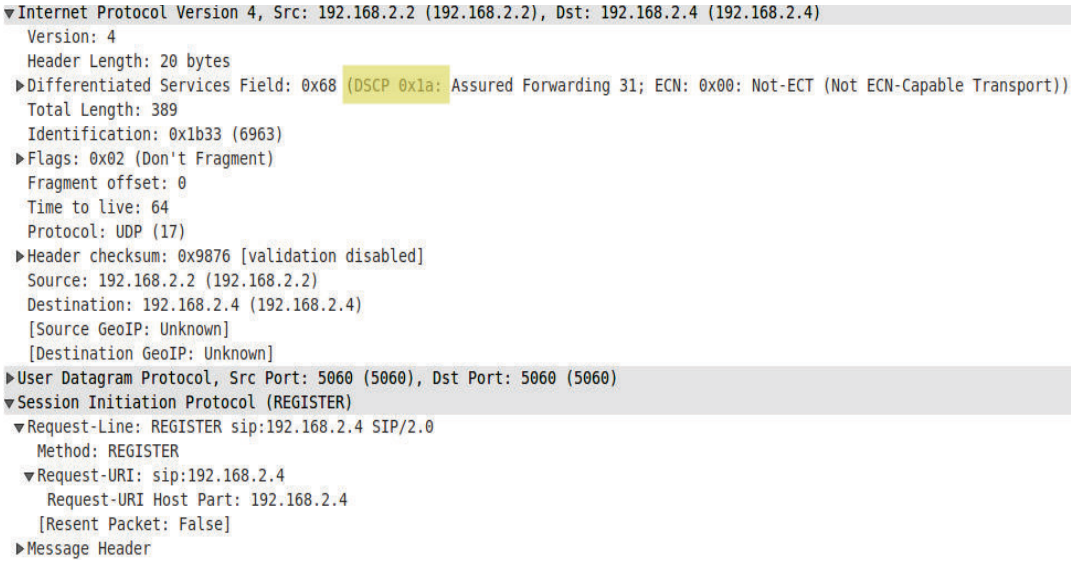


Figura 3.24 Trama de registro

Se puede destacar el hecho de que el número DSCP está configurado con el valor por defecto ya que es el primero que llega al controlador y a partir de este se crea el flujo, sin embargo, para los demás paquetes, la prioridad cambiará y se aplicará la regla establecida por el controlador.

Se responde con una trama de aceptación (*OK*) y se espera por una trama de invitación (*INVITE*) para realizar una llamada.

Al momento de realizar el marcado se emite la solicitud de invitación (*INVITE*) al servidor, este responde con una trama indicando que está intentando establecer la conexión (*TRYING*) y reenvía la solicitud de llamada al *host* destino, el mismo que responde con una trama *TRYING* de igual manera.

Se procede con el timbrado (*RING*) en el destino y el eco del mismo se refleja en el origen, si la llamada es contestada, se envían tramas de confirmación (*OK*) del establecimiento de la llamada y los respectivos *ACK* de cada *host.* Finalizado el intercambio, se procede con el envío de voz sobre RTP.

En la Figura 3.21 se aprecia una trama RTP capturada.

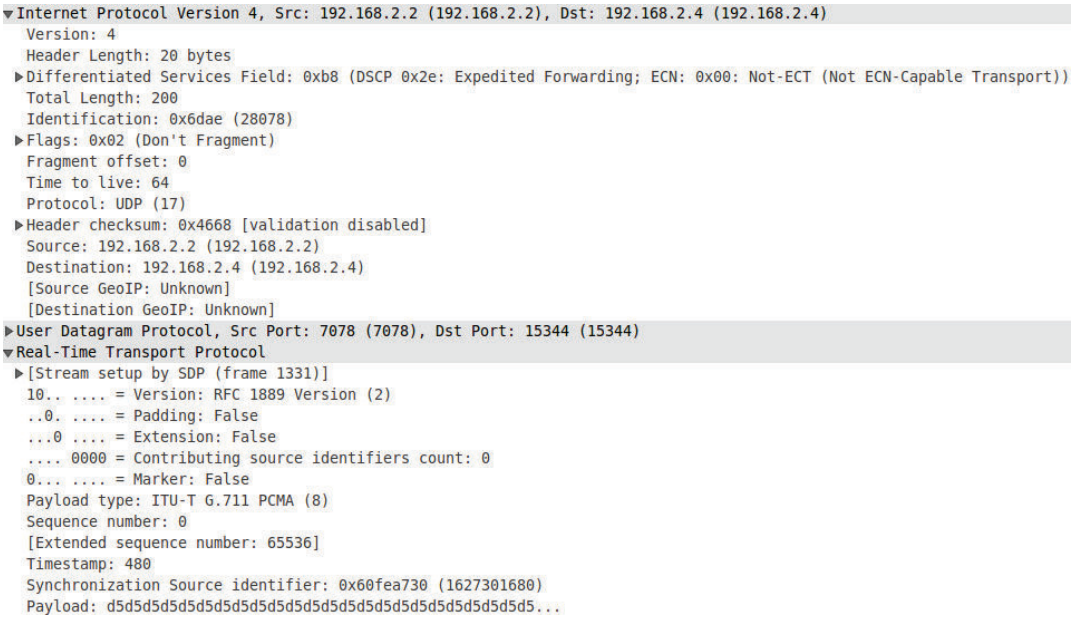


Figura 3.25 Trama RTP

Al finalizar la conversación se envían tramas de desconexión (*BYE*) y confirmación (*ACK*) de la finalización de la llamada. Como se comentó, el campo DSCP de la Figura 3.22 para el protocolo SIP, ha variado el resultado se muestra en la Figura 3.24 en una trama de desconexión.

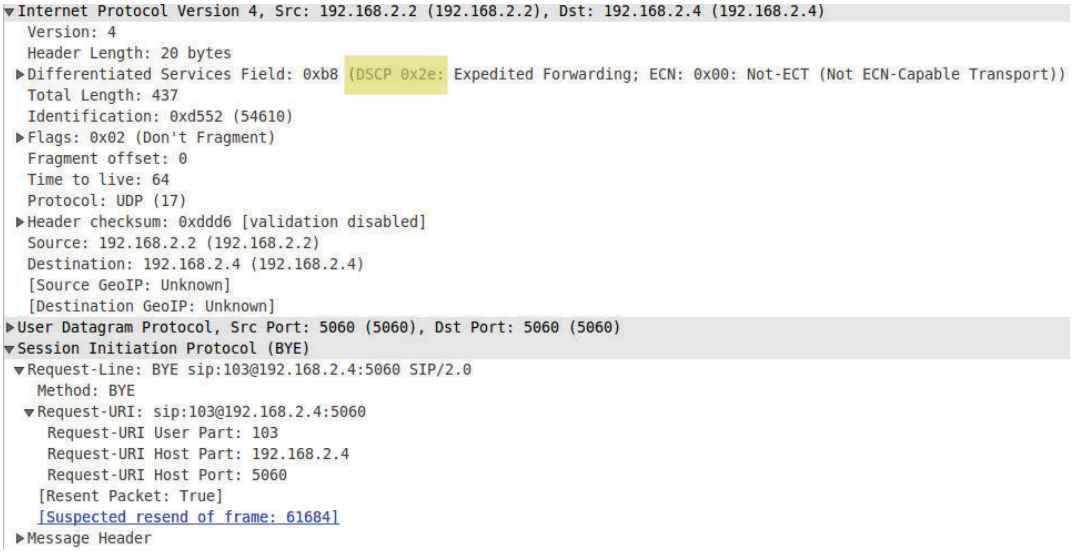


Figura 3.26 Paquete de finalización de llamada

El proceso indicado se ejecutó de forma correcta y el resultado fue una llamada exitosa; el audio es claro y la conexión estable.

3.7 VALIDACIÓN DE ENCOLAMIENTO

Para proceder con las validaciones de encolamiento, se debe determinar en primer lugar, el ancho de banda que se encuentra disponible en los puertos de los *switches*, para ello, se envía tráfico sin la creación ni configuración de colas mediante la utilización del módulo forwarding.

Iperf es una aplicación cliente-servidor que permite determinar el ancho de banda del enlace. La instalación se la realiza desde el repositorio de Ubuntu, mediante la ejecución del comando apt-get install iperf; adicionalmente, este software tiene una interfaz gráfica que permite realizar las mediciones de una manera sencilla, esta se denomina jperf y se la puede descargar de [95], para ejecutarla simplemente se ejecuta el *script* descargado con el comando ./jperf.sh y se desplegará el programa.

