Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Comunicación y Redes

Diseño y Desarrollo de una Herramienta de Medición de Desempeño para Controladores OpenFlow

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
por los Bachilleres
Daniel Alejandro Tovar Rodríguez
Alberto Alejandro Cavadia Porras
para optar al título de
Licenciado en Computación

Prof. Eric Gamess
Prof. Roger Bello

Caracas, Septiembre 2017

Universidad Central de Venezuela

Facultad de Ciencias
Escuela de Computación
Laboratorio de Comunicación y Redes

ACTA DE VEREDICTO

Quienes suscriben, miembros del jurado designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por los Bachilleres Daniel Tovar (C.I. V-20.603.552) y Alberto Cavadia (C.I. V-20.364.621), con el título "**Diseño y Desarrollo de una Herramienta de Medición de Desempeño para Controladores OpenFlow**", a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el nombrado trabajo por cada uno de los miembros del jurado, éste fijó el día lunes 02 de octubre de 2017, para que sus autores lo defendiera en forma pública, en la Sala de Internet II, Laboratorio LACORE, de la Escuela de Computación, mediante una exposición oral de su contenido, luego de la cual respondieron satisfactoriamente a las preguntas que les fueron formuladas por el jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela.

Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

Es de aclarar que el Profesor Eric Gamess se encuentra de permiso fuera del país. Por esta razón, el Profesor Robinson Rivas, director de la Escuela de Computación, firma el presente documento en su lugar.

En fe de lo cual se levanta el presente acta en Caracas a los 02 días del mes de octubre del año 2017, dejando constancia que el Profesor Roger Bello actuó como coordinador del jurado.

prof. Robinson Rivas M.
por el Prof. Eric Gamess
Tutor

Prof. Roger Bello
Tutor

Prof. Antonio Russoniello
Jurado Principal

Prof. Darwing Hernández
Jurado Principal

REPUBLICA DE VENEZUELA
Facultad de Ciencias
Escuela de Computación
Universidad Central - Caracas

Dedicatoria de Daniel Tovar

Para mis padres, Omaira Rodríguez y Miguel Tovar. Su apoyo incondicional me llevó a este logro.

Dedicatoria de Alberto Cavadia

Para Carlos De Pasquale y Benito Prado, porque ustedes también se merecen este título.

Agradecimientos de Daniel Tovar

Primero me gustaría dar las gracias a cada uno de los profesores de la Facultad de Ciencias. Dedicados, a pesar de las situaciones difíciles e injustas. Gracias a mi familia que siempre veló por mi educación, y me formó para ser el hombre que hoy está logrando esta meta.

Agradecimientos de Alberto Cavadia

Gracias a mi familia por haberme dado la libertad que todo estudiante universitario merece, y por ser al mismo tiempo la familia que cuida de no alejarte del sendero.

Gracias a todos mis amigos de la universidad, los que están y los que no, porque la universidad no hubiera sido la misma experiencia académica sin la competencia ni las risas. Fue una etapa genial, y este trabajo representa la promesa que compartimos al entrar.

Diseño y Desarrollo de una Herramienta de Medición de Desempeño para Controladores OpenFlow

Autores Daniel Alejandro Tovar Rodríguez y Alberto Alejandro Cavadia Porras.
Tutores: Prof. Eric Gamess y Prof. Roger Bello.

Resumen

El presente Trabajo Especial de Grado consiste en el desarrollo de una aplicación para la medición de desempeño de controladores SDN bajo OpenFlow 1.3, denominada OFC-Benchmark. La aplicación, permite la realización de pruebas basadas en mediciones de latencia, rendimiento, consumo de memoria RAM, consumo de CPU y utilización de las interfaces de red, sin olvidar el post procesamiento de los resultados en forma de gráficos.

Este trabajo, comienza con un proceso de investigación en torno a las Redes Definidas por Software (SDN), abarcando el estudio del protocolo OpenFlow y los controladores que hacen uso de este. Posterior a ello el estudio fue enfocado en herramientas de desempeño desarrolladas previamente, como Cbench, la cual fue utilizada junto al grupo de controladores estudiados como base de desarrollo. La construcción de esta herramienta, OFC-Benchmark, se basó en la metodología Ad Hoc incremental para los requerimientos funcionales planteados, metodología que combina elementos de un modelo secuencial con la filosofía iterativa de construcción de prototipos, donde cada iteración, se divide en las etapas de análisis, diseño, codificación y pruebas.