La herramienta envía paquetes TCP para la verificación del ancho de banda y UDP para medir *jitter*, en primer lugar, se establece el envío TCP con el valor de 10 tramas que se encuentra por defecto y la medición en Mbps. En la Figura 3.25 se muestra la configuración indicada.

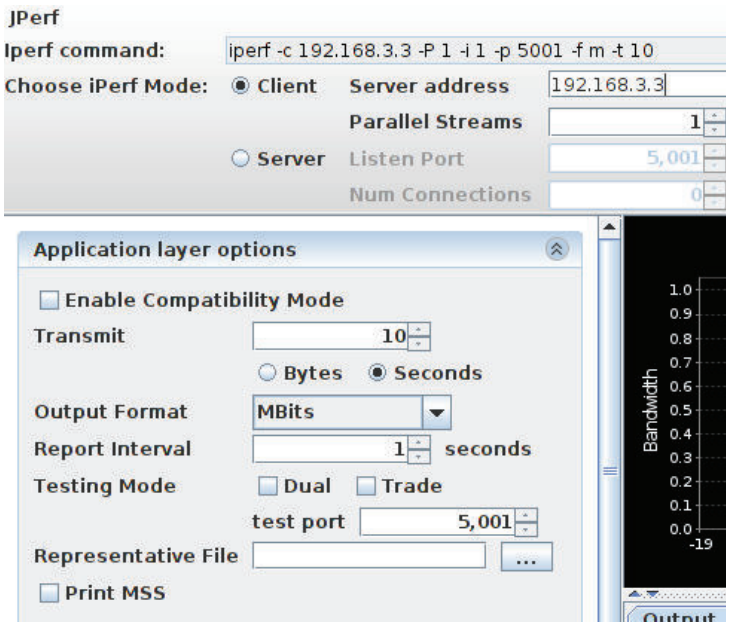


Figura 3.27 Parámetros para envío de paquetes TCP

De acuerdo a las mediciones obtenidas, se determina que el enlace es aproximadamente de 1Gbps o Gigabit Ethernet en cada puerto del conmutador; el resultado se muestra en la Figura 3.26.

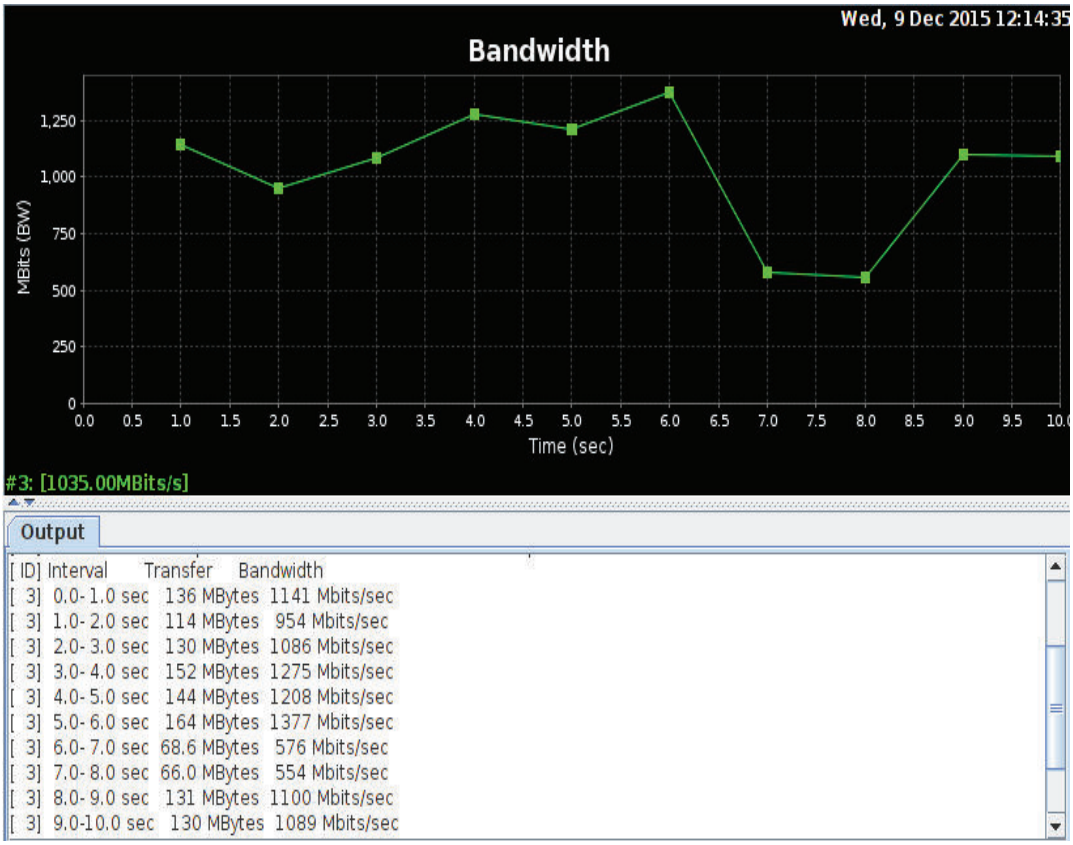


Figura 3.28 Resultados de mediciones de ancho de banda

Para verificar si la solución de calidad de servicio es funcional, se debe saturar los enlaces, en este caso se reducirá el ancho de banda disponible de 1Gbps a 30 Mbps, para que dicha saturación se produzca con menos información, esta configuración se la realiza dentro de los *switches* al momento de la creación de las colas, este proceso se describirá posteriormente.

Al obtener el ancho de banda disponible, se puede establecer los valores limitantes para el tráfico VoIP.

Para determinar el ancho de banda que una llamada requiere se utiliza una herramienta de Elastix en línea [96].

Se establecen los parámetros para el cálculo: el protocolo a utilizar, las llamadas simultáneas y la codificación, esto entrega los valores necesarios para satisfacer la transmisión de datos.

En la Figura 3.27 se visualiza el cálculo. De acuerdo a los valores obtenidos, se procede con la creación de las reglas de encolamiento y priorización en cada puerto, para ello, se utilizan los comandos descritos en la Figura 3.28.

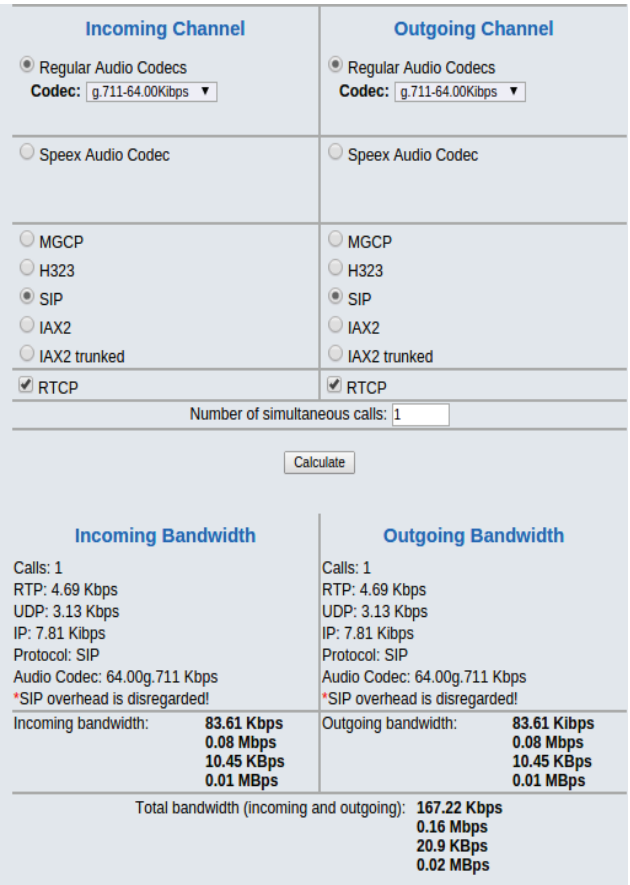


Figura 3.29 Medición de ancho de banda de una llamada VoIP

```
1  ovs-vsctl set port vnet2 qos=@newqos -- --
2  id=@newqos create qos type=linux-htb other-config:max-rate=30000000
3  queues:0=@q0 queues:1=@q1 -- --
4  id=@q0 create queue other-config:max-rate=5000000 -- --
5  id=@q1 create queue other-config:min-rate=5000000 other-config:max-rate=30000000
```

Figura 3.30 Creación de colas en OVS

Para el ejemplo, en la línea número uno se configura al puerto vnet2 para que acepte la creación de colas y en la línea dos, se establece el ancho de banda máximo que permitirá dicho puerto, el cual como se mencionó anteriormente debe ser reducido de 1Gbps a 30Mbps.

En la línea tres, se crean dos colas con ID 0 y 1 respectivamente, en la línea cuatro, se visualiza la configuración para la cola 1, se establece un ancho de banda de 5 Mbps mínimo y un valor máximo de 30 Mbps, esta cola recibirá el tráfico de voz. Finalmente, en la línea cinco, para el resto del tráfico se configura un ancho de banda de 5 Mbps máximo sobre la cola con ID 0, este valor se escogió para que la saturación sea alta y los valores de saturación se aprecien

con facilidad. OpenFlow y Open vSwitch permiten asociar el puerto destino con el ID de la cola, es por ello que en cada puerto se tendrán dos colas con id “1” y “0” respectivamente, de esta manera el reenvío de paquetes se facilita notablemente y se descarta la posibilidad de inconsistencias. En la Figura 3.29 se aprecia el comando para verificar la creación de las colas y la configuración de cuatro de ellas en dos puertos diferentes del *switch* br0.

```
root@tesis:/home/tesis# ovs-vsctl list queue
_uuid               : c16262b9-a303-4b7d-9bec-f438640d92c3
dscp                : []
external_ids        : {}
other_config        : {max-rate="5000000"}

_uuid               : 44821ba9-5164-4293-bd99-fdc45e40f996
dscp                : []
external_ids        : {}
other_config        : {max-rate="5000000"}

_uuid               : df676172-1226-4f2e-8061-02d61b4035e6
dscp                : []
external_ids        : {}
other_config        : {max-rate="15000000", min-rate="5000000"}

_uuid               : 5c6f40c3-a806-476c-b167-c52147d5726e
dscp                : []
external_ids        : {}
other_config        : {max-rate="15000000", min-rate="5000000"}
```

Figura 3.31 Verificación de colas en el *switch*

Se procede con la medición del ancho de banda en los puertos del *switch*, los resultados arrojados por la herramienta son satisfactorios y se puede comprobar que la cola “0” tiene un límite máximo de 5 Mbps, en la Figura 3.30 se muestra la medición respectiva.

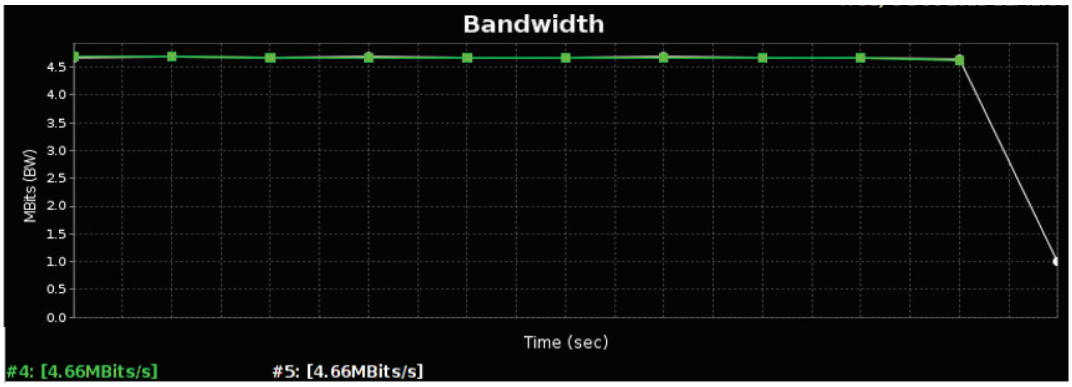


Figura 3.32 Resultados de ancho de banda

De igual manera se procede con las mediciones de *jitter* el detalle se aprecia en la Figura 3.31.

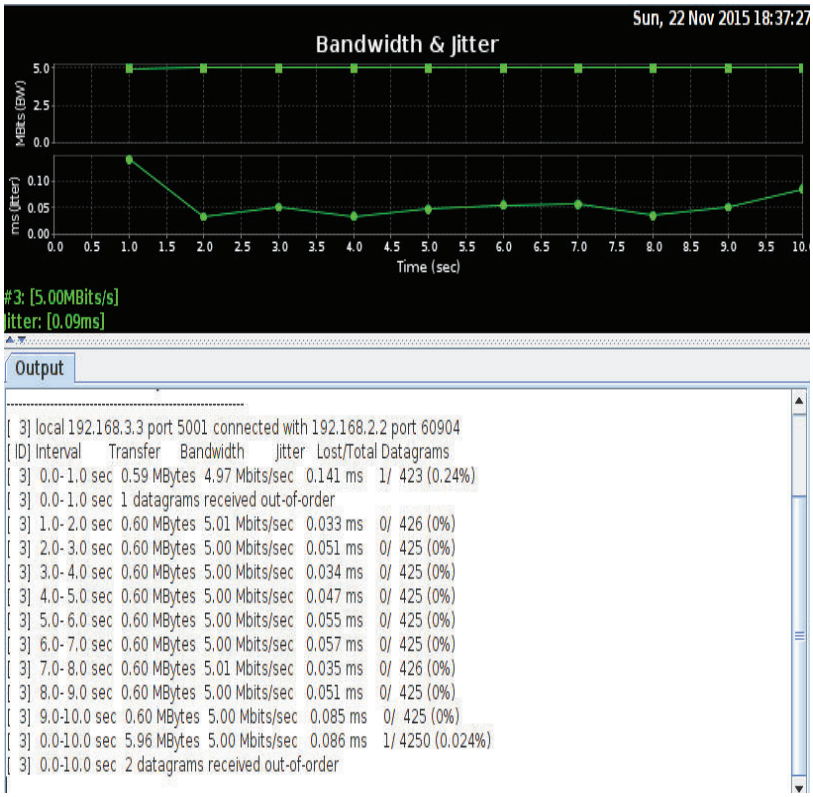


Figura 3.33 *Jitter* en cola 0

Adicionalmente se envía paquetes ICMP a través de la cola para determinar su latencia, el resultado se aprecia en la Figura 3.32.

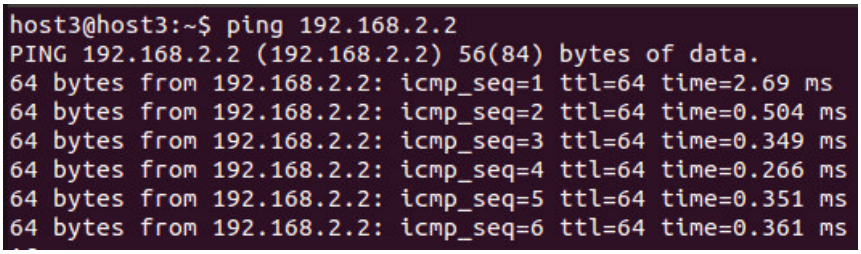


Figura 3.34 Paquetes ICMP en la cola 0

Dentro del *switch*, se puede apreciar la cantidad de paquetes que fueron procesados en cada cola con el comando ovs-ofctl –O OpenFlow13 queue-stats CONMUTADOR PUERTO. En la Figura 3.33 se aprecian los paquetes que se procesaron por cola para el puerto vnet3.

```
root@tesis:/home/tesis# ovs-ofctl -O Openflow13 queue-stats br0 vnet3
OFPST_QUEUE reply (OF1.3) (xid=0x4): 2 queues
  port 3 queue 0: bytes=19631086, pkts=239142, errors=0, duration=4294947350.359
0967296s
  port 3 queue 1: bytes=0, pkts=0, errors=0, duration=4294947350.3590967296s
```

Figura 3.35 Paquetes en cola 0

Como se esperaba, todos los paquetes utilizados para las mediciones se enviaron por la cola por defecto (q0). Si se realiza una llamada telefónica se visualiza que los paquetes se envían a través de la cola 1 como se aprecia en la Figura 3.34.

```
root@tesis:/home/tesis# ovs-ofctl -O Openflow13 queue-stats br0 vnet2
OFPST_QUEUE reply (OF1.3) (xid=0x4): 2 queues
  port 2 queue 0: bytes=30643714, pkts=27276, errors=333, duration=4294935516.34
65967296s
  port 2 queue 1: bytes=1792, pkts=3, errors=0, duration=4294935516.3465967296s
```

Figura 3.36 Paquetes en cola 1

De acuerdo a lo comentado, las pruebas de encolamiento son satisfactorias y se obtiene los resultados esperados con las configuraciones establecidas. El siguiente paso es saturar la red, para ello se emitirá video *streaming* al mismo tiempo que se realiza una llamada telefónica y se enviará paquetes TCP a través de un generador de tráfico para la saturación.