OFC-Benchmark está desarrollado en lenguaje C y solo trabaja bajo ambientes Unix. Fue diseñado para comunicarse usando tres protocolos fundamentales, TCP para la comunicación entre nodos, y OpenFlow 1.3 junto a SNMPv2 para la comunicación con el controlador. OFC-Benchmark ofrece funcionalidades interesantes y deseables en el ámbito de benchmarking, como es la habilidad de trabajar en un modelo distribuido, obtener indicadores de hardware y la presentación de resultados de manera gráfica.

Palabras clave: SDN, OpenFlow, Gnuplot, Controlador, Latencia, Rendimiento, SNMP, Lenguaje C.

El código fuente de OFC-Bechmark, se encuentra disponible para su descarga y modificación a través del siguiente link o código QR.

https://github.com/No6things/ofc-benchmark

Tabla de Contenido

Índice de Figuras

Índice de Tablas

1. Introducción

El crecimiento desmesurado de datos en la web, la virtualización de servicios y el cambio del patrón de tráfico entre usuarios y centro de datos ha llevado a una reevaluación del diseño actual para la administración de datos y dispositivos. El control de paquetes y flujos se ha vuelto más exigente y complicado, influyendo negativamente en la calidad del servicio.

Tendencias en auge como el manejo de masivas cantidades de datos en Big Data o Cloud Computing ha resultado en que proveedores de servicios mejoren la calidad de sus conexiones de manera mucho más flexibles y sobre todo menos dependientes de la mano del hombre, pero las redes tradicionales evolucionan lentamente, tienen limitaciones en su arquitectura que las hace incapaces de adaptarse a las necesidades de una empresa dada su estructura cerrada y estática implantada por antiguos estándares y protocolos de red, esto les imposibilita adaptarse adecuadamente a la demanda de usuarios y aplicaciones así que el esfuerzo se dirige al intento de aprovechar al máximo las redes existentes, invirtiendo gran parte de sus recursos en ello, sin embargo es una solución temporal, ya que ninguna de las arquitecturas de red fue diseñada para satisfacer los requerimientos actuales, dando como resultado la experimentación en nuevas alternativas y la creación de SDN (Software Defined Networking), un nuevo paradigma para experimentar e intentar cubrir las nuevas necesidades sin interrumpir en las redes existentes, prometiendo transformar las arquitecturas y la gestión de redes.

En las SDNs la inteligencia de la red se encuentra lógicamente centralizada en los controladores, estos ofrecen diversos protocolos y estándares para la comunicación con los dispositivos administrados, en estos resalta OpenFlow, un protocolo emergente y abierto que permite al controlador la determinación del camino de reenvío de paquetes que sigue el flujo en una red de switches.

SDN, propone nuevos controladores y switches con soporte OpenFlow. Pero, sin una organización que guíe el mercado, la elección de un controlador OpenFlow por parte de los administradores de red, carece de un norte. Se necesita de una evaluación objetiva para esta elección, dada las diversas características de los controladores existentes en el mercado. Por lo que se hace evidente, la necesidad de herramientas que evalúen el desempeño de estos controladores, y permitan su comparación directa, para un apoyo directo a la toma de esta decisión.

La investigación realizada hasta ahora nos otorga una base para reafirmar la necesidad de una herramienta que permita evaluar nuevas características junto al desempeño de controladores SDN, dadas las distintas implementaciones de controladores OpenFlow en un contexto SDN.

La investigación realizada está estructurada en 8 capítulos, los cuales se resumen a continuación:

Capítulo 2: En este capítulo se plantea formalmente el problema estudiado, así como los objetivos que se desean cumplir durante la investigación.

Capítulo 3: Se describe el concepto de las redes definidas por software, su paradigma y arquitectura, comparada con las redes tradicionales actuales y se lista alguna de sus aplicaciones y sus posibles problemas de escalabilidad.

Capítulo 4: Incluye conceptos y descripciones del protocolo OpenFlow enfocando su versión 1.3, tales como tipos de switches involucrados y mensajes, con la inclusión de aspectos relevantes en casos de uso en situaciones actuales.

Capítulo 5: Se presentan cuatro controladores (OpenDaylight, Ryu, Floodlight y OpenMUL), los cuales son descritos de manera general junto a su arquitectura y los componentes que ofrece.

Capítulo 6: Se definen diversas herramientas de interés para la investigación en el ámbito correspondiente a la evaluación de controladores OpenFlow.

Capítulo 7: Una lista de tres trabajos de interés que tienen relación directa con la investigación, los cuales dan soporte a conceptos y propuestas realizadas.

Capítulo 8: Se define la metodología de desarrollo a implementar en la construcción de la aplicación propuesta.

Capítulo 9: Se muestra el proceso de desarrollo de la aplicación, siguiendo la metodología propuesta, así como un análisis general de la aplicación.

2. El Problema

En este capítulo encuentran la descripción del problema a tratar, su justificación, los objetivos, la metodología a seguir en la resolución, y el alcance del trabajo propuesto.

2.1 Planteamiento del Problema

En el creciente auge que acarrea la utilización de las nuevas tecnologías alrededor de SDN, la necesidad de tener una guía en el desarrollo de implementaciones se hace imperativa. En el caso particular de controladores OpenFlow, como dicen los autores de [1], hay una falta. Falta que ha generado en desarrolladores el urge de crear controladores OpenFlow únicos, donde cada uno con sus distintas funcionalidades atienden distintos contextos.

Para evaluar los distintos controladores OpenFlow, se han realizado herramientas de medición de desempeño que midan sus cualidades innatas y permita compararlas. Sin embargo, el desarrollo de estas herramientas ha sido insuficiente, incluso orientado a satisfacer la necesidad de medir características concretas como es el caso del rendimiento y flujos soportados en la herramienta Cbench. En otros trabajos como [1] se desarrolla una herramienta para satisfacer una necesidad de establecer estadísticas individuales en pruebas con situaciones similares a las de un centro de datos.

Dadas estas razones, la propuesta de Trabajo Especial de Grado se inmiscuye en el desarrollo de herramientas de medición de desempeño para controladores OpenFlow como: Propuesta de desarrollo de una herramienta de medición de desempeño para controladores OpenFlow. Dicha herramienta permite evaluar un espectro más grande de aspectos que las herramientas de medición de desempeño predecesoras.

2.2 Justificación del Problema

Siempre estamos en la búsqueda de un avance tecnológico. Actualmente, una de las áreas más importante es la comunicación. Las redes de computadores soportan esta responsabilidad hace ya unas décadas y es nuestra responsabilidad mejorar el desempeño de ellas.

SDN junto a las tecnologías que permiten implementar su esquema, dirigen sus esfuerzos a incentivar al desarrollo de herramientas que se adapten cada vez más a las nuevas perspectivas. Estas herramientas pasarán por un proceso de evaluación, ajuste y reformas. Sin embargo, es el usuario final quien decide que herramientas perdurarán en el tiempo.

La competencia de estas herramientas genera un resultado positivo para las empresas, pues en el futuro de estas dependerá de su desempeño, de sus servicios y la estructura de sus sistemas de comunicación. El costo de una mala selección de una infraestructura de comunicación adecuada es sumamente alto, y

por tal razón debe haber herramientas de medición de desempeño que permitan estudiar las implementaciones realizadas de los elementos protagónicos en el esquema SDN. Tal es la situación que deseamos abordar desarrollando una herramienta de medición de desempeño para controladores SDN que utilizan la tecnología OpenFlow.

2.3 Objetivo General

Desarrollar una herramienta de medición del desempeño de controladores SDN que soporten la tecnología OpenFlow 1.3, con el lenguaje de programación C y que reporte resultados que no se dan en las herramientas actuales.