3.7.1 PRUEBAS DE SATURACIÓN

Para generar tráfico, se utiliza la herramienta Mausezahn, es un generador de tráfico escrito en lenguaje de programación C que permite enviar casi todos los tipos de paquetes posibles. Se utiliza principalmente para probar VoIP o redes de multidifusión, también para las auditorías de seguridad para verificar si los sistemas son robustos ante ataques [87].

La instalación del generador es sencilla y se lo obtiene del repositorio de Ubuntu con el comando sudo apt-get install mz. Existen muchas posibilidades para la generación de diferentes tramas con diferentes características dentro de la bibliografía [88] se encuentra una guía detallada.

En el Proyecto se genera un ataque TCP sincrónico a un determinado *host* con el comando mz eth0 -A rand -B HOST DESTINO -c 0 -t tcp "dp=1-1023, flags=syn", el cual indica que se envíe a través de la interfaz eth0, con una dirección IP

origen aleatoria a una dirección IP destino específica y con un puerto destino aleatorio entre 1 y 1023, tramas TCP de forma continua.

Para la transmisión de video, se utiliza ffmpeg y su servidor ffserver, que se obtiene del repositorio de Ubuntu con el comando apt-get install ffmpeg. Esta herramienta permite levantar un servidor de *streaming* para transmisión de audio y video mediante el protocolo TCP.

Antes de poder transmitir, se deben realizar algunas configuraciones, para ello se edita el archivo ffserver.conf, que se encuentra dentro del directorio /etc/, donde se incluye el número de clientes para los que estará disponible el video, la dirección IP y puerto del servidor, en este caso se utiliza como servidor de *streaming* al *host* número cuatro que tiene por dirección IP 192.168.3.3 y se transmite por el puerto 8090, adicionalmente se especifica el formato de video a transmitir, si se desea enviar audio o no, la resolución, número de tramas por segundo, entre otros, el detalle se muestra en la Figura 3.35.

Se transmitirá la captura de la cámara web, para enviar el *streaming* se utiliza el comando que se muestra en la Figura 3.37.

ffserver.conf

```
Port 8090
BindAddress 192.168.3.3
MaxHTTPConnections 10
MaxClients 3
MaxBandwidth 1000000
CustomLog -
<Feed feed1.ffm>
File /tmp/feed1.ffm
File MaxSize 1G
ACL allow 192.168.3.3
</Feed>
<Stream tesis.webm>
Feed feed1.ffm
Format webm
NoAudio
VideoCodec libvpx
VideoSize 650x500
VideoFrameRate 10
AVOptionVideo flags +global_header
AVOptionVideo cpu-used 0
   AVOptionVideo qmin 10
   AVOptionVideo qmax 42
   AVOptionVideo quality good
   AVOptionAudio flags +global_header
PreRoll 15
StartSendOnKey
```

Figura 3.37 Configuraciones de ffserver

```
root@host4:/home/host4# ffmpeg -i /dev/video0 http://192.168.3.3:8090/feed1.ffm
```

Figura 3.38 Comando para transmisión de video

Desde un *host* diferente mediante un navegador web, se accede a la transmisión del video ingresando la URL del servidor como se muestra en la Figura 3.37.

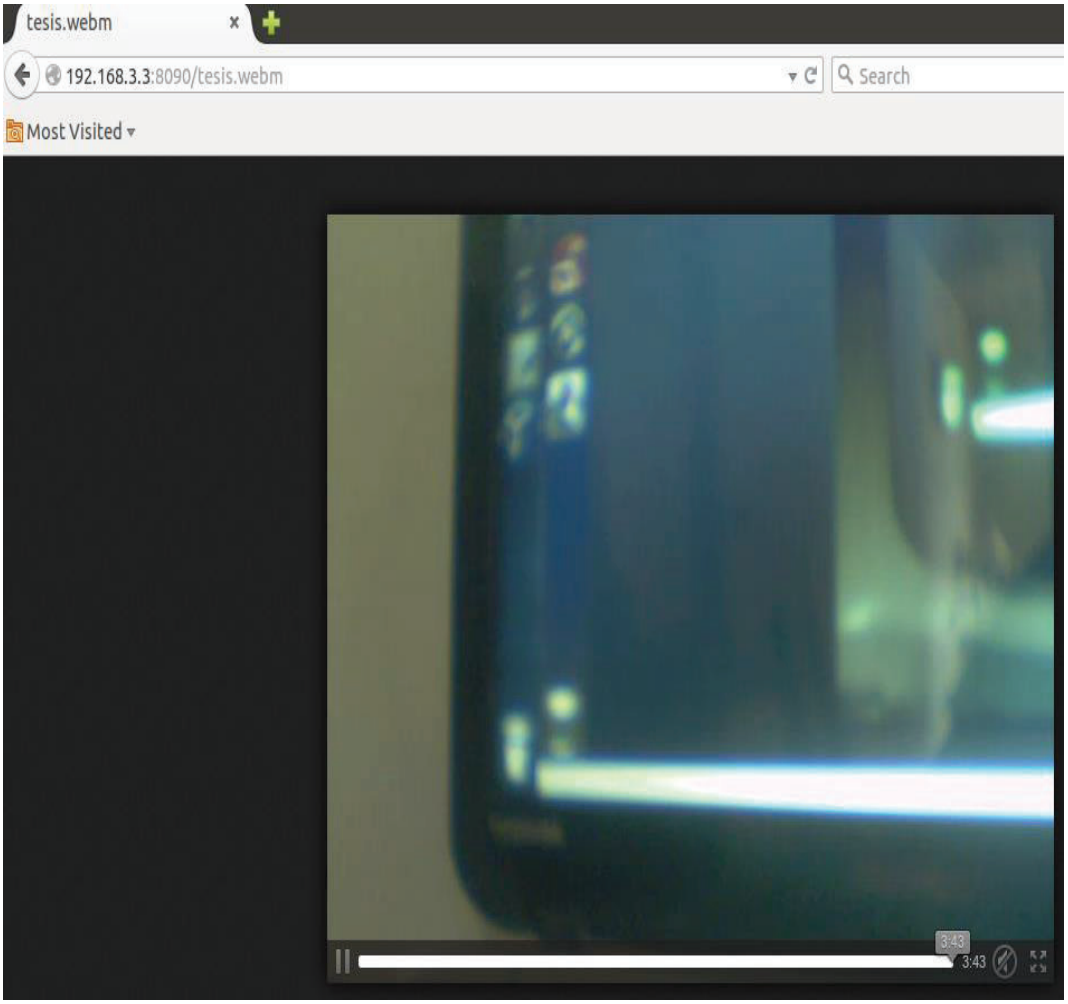


Figura 3.39 Video *streaming*

De acuerdo a lo indicado, al momento que el video se emita, se realiza una llamada telefónica y se satura la red con el generador de tráfico, en este caso la llamada se realiza entre el *host* uno con dirección IP 192.168.2.2 hacia el *host* tres con dirección IP 192.168.3.2, el *host* uno de igual manera se encuentra enlazado al video *streaming* emitido por el *host* número cuatro y finalmente el *host* dos emite tráfico TCP con el generador hacia el *host* cuatro.

Se procede con las mediciones respectivas, utilizando iperf. Las mediciones de *jitter* en el enlace entre los switches, reflejan un claro aumento, los detalles se muestran en la Figura 3.38.