2.4 Objetivos Específicos

1. Crear un ambiente de desarrollo apropiado con el uso de librerías pertinentes.
2. Desarrollar un módulo para la generación y administración de paquetes OpenFlow.
3. Desarrollar un módulo para la recopilación de métricas para el análisis de estadísticas:
    a. RTT de mensajes OpenFlow.
    b. Utilización del CPU.
    c. Utilización de la memoria usada.
    d. Cantidad de paquetes enviados.
    e. Cantidad de paquetes recibidos por el controlador.
    f. Cantidad de paquetes procesados por el controlador.
    g. Cantidad de flujos definidos por el controlador.
4. Desarrollar funcionalidades que permitan realizar pruebas de estrés con un envío de paquetes incremental de manera automática:
    a. Cantidad de switches.
    b. Cantidad de paquetes.
    c. Tamaño del paquete.
    d. Tiempo de envío entre paquetes.
    e. Campos del paquete.
    f. Cantidad de paquetes afectados por dichas configuraciones.
    g. Orden de envío de los diferentes tipos de paquetes presentes:
        i. Secuencial.
        ii. Aleatorio.
5. Desarrollar funcionalidades que permitan realizar pruebas de estrés con un envío de una cantidad dada de paquetes.
6. Instalar y operar los controladores Ryu, Floodlight, OpenDaylight y OpenMUL para probar la herramienta propuesta.
7. Instalar y operar el virtualizador de switches OpenFlow, Open vSwitch y Mininet.
8. Analizar la herramienta desarrollada, comparando resultados con la herramienta Cbench bajo los mismos ambientes de prueba.

2.5 Alcance

Desarrollar una herramienta de medición de desempeño para controladores OpenFlow comparable a herramientas ya existentes como Cbench y Hcprobe, con la capacidad de evaluar nuevas métricas y proveer nuevas funcionalidades a través de la generación directa de mensajes OpenFlow. Adicionalmente se espera estar en la capacidad de adaptar la herramienta con funcionalidades opcionales para la realización de experimentos con virtualizadores de switches OpenFlow.

3. Redes Definidas por Software

Desde sus inicios las redes de comunicaciones han requerido de administración, pero para entonces resultaba algo dificultoso dada la cantidad de diversos dispositivos de red como routers, switches y firewalls de diferentes fabricantes y configuraciones. Para facilitar esto surgieron protocolos de administración que resolvían el problema de los numerosos fabricantes, pero resultaban limitados con relación a la administración directa de dispositivos con protocolos e interfaces propietarias ya que imposibilitaba mantener una única interfaz remota.

Cada fabricante por su lado desarrolló sistemas de administración centralizados que solucionaban las carencias de estos protocolos de administración, pero que sólo funcionaban con sus propios dispositivos. Eso limita la capacidad de administración de cada herramienta, aumentando la cantidad de herramientas administrativas a manejar, sumando de esta manera, complejidad y a su vez elevando el costo operacional de la red en cuestión.

Debido a la lenta aparición de estándares que solucionaran estos problemas, algunos investigadores optaron por desarrollar soluciones propias, pero estas soluciones probadas en laboratorios requerían de un largo tiempo para que fueran estandarizadas por la IETF (Internet Engineering Task Force) o por alguna otra organización de estándares [2]. Debido a esta aparición lenta de nuevos estándares se dio origen a la aparición de las llamadas redes activas, que eran redes que contaban con un API (Application Programming Interface) la cual expone los recursos de red de un dispositivo, tales como procesador, almacenamiento y colas de paquetes, permitiendo el desarrollo de nuevas funcionalidades a ser aplicadas a un subconjunto de paquetes que cruzaran el nodo en cuestión [2]. Gracias a estas APIs los investigadores podían programar las redes y realizar pruebas de nuevos mecanismos que representaban alternativas diferentes a los servicios ofrecidos por las redes tradicionales.

Luego, con el incremento de los volúmenes de datos en las redes, se aumentó la atención en la fiabilidad de las redes y en su previsibilidad. El aumento en la utilización de servicios de red causa un incremento en las mismas, ya que, si antes una red tenía 100 nodos, ahora podría tener 1000 lo cual implica 1000 sistemas de configuración, esto ha hecho que los fabricantes busquen nuevas maneras para que la programación no se realice en cada nodo sino de una manera global [3].

Como respuesta a la necesidad de mejorar el rendimiento de las redes, manteniendo la arquitectura ya existente y reutilizándola, nacen las Redes Definidas por Software que son un conjunto de técnicas desarrolladas para lograr que las redes, que interconectan los dispositivos actuales, sean programables. Para lograr este cometido, se modifica la arquitectura tradicional de los dispositivos de red la cual consta de dos capas: datos y control.

3.1 Arquitectura SDN vs. Arquitectura Tradicional

En las redes tradicionales el envío de paquetes y las decisiones de enrutamiento de alto nivel suceden dentro del mismo dispositivo, mientras que en las redes definas por software estas dos funcionalidades se separan permitiendo tener un control centralizado.

En una red convencional cuando un paquete llega a un switch, las reglas integradas en el firmware propietario del switch deciden a donde transferir el paquete, el switch reenvía todo el flujo al mismo destino, por el mismo puerto de salida. Mientras que en las redes definidas por software es posible tener acceso a las tablas de flujos del switch donde se encuentran contenidas estas reglas con la finalidad de modificar, eliminar o añadir nuevas reglas de flujo, lo cual permite controlar el tráfico de la red de manera dinámica.

SDN es un nuevo paradigma que separa el plano de control, del resto de capas del dispositivo tal como se muestra en la Figura 3.1 tomada de [4]. La mayoría de datos sólo son procesados por el plano de datos del dispositivo de red, este plano consiste en una serie de puertos que son usados para la recepción y retransmisión de data, correspondiente a la tabla de reenvío en la cual se encuentran las interfaces de salida de los datos dependiendo de su dirección destino. Así el plano de datos es el responsable del almacenamiento y despacho de datos, junto a la modificación de cabeceras en caso ser necesario.

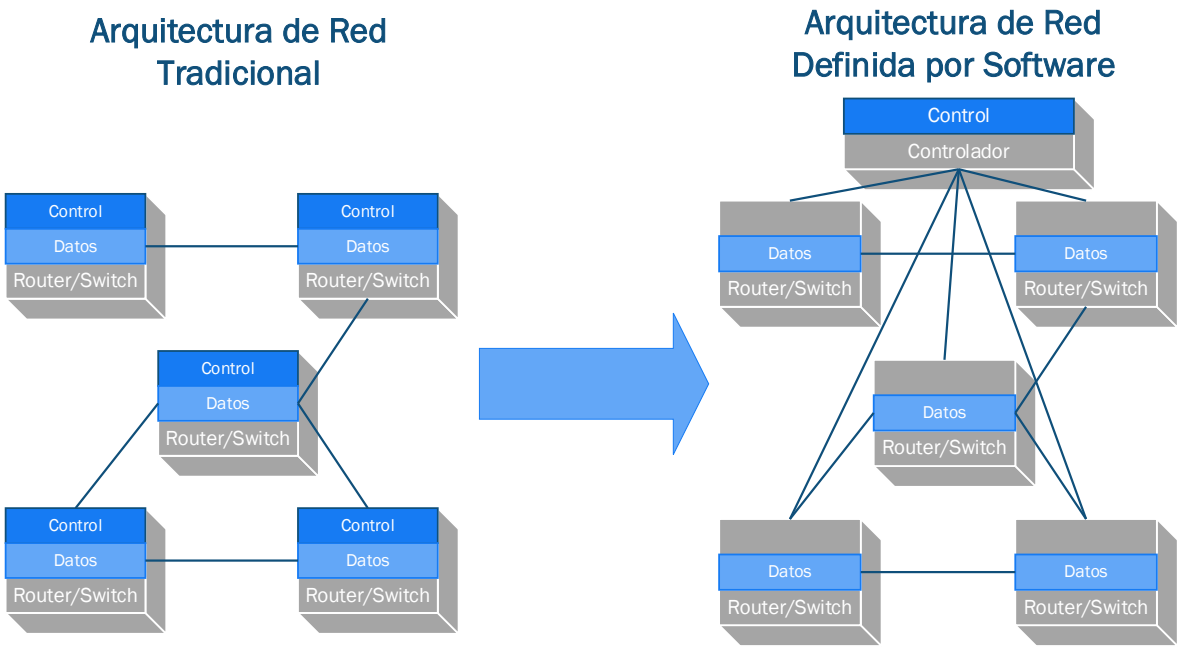

Figura 3.1: Cambio de Paradigma - El Controlador

El plano de control se separa físicamente de dispositivo de red llevando la toma de decisiones fuera del dispositivo y poniendo este servicio en un dispositivo centralizado llamado controlador SDN, dedicado a responder solicitudes de los dispositivos de red y accesible a configuración a través de APIs.

Como un ejemplo de APIs tenemos los sistemas operativos, que proveen APIs para facilitar el desarrollo de aplicaciones, ya que estas otorgan una imagen abstracta del recurso del dispositivo a utilizar, simplificada, que le oculta al desarrollador todos los detalles innecesarios de los diferentes tipos de dispositivos, modelos, vendedores, componentes, etc.