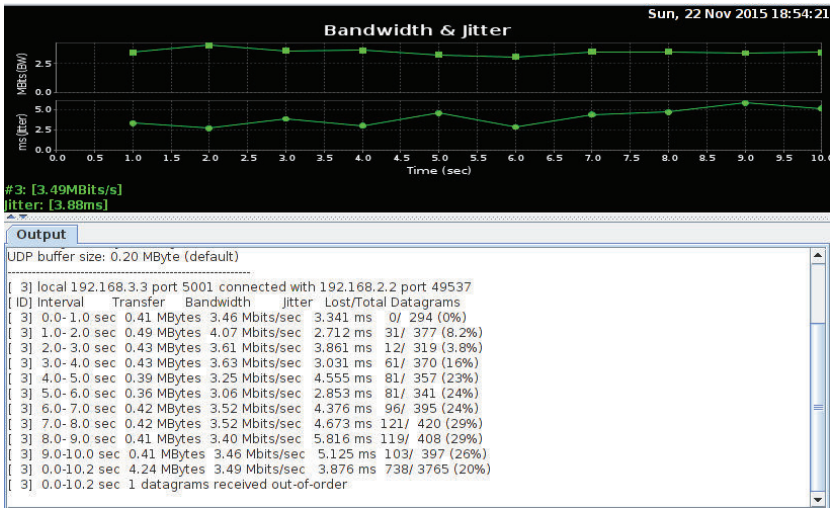


Figura 3.40 *Jitter* con saturación

Adicionalmente, se envían mensajes ICMP para determinar cuánto demora la respuesta de un *host* ante la solicitud de otro, de igual manera se refleja un incremento, el resultado se aprecia en la Figura 3.39.

```
host3@host3:~$ ping 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=64 time=10.8 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=64 time=9.10 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=64 time=2.16 ms
64 bytes from 192.168.2.2: icmp_seq=4 ttl=64 time=0.363 ms
64 bytes from 192.168.2.2: icmp_seq=5 ttl=64 time=2.59 ms
64 bytes from 192.168.2.2: icmp_seq=6 ttl=64 time=0.371 ms
64 bytes from 192.168.2.2: icmp_seq=7 ttl=64 time=0.316 ms
64 bytes from 192.168.2.2: icmp_seq=8 ttl=64 time=1.48 ms
64 bytes from 192.168.2.2: icmp_seq=9 ttl=64 time=0.306 ms
64 bytes from 192.168.2.2: icmp_seq=10 ttl=64 time=0.673 ms
```

Figura 3.41 Ping con saturación

Al verificar la transmisión de video, se visualiza que el *streaming* ya no es fluido si no que requiere la carga del buffer para su posterior visualización lo que muestra que se tiene un encolamiento en el enlace, como se aprecia en la Figura 3.40.

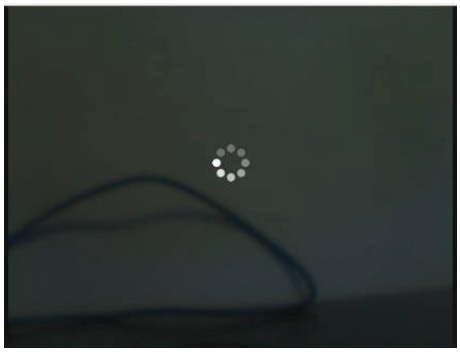


Figura 3.42 *Streaming* con saturación

Con respecto a la calidad de la llamada, se observa que los valores de los índices son constantes a pesar de la carga de tráfico de la red. Dentro de Linphone existe un medidor de la calidad de la llamada y el mismo se mantienen constante en el valor más alto, este se aprecia en la Figura 3.41; estos resultados indican que la solución de calidad de servicio está trabajando de la forma esperada.

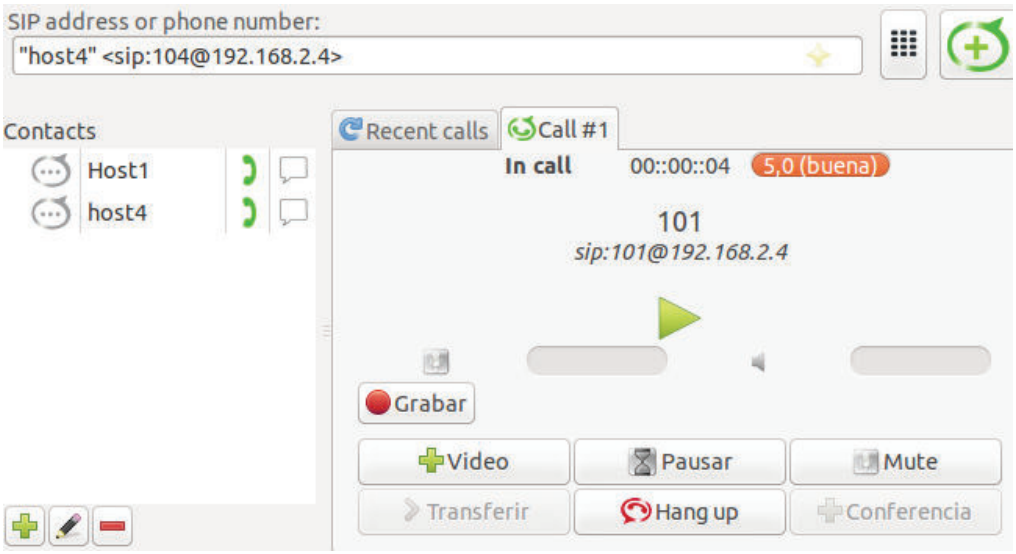


Figura 3.43 Calidad de la llamada telefónica

Cada una de las mediciones se las realizó en diez ocasiones y se obtuvo el valor promedio, el detalle de las mediciones con el enlace libre sin colas, el enlace libre con colas y, finalmente, con el enlace saturado se visualizan en la tabla 3.1.

	Red sin colas	Red con colas	Red con saturación
Latencia	N/A	0.75 ms	2,8 ms
Jitter	N/A	0.0534 ms	4.42 ms
Ancho de banda	1 Gbps	4.8 Mbps	4.6 Mbps

Tabla 3.1 Resumen de resultados

3.8 PRESUPUESTO REFERENCIAL DEL PROTOTIPO

El presupuesto referencial se presenta en la tabla 3.2

CANTIDAD	ITEM	VALOR	TOTAL
1	Computador	$852.89	$852.89
100 (**)	Instalación y configuración del controlador y dispositivos de red	$20	$2000
100 (**)	Desarrollo del módulo	$20	$2000
TOTAL			$4852.89

(**) Número de horas empleadas en capacitación, desarrollo y pruebas

Tabla 3.2 Presupuesto referencial

El presupuesto para el número de horas empleadas en capacitación, desarrollo y pruebas se ha basado al costo por hora que se paga a un Ingeniero en Electrónica y Redes de Información junior en el mercado actual y el equipo de acuerdo a la factura del Anexo H.

CAPÍTULO IV
CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- Al finalizar el proyecto, se obtiene la implementación, de la SDN con todos sus componentes y de la aplicación que brinda calidad de servicio, que mediante la generación de reglas de flujo y clasificación del tráfico demuestra que sobre las Redes Definidas por Software se puede implementar cualquier tipo de aplicación de las redes tradicionales, mostrando su flexibilidad y funcionalidad.

- Para la interconexión de los dispositivos se utilizó *switches* basados en software implementados con la herramienta Open vSwitch*,* es importante aclarar que este software permite la creación de las reglas de flujo y posterior encolamiento con comandos propios de configuración, semejantes a los dispositivos de red de las redes tradicionales, sin embargo, el objetivo del Proyecto es la automatización de la inserción de las reglas de flujo en el *switch*, demostrando la separación del plano de datos y del plano de control, que caracteriza a una Red Definida por Software.

- Es importante mencionar que actualmente Floodlight cuenta con un módulo para calidad de servicio que trabaja sobre una versión anterior del controlador y maneja los flujos de acuerdo a políticas preestablecidas mientras tanto que el módulo desarrollado en el Proyecto, realiza el direccionamiento de paquetes de forma automática, sin conocer el origen de la información a enviar, discrimina la información e inserta las reglas de flujo en los *switches* los mismos que procesan la información en dos colas con mayor y menor prioridad respectivamente, brindando así la calidad de servicio requerida.

- Como parte del desarrollo de la aplicación, a los paquetes que cumplen con las condiciones establecidas por el *match,* se les asigna determinadas acciones que son efectuadas por los *switches,* una de ellas es el cambio del parámetro DSCP al valor más alto permitido en cada uno de los paquetes que se desea priorizar, este parámetro, no es tomado en cuenta al momento de la saturación o reenvío de la información en la SDN establecida, sin embargo, se lo configura para complementar la solución de calidad de servicio, pensando en la posibilidad que dicho paquete viaje fuera de la red, hacia un dispositivo capaz de verificar este campo, analizando el segundo octeto de la cabecera IP y comparándolo con una tabla de priorización preestablecida, de acuerdo al valor encontrado el paquete será enrutado con una mayor o menor prioridad.