La arquitectura de SDN provee una estructura abierta en la cual cada función de control está separada del dispositivo de red y estas están localizadas en un único servidor de control el cual es accesible remotamente. Esta configuración permite a la infraestructura subyacente abstraerse en APIs que las muestran como aplicaciones y servicios de red, permitiéndole a la red ser tratada como una entidad lógica [5].

En la Figura 3.2 tomada de [5] se muestra la estructura lógica de una red definida por software. Un controlador central realiza todas las funciones complejas, incluyendo el enrutamiento, declaración de políticas y verificaciones de seguridad. Este constituye el plano de control SDN, y consiste en uno o más servidores SDN.

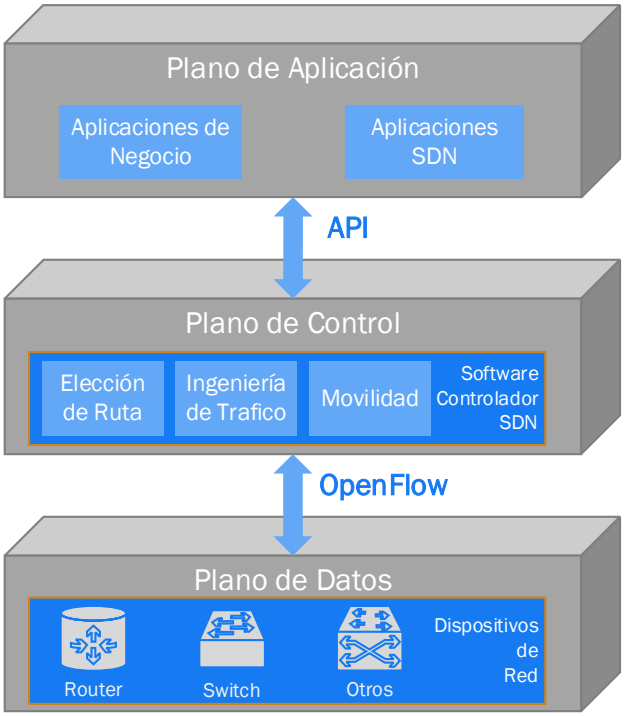

Figura 3.2: Arquitectura de una Red Definida por Software

En el momento en que un paquete llega al dispositivo de red, este busca en su tabla de reenvío la interfaz de salida, si la tabla tiene una entrada para este flujo, reenvía el paquete por el puerto indicado. De no existir entrada alguna, el dispositivo de red envía una solicitud al controlador SDN, donde se verifica si el flujo es permisible en la red, de ser así, este designa un camino para este flujo y agrega las entradas necesarias en la tabla de reenvío de cada nodo del camino

elegido. Posteriormente el dispositivo de red reenvía el paquete basándose en la tabla de reenvío dictada por el controlador.

Con todas las funciones complejas absorbidas por el controlador SDN, el switch únicamente debe de manejar los flujos de datos con las entradas de sus propias tablas de reenvío. Las comunicaciones entre el controlador SDN y los demás dispositivos de red se realizan mediante un protocolo estandarizado, uno de los más comunes es el protocolo OpenFlow. El controlador es responsable del manejo de las tablas de reenvío de los dispositivos de red, el manejo de este controlador se hace a través de una API que permite ingresar una variedad de modificaciones en el comportamiento de la red, sin realizar cambios en los aspectos de bajo nivel de la red.

En la capa de aplicación, los desarrolladores de red pueden diseñar programas que realicen diversas tareas sin necesidad de saber cómo es que los dispositivos de la red operan, por lo que sólo se abstraen de estos conceptos y se dedican únicamente al desarrollo de la lógica de la aplicación. Lo que le permite un desarrollo y despliegue más rápido de nuevas aplicaciones que dirijan los flujos de la red para satisfacer necesidades específicas en términos de seguridad o rendimiento.

3.2 Dominios

En empresas grandes, la implementación de SDN con un único controlador para manejar todos los dispositivos de la red podría ser contraproducente o indeseable, ya que esto podría sobresaturar el controlador causando un desmejoramiento en el rendimiento de la red. La solución para este problema es dividir la red en varios dominios SDN los cuales tendrán un controlador específico de dominio tal como se muestra en la Figura 3.3, tomada de [5].

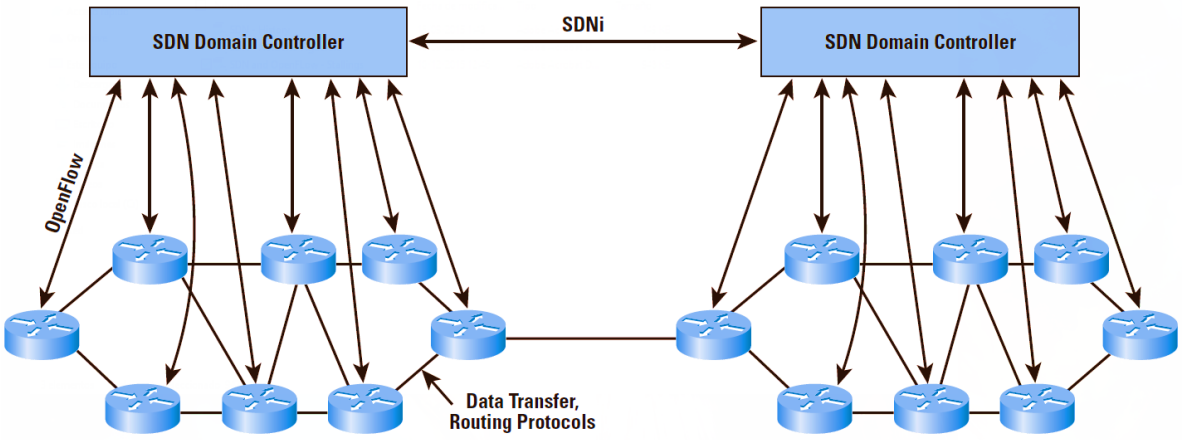

Figura 3.3: Estructura de Dominios SDN

Una de las razones para el uso de dominios SDN es la escalabilidad, ya que el número de dispositivos que un controlador SDN puede manejar es limitado. Por esto en una red de numerosos dispositivos de red puede ser necesario la implementación de múltiples controladores SDN.

Otra razón es la privacidad, ya que un cliente podría querer implementar diferentes políticas de privacidad en diferentes dominios SDN. Por ejemplo, un dominio podría ser dedicado para un repertorio de usuarios que implementen sus propias políticas de privacidad, solicitando así alguna información de red del dominio como la topología de la red, que no sea revelada ante otros entes.

La existencia de múltiples dominios crea la necesidad que los controladores se comuniquen con cada uno de los otros mediante protocolos estandarizados para el intercambio de información de enrutamiento. La IETF posee un draft de SDNi, un protocolo para el intercambio de mensajes interdominios SDN. Este protocolo se ve en la Figura 3.3 como el protocolo utilizado para cumplir este propósito. Algunas de las funciones que tendría este protocolo serian:

- Coordinar la configuración del flujo originado por aplicaciones que contienen información con requisitos de trayectoria, calidad de servicio, acuerdos de nivel de servicio, entre otros, a través de los diferentes dominios SDN.
- Intercambio de información de accesibilidad para facilitar el enrutamiento interdominio. Este intercambio permite a un flujo atravesar múltiples dominios y obtener el camino más adecuado cuando existen varios caminos disponibles.

3.3 Aplicaciones

Las redes SDN pueden ser aplicadas en una gran variedad de entornos, en los cuales, separando los planos de control y de dato, y volviendo la red programable se permite un control más dinámico, simplificando el desarrollo de nuevas aplicaciones, servicios y protocolos de red que cumplan con mayor rendimiento y de una manera más directa con las necesidades del cliente.

En las redes empresariales una gestión adecuada es de vital importancia, las redes SDN pueden ser usadas para manejar políticas de red mediante su programación, pueden manejar funciones de middlebox como firewalls, NAT (Network Address Translation), balanceo de carga o control de acceso, reduciendo así el costo operacional y de administración de la red.

En los centros de dato un gran problema es el alto consumo energético, las redes SDN pueden permitir usar sólo una parte de la red en momentos de menor demanda, de tal forma de que no se influya el rendimiento, pero disminuya el consumo energético de la red.