- El costo aproximado de la implementación del Proyecto es de $4852.59, por lo que se concluye y se demuestra que con un presupuesto bajo se puede brindar una solución de calidad de servicio para una topología de red pequeña utilizando Redes Definidas por Software. Si se desea aumentar el tamaño de la red, se debe considerar el costo de las horas de desarrollo que requeriría la modificación del código, para determinar cada uno de los nuevos enlaces, creación de nuevas reglas de flujo, entre otros.

- La estructura modular de Floodlight, permite crear aplicaciones independientes que cumplan diversas funciones, y si es necesaria su interacción con otros módulos ya creados, solo se los instancia y se accede a sus características y funcionalidades, lo que permite un ambiente flexible al momento del desarrollo.

4.2 RECOMENDACIONES

- Dentro de la Red Definida por Software, se utilizó una estructura totalmente virtualizada y su funcionamiento fue el adecuado, por lo que se recomienda a los desarrolladores, se utilice la virtualización como una

alternativa para la realización de pruebas para el desarrollo de aplicaciones.

- El Proyecto contempla una estructura de dos *switches* en un ambiente virtualizado, por lo que se recomienda la extensión del estudio con más dispositivos, la priorización para distintos protocolos de acuerdo a su importancia dentro de la red y la utilización de equipos físicos para validar la solución en un ambiente acorde a una infraestructura real dentro de un entorno empresarial.

- Para los programadores que deseen establecer una red SDN virtualizada, se recomienda seguir los pasos oficiales de instalación de las diferentes herramientas utilizadas durante el desarrollo de este Proyecto, ya que en la web existen diversos tutoriales que generaron complicaciones durante la implementación en cuanto a incompatibilidades y omisión de librerías.

- Para el establecimiento de las llamadas telefónicas se utilizó Elastix como servidor de telefonía y Linphone para los terminales, utilizando el protocolo SIP para el establecimiento de las llamadas, sin embargo, estos programas son compatibles con otros protocolos como por ejemplo IAX2, por lo que se recomienda realizar pruebas con otros protocolos de transmisión de audio y video.

- Se recomienda la creación de un repositorio documental que abarque la teoría de las SDN y sus componentes, ya que en la web, la información es diversa y en muchos casos no es clara, principalmente sobre Floodlight y Open vSwitch, por lo que se tuvo que recurrir a los documentos originales de los desarrolladores que no siempre se adaptan a la solución que se desea plantear.

- Si se desea que el parámetro DSCP sea considerado para la priorización o encolamiento dentro de una SDN, se recomienda incluirlo como parte del *match* que discrimina los paquetes, tomando en cuenta que si a lo

largo del trayecto este es modificado, el *switch* con la regla de priorización lo omitirá ya que el *match* no coincidiría.

- Open vSwicth y las máquinas virtuales tienen una relación directa con el *kernel* de Linux, por lo que se recomienda que una vez instalados, no se actualice ni se modifique el *kernel* ya que no existe la posibilidad de una actualización para estas herramientas y será necesaria una reinstalación, perdiendo las configuraciones realizadas.

- Para despejar inquietudes sobre el controlador Floodlight se recomienda, consultar al grupo abierto de desarrolladores https://groups.google.com /a/openflowhub.org/forum/#!forum/floodlight-dev, el mismo que fue fundamental durante el desarrollo del Proyecto.

- Para realizar llamadas telefónicas se utilizó como servidor de telefonía a Elastix, sin embargo, solo se configuró los parámetros principales para poder registrar una extensión, por lo que se recomienda el análisis e implementación de las diversas opciones que este servidor ofrece.

- Para la transmisión de video *streaming* se envía el video captado por una cámara web para que una máquina virtual creada en VirtualBox permita el acceso a un periférico conectado por USB, es necesario obtener un paquete de extensión, por lo que se recomienda su instalación, así como la utilización del software en su última versión y su ejecución como administrador dentro de la distribución de Linux, caso contrario no se podrá hacer uso de un dispositivo externo.

- Floodlight en su versión actual es compatible con el protocolo OpenFlow en sus versiones 1.1, 1.2 y 1.3, sin embargo, Open vSwitch, maneja todas las versiones del protocolo y por defecto establece la más reciente que es la 1.4, por lo que se recomienda siempre al momento de crear un *switch* configurar una versión del OpenFlow compatible con Floodlight ya que de

lo contrario no se podrá realizar el intercambio de mensajes entre el controlador y los dispositivos de red.

- Para el Proyecto no fue necesaria la comunicación fuera del computador físico, sin embargo, si se desea enviar información fuera de este, se recomienda la conexión del *switch* con la interfaz de salida (eth0 generalmente), lo que crea un enlace *trunk* entre la tarjeta de red física y el *switch*, permitiendo por ejemplo tener salida hacia Internet o la configuración de las direcciones IP por medio de un servidor DHCP.

- Si se desea virtualizar varios computadores como es el caso del presente Proyecto, es recomendable que el equipo físico tenga las características adecuadas tanto en memoria, procesamiento y almacenamiento para poder trabajar de una forma adecuada.

- Durante el desarrollo del proyecto se realizaron varias configuraciones a través de los terminales de las distribuciones Linux utilizadas, estos procesos son repetitivos y se deben realizar cada vez que se desea levantar el ambiente, por lo que se recomienda siempre crear *scripts* para que la ejecución de tareas repetitivas se realice rápidamente. Un ejemplo se muestra en el Anexo K.

- Para el enlace entre los *switches* existe una alternativa de comunicación utilizando interfaces del tipo *patch,* estas son configuradas en cada *switch* asignándole la referencia de su par correspondiente (interfaz destino) con el que interactuará, sin embargo, esta opción no permite la creación de colas en el enlace debido a que un *patch port* no existe dentro del *kernel* de Linux y solo existe a nivel de Open vSwitch, por lo que se recomienda la utilización de interfaces veth.

REFERENCIAS BIBLIOGRÁFICAS

[1] OPEN NETWORKING FOUNDATION, “Software-Defined Networking: The New Norm for Networks”, https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf *(Consultado el 1 de Noviembre 2014).*

[2] OPEN NETWORKING FOUNDATION, “SDN architecture”, https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf *(Consultado el 2 de Noviembre 2014).*

[3] MARGARET ROUSE,”SDN-Data” Plane, http://searchsdn.techtarget.com/definition/data-plane-DP*(Consultado el 10 de Noviembre 2014).*

[4] PC MAGAZINE,”Definition of: API”, http://www.pcmag.com/encyclopedia/term/37856/api *(Consultado el 10 de Noviembre 2014).*

[5] OPEN NETWORKING FOUNDATION, ‘’SDN Workshops‘’, https://www.opennetworking.org/sdn-resources/sdn-library/whitepapers *(Consultado el 12 de Noviembre 2014).*

[6] SFLOW, ‘’ Software defined networking’’, http://blog.sflow.com/2012/05/software-defined-networking.html *(Consultado el 13 de Noviembre 2014).*

[7] TEEMU KOPONEN,”Structure and desing of software defined networks”, http://netseminar.stanford.edu/seminars/14_03_13.pdf *(Consultado el 13 de Noviembre 2014).*

[8] BRIAN MARSHALL/STEPHEN PATEL,’’ Software defined networks ’’, http://big*switch*.com/sites/default/files/sdn_resources/isi-research.pdf *(Consultado el 13 de Noviembre 2014).*

[9] BIG *SWITCH* NETWORKS, ‘’The Open SDN Architecture’’, http://big*switch*.com/sites/default/files/sdn_overview.pdf (*Consultado el 15 de Noviembre 2014).*

[10] BIG *SWITCH* NETWORKS, ‘’Big Network Controller’’, http://big*switch*.com/products/SDN-Controller (*Consultado el 15 de Noviembre 2014).*

[11] FLOWGRAMMABLE, “Flowgrammable stack”, http://flowgrammable.org/sdn/flowgrammable-stack/ (*Consultado el 21 de Noviembre 2014).*

[12] FLOWGRAMMABLE, “Packet out”, http://flowgrammable.org/sdn/openflow/message-layer/packetout/ (*Consultado el 21 de Noviembre 2014).*

[13] OPEN NETWORKING FOUNDATION, “SDN Architecture Overview”, https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf (*Consultado el 1 de Diciembre 2014).*

[14] HSR, “Software Defined Network”, http://eprints.hsr.ch/378/1/Software_Defined_Network.pdf (*Consultado el 1 de Diciembre 2014).*

[15] THE NEW STACK, “Defining Software Defined Network”, http://thenewstack.io/defining-software-defined-networking-part-1/ (*Consultado el 1 de Diciembre 2014).*

[16] PROJECT FLOODLIGHT, “Floodlight Documentation”, http://docs.projectfloodlight.org/display/floodlightcontroller/Floodlight+Documentation, *(Consultado el 11 de Diciembre de 2014).*

[17] REAL ACADEMIA DE INGENIERÍA, “Calidad de servicio”, http://diccionario.raing.es/es/lema/calidad-de-servicio-0 (*Consultado 21 de Diciembre de 2014)*

[18] TECHTARGET, “Floodlight primer: An *OpenFlow* controller”, http://searchsdn.techtarget.com/tip/Floodlight-primer-An-*OpenFlow*-controller (*Consultado 25 de Diciembre de 2014)*

[19] GITHUB, “Floodlight, ”https://github.com/floodlight/floodlight (*Consultado 25 de Diciembre de 2014)*

[20] FLOODLIGHT PROJECT, “The controller,” http://docs.projectfloodlight.org/display/floodlightcontroller/The+Controller (*Consultado 25 de Diciembre de 2014)*

[21] SDN CENTRAL, “What is a floodlight controller?”

https://www.sdncentral.com/resources/sdn/sdn-controllers/open-source-sdn-controllers/what-is-floodlight-controller/ (*Consultado 25 de Diciembre de 2014)*

[22] LINKEDIN, “Big *Switch* Networks”, https://www.linkedin.com/company/big-*switch*-networks (*Consultado 26 de Diciembre de 2014)*

[23] THE NEW STACK, ‘’Architecture’’, http://thenewstack.io/sdn-series-part-v-floodlight/ (*Consultado 27 de Diciembre de 2014)*

[24] *REAL ACADEMIA DE INGENIERÍA, “Computación en la nube”,* http://diccionario.raing.es/es/lema/computación-en-la-nube (*Consultado 27 de Diciembre de 2014)*

[25] CHILEAGIL*, “Jithon: Desarrollo ágil en la plataforma Java”,* http://chileagil.cl/2009/10/12/jython-desarrollo-agil-en-la-plataforma-java-parte-1/*,* (*Consultado 28 de Diciembre de 2014)*

[26] OPEN NETWORKING FOUNDATION, ‘’*OpenFlow switch* especifications’’, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-*switch*-v1.3.4.pdf , (*Consultado 28 de Diciembre de 2014)*

[27] COMPUTERWEEKLY.COM, ‘’ The history of *OpenFlow*’’, http://www.computerweekly.com/feature/The-history-of-*OpenFlow*, (*Consultado 28 de Diciembre de 2014)*

[28] STANFORD UNIVERSITY, ‘’Ethane: A security management architecture’’, http://yuba.stanford.edu/ethane/ (*Consultado 28 de Diciembre de 2014)*

[29] CRUNCHBASE, ‘’Nicira Networks’’, http://www.crunchbase.com/organization/nicira (*Consultado 28 de Diciembre de 2014)*

[30] TECNOLOGÍAS EMERGENTES, ‘’Definición de tecnologías emergentes’’, http://tecmethepelonex.blogspot.com/2008/08/definicion-tecnologias-emergentes.html, (*Consultado 29 de Diciembre de 2014)*

[31] SREENIVAS MAKAM’S BLOG, ‘’*OpenFlow* hisrotry and some hands on’’,http://sreeninet.wordpress.com/2014/01/16/openflow-overview/ (*Consultado 31 de Diciembre de 2014)*

[32] SLIDESHARE, ‘’Introduction to *OpenFlow*’’, http://www.slideshare.net/rjain51/m-14ofl (*Consultado 31 de Diciembre de 2014)*

[33] NETSOCKET, ‘’Centralized vs. Distributed vs. Hybrid SDN – Which is the Best Approach for Today’s Networks?’’,http://www.virtualnetwork.com/blog/2013/08/centralized-vs-distributed-vs-hybrid-sdn-which-is-the-best-approach-for-todays-networks/ (Consultado 31 de Diciembre de 2014)

[34] ONRC,’’Modern SDN Stack’’,http://onrc.berkeley.edu/research_modern_sdn_stack.html, (Consultado 31 de Diciembre de 2014)

[35] SLIDESHARE, ‘’SDN a Brief introduction’’,http://es.slideshare.net/jasonhoutw/software-defined-network, (Consultado 31 de Diciembre de 2014)

[36] NOX, ‘’Ábout NOX’’http://www.noxrepo.org/nox/about-nox/, (Consultado 31 de Diciembre de 2014)

[37] TECHTARGET,’’Hypervisor’’, http://searchservervirtualization.techtarget.com/definition/hypervisor (Consultado 31 de Diciembre de 2014)

[38] THOMAS D. NADEAU & KEN GRAY, ‘’SDN Software defined networks’’, editorial O’REILLY, Version PDF. (Consultado 31 de Diciembre de 2014)

[39] CISCO,”What is Netconf” , http://www.tail-f.com/education/what-is-netconf/ (Consultado 6 de Enero de 2015)

[40] TOM’S IT PRO, ‘’ Pica8: First to Adopt *OpenFlow* 1.4; Why Isn't Anyone Else?’’, http://www.tomsitpro.com/articles/pica8-openflow-1.4-sdn-*switches*,1-1927.html (Consultado 6 de Enero de 2015)

[41] OPEN NETWORKING FOUNDATION, “*OpenFlow Switch* Specification”, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-*switch*-v1.3.4.pdf (Consultado 6 de Enero de 2015).

[42] WEBOPEDIA, “LLDP”, http://www.webopedia.com/TERM/L/LLDP.html (Consultado 6 de Enero de 2015).

[43] SW*HOST*ING, ”Transport Layer Security (TLS): qué es y cómo funciona”, http://www.sw*host*ing.com/blog/transport-layer-security-tls-que-es-y-como-funciona/ (Consultado 6 de Enero de 2015).

[44] CTXDOM.COM, ”Qué es multiphat?”, http://www.ctxdom.com/index.php?option=com_content&view=article&id=323:que-es-el-multipath&catid=31:general&Itemid=72 (Consultado 6 de Enero de 2015).

[45] IVAN ESTEVEZ, ¿Qué es multicast y para qué sirve?”, http://www.somosbinarios.es/que-es-multicast/ (Consultado 6 de Enero de 2015).

[46] PIYUSH KUMAR, “What is bitmasking? What kind of problems can be solved using it?”,https://www.quora.com/What-is-bitmasking-What-kind-of-problems-can-be-solved-using-it (Consultado 10 de Enero de 2015).

[47] COMPUTER HOPE, “Data path”, http://www.computerhope.com/jargon/d/datapath.htm (Consultado 15 de Enero de 2015).

[48] MARGARET ROUSE, ”REST definition”,http://searchsoa.techtarget.com/definition/REST (Consultado 15 de Enero de 2015).

[49] IBM, “5 cosas que usted no sabía acerca de... java.util.concurrent, Parte 2”, http://www.ibm.com/developerworks/ssa/java/library/j-5things5/ (Consultado 15 de Enero de 2015).

[50] MARGARET ROUSE , “NoSQL definition”, http://searchdatamanagement.techtarget. com/definition/NoSQL-Not-Only-SQL Consultado 15 de Enero de 2015).

[51] OPENV*SWITCH*.ORG,” Extending Networking into the Virtualization Layer”, http://openv*switch*.org/papers/hotnets2009.pdf, (*Consultado el 22 de junio 2015*)

[52] *HOST*NAME, “¿Qué es APACHE y para qué sirve?”, http://clientes.*host*name.cl/knowledgebase.php?action=displayarticle&id=35 (*Consultado el 2 de julio 2015*)

[53] VIRTUALBOX.ES, “Características”, http://virtualbox.es/caracteristicas/ (*Consultado el 4 de julio 2015*)

[54] VIRTUALBOX.ORG," Chapter 1. First steps" , https://www.virtualbox.org/manual/ch01.html, (*Consultado el 14 de julio 2015*)

[55] WEBOPEDIA, "Hypervisor", http://www.webopedia.com/TERM/H/hypervisor.html (*Consultado el 18 de julio 2015*)

[56] ELASTIX, "Información del producto", http://www.elastix.org/index.php/es/informacion-del-producto/informacion.html (*Consultado el 18 de septiembre 2015*)

[57] FREEBSD, "Acerca de Freebsd",https://www.freebsd.org/es/about.html (*Consultado el 18 de septiembre 2015*)

[58] INFORMATICA HOY, "Qué son los programas softphones?" http://www.informatica-hoy.com.ar/voz-ip-voip/Que-son-los-programas-softphone.php, (*Consultado el 18 de septiembre 2015*)

[59] YAHOORESPUESTAS, "¿Que significa roaming en los celulares?", https://mx.answers.yahoo.com/question/index?qid=20081230111854AAChxc2 (Consultado el 18 de septiembre 2015)

[60] OPEN V*SWITCH*, "Production Quality, Multilayer Open Virtual *Switch*", http://openv*switch*.org/ (*Consultado el 18 de febrero 2015*)

[61] TE-YEN LIU "The Basic Introduction of Open v*Switch*", http://es.slideshare.net/teyenliu/the-basic-introduction-of-open-v*switch* (*Consultado el 18 de mayo 2015*)

[62] RALF SPENNEBERG "Virtual *switch*ing with Open v*Switch*", http://www.admin-magazine.com/CloudAge/Articles/Virtual-*switch*ing-with-Open-v*Switch* (*Consultado el 25 de mayo 2015*)

[63] DANNY KIM, "Install *OpenFlow* Network" (OpenVSwich, Floodlight, VirtualBox), http://dannykim.me/danny/openflow/57620?ckattempt=1 (*Consultado el 30 de mayo 2015)*

[64] DANNY KIM, "Install eclipse to create a floodlight module", http://dannykim.me/danny/openflow/57680(*Consultado el 30 de mayo 2015*)

[65] ROAN, "Openv*Switch* Debug$ Enviroment", http://roan.logdown.com/, (*Consultado el 20 de junio 2015*)

[66] GITHUB, “Open v*Switch*”, https://github.com/openv*switch*/ovs/blob/master/README.md (*Consultado el 22 de junio 2015*)

[67] OSCAR BELMONTE, “Introducción al lenguaje de programación Java”, http://www3.uji.es/~belfern/pdidoc/IX26/Documentos/introJava.pdf, (*Consultado el 18 de septiembre 2015*)

[68] JAVA, “Qué es la tecnología Java y para que la necesito?”, https://www.java.com/es/download/faq/whatis_java.xml (*Consultado el 19 de septiembre 2015*)

[69] INFOR.UVA.ES, ”Que es Java”, http://www.infor.uva.es/~jmrr/tgp/java/JAVA.html (*Consultado el 19 de septiembre 2015*)

[70] LENGUAJES DE PROGRAMACIÓN, “Programación java”, http://www.lenguajes-de-programacion.com/programacion-java.shtml (*Consultado el 25 de septiembre 2015*)

[71] WIKILIBROS, “Programación en java”, https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java/Caracter%C3%ADsticas_del_lenguaje(*Consultado el 25 de septiembre 2015*)

[72] GENBETA, “Eclipse IDE”, http://www.genbetadev.com/herramientas/eclipse-ide, (*Consultado el 2 de octubre 2015*)

[73] ECURED, “Eclipse, entorno de desarrollo integrado”,http://www.ecured.cu/index.php/Eclipse,_entorno_de_desarrollo_integrado (*Consultado el 2 de octubre 2015*)

[74] MARVIN GONZÁLEZ ,“IDE-Eclipse”,http://es.slideshare.net/MagaLasic/presentacion-eclipse-grupo-6 (*Consultado el 2 de octubre 2015*)

[75] IANA.ORG, “RTP-Parameters”,http://www.iana.org/assignments/rtp-parameters/rtp-parameters.xhtml (*Consultado el 12 de octubre 2015*)

[76] *NETWORKSORCERY, “RTP”, http://www.networksorcery.com/enp/protocol/rtp.htm (Consultado el 12 de octubre 2015)*

[77] GRUPOS DE TRABAJO DE REDIRIS,"SIP", https://www.rediris.es/mmedia/gt/gt2003_1/sip-gt2003.pdf (*Consultado el 15 de octubre 2015*)

[78] 3CX, "Informacion y Preguntas frecuentes acerca de SIP", http://www.3cx.es/voip-sip/sip-faq/ (*Consultado el 15 de octubre 2015*)

[79] IETF,"SIP" https://tools.ietf.org/html/rfc3261(*Consultado el 15 de octubre 2015*)

[80] NETWORKSTATIC, "Tutorial to Build a FloodLight SDN *OpenFlow* Controller Module", http://networkstatic.net/tutorial-to-build-a-floodlight-sdn-openflow-controller-module/ (*Consultado el 15 de octubre 2015*)

[81] UB.EDU.AR, "VISUAL AGE for JAVA" http://www.ub.edu.ar/catedras/ingenieria/ing_software/ubftecwwwdfd/java/java.htm (*Consultado el 18 de octubre 2015*)

[82] IETEF," About the IETF" https://www.ietf.org/about/ (*Consultado el 18 de octubre 2015*)

[83] ELASTIX, "Crear Extensiones en Elastix", http://elastixtech.com/curso-basico-de-elastix/crear-extensiones-en-elastix/ (*Consultado el 19 de octubre 2015*)

[84] APRENDE EN LÍNEA, "¿Qué es la URL de una página web?", http://aprendeenlinea.udea.edu.co/boa/contenidos.php/b9e5ef1074364b4b56c5f894e4d634c1/18/1/contenido/ (*Consultado el 19 de octubre 2015*)

[85] LINPHONE, "About", http://www.linphone.org/about.html (*Consultado el 28 de octubre 2015*)

[86] ELASTIX, "SIP en Elastix", http://elastixtech.com/sip-en-elastix/ (*Consultado el 1 de noviembre 2015*)

[87] PERIHEL.AT, "Mausezahn", http://www.perihel.at/sec/mz/ (*Consultado el 1 de noviembre 2015*)

[88] PERIHEAL.AT, "Mausezahn User's Guide", http://www.perihel.at/sec/mz/mzguide.html#tcp (Consultado el 1 de noviembre 2015)

[89] ADVANXER.COM, "Installing iperf on ubuntu", https://advanxer.com/blog/2012/06/installing-iperf-on-ubuntu/ (Consultado el 8 de noviembre 2015)

[90] CTAMLIBRE.COM, “Pasos para hacer streaming de video desde software libre”, http://ctamlibre.org/?q=video/node/366 (Consultado el 8 de noviembre 2015)

[91] GALLARDOWORK, “Cómo hacer Streaming de vídeo con VLC & Icecast”, http://gallardowork.blogspot.com/2015/02/como-hacer-streaming-de-video-con-vlc.html (Consultado el 8 de noviembre 2015)

[92] SEARCHUNIFIEDCOMMUNICATIONS, “jitter definition”, http://searchunifiedcommunications.techtarget.com/definition/jitter (Consultado el 8 de noviembre 2015)

[93] ECURED, “latencia Informática”, http://www.ecured.cu/Latencia_inform%C3%A1tica (Consultado el 8 de noviembre 2015)

[94] FFMPEG.ORG, “FFmpeg”, https://www.ffmpeg.org/ (Consultado el 18 de noviembre 2015)

[95] XJPERF, “JPerf 2.0.2”, https://code.google.com/p/xjperf/ (Consultado el 1 de diciembre 2015)

[96] ELASTIX, “Calcular Ancho de Banda en VoIP”, http://elastixtech.com/calcular-ancho-de-banda-en-voip/ (Consultado el 1 de diciembre 2015).

ANEXOS

ANEXO A: INSTALACIÓN DE OPEN VSWITCH 2.4.0 EN UBUNTU 14.04.3

ANEXO B: INSTALACIÓN DE FLOODLIGHT EN UBUNTU 14.04.3

ANEXO C: INSTALACIÓN DE VIRTUALBOX 5.0 EN UBUNTU 14.04.3

ANEXO D: INSTALACIÓN DE ELASTIX EN VIRTUALBOX 5.0

ANEXO E: PREPARACIONES DE ECLIPSE

ANEXO F: TABLA DSCP

ANEXO G: CÓDIGO DESARROLLADO

ANEXO H: INSTALACIÓN DE WIRESHARK H1

ANEXO I: PROFORMA DEL EQUIPO EMPLEADO EN EL PROYECTO

ANEXO J: VALORES INCLUIDOS EN EL ARCHIVO DE CONFIGURACIÓN

ANEXO K: SCRIPT PARA CREACIÓN DE LOS SWITCHES EN OPEN VSWITCH