Por último, en las infraestructuras de red de acceso inalámbrico SDN puede ayudar a los operadores móviles a simplificar la administración de sus redes y permitir nuevos servicios que soporten un incremento del tráfico previsto para las nuevas redes.

3.4 Escalabilidad

A pesar de las múltiples ventajas de SDN, al presentar una solución centralizada, se ha revelado que posee un problema de escalabilidad, repercutiendo en el desempeño de las redes. Por ejemplo: En un pequeño centro de datos con 100.000 hosts, la tasa máxima de llegada de flujos puede ser hasta de 300.000 por segundo, y la tasa promedio es entre 1.500 y 10.000 flujos por segundo [6]. Para solucionar este caso con un controlador y un rendimiento promedio de 2.000 flujos por segundo, a tasa promedio, se necesitarían entre uno y cinco controladores, y a tasa máxima, 150 de estos, y teniendo una mayor escala, se agrava el problema presentado.

La solución de este problema encontró dos vertientes, una es la mejora del controlador en sí, optimizando su manejo multi-hilo o desarrollando nuevos controladores construidos desde el kernel, de manera tal que se elimine el intercambio de ambientes entre el espacio de usuario y kernel [7]. La otra opción, para redes hibridas, compone la clasificación de flujos y eventos acorde a su duración y prioridad, donde los flujos de corta estadía puedan ser manejados por los switches y aquellos de larga estadía se manejen en el controlador a modo balanceo [8].

4. OpenFlow 1.3

OpenFlow es un estándar abierto diseñado como protocolo de comunicación entre el plano de control y datos inherente en dispositivos de red. Fue desarrollado en la Universidad de Stanford a finales el 2008 [9] con el propósito de permitir la investigación e innovación sobre las infraestructuras ya existentes bajo el uso de múltiples flujos que no interfirieran con tráfico en producción.

En el 2011, la ONF (Open Networking Foundation) fue formada por un grupo de proveedores de servicio [1] para comercializar, estandarizar y promover el uso de OpenFlow en redes de producción. La ONF como un nuevo tipo de organización de desarrollo de estándares posee un departamento de mercadeo muy activo que se ha usado para promover el protocolo OpenFlow y cualquier esfuerzo relacionado a SDN. Actualmente la organización mantiene conferencias anuales llamada ONS (Open Networking Summit) como parte de estos esfuerzos [10].

La visión que promueven las Redes Definidas por Software sirve como puente para OpenFlow, el cual dispensa beneficios substanciales que centralizan el plano de control y crean múltiples flujos con características programables definidas a través de un alto nivel de abstracción, esto deja OpenFlow con la capacidad de dar paso a la experimentación en nuevos protocolos que pongan a prueba los paradigmas actuales que definen la manera en la que definimos y creamos las redes.

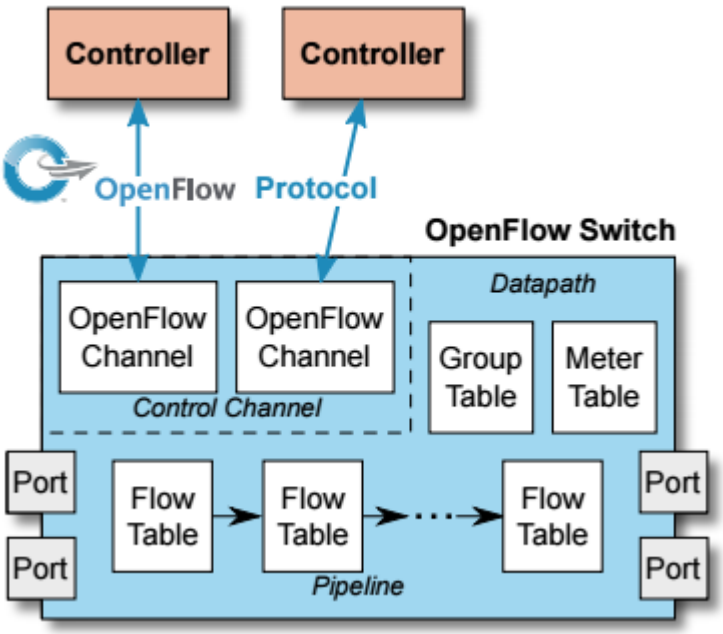

Figura 4.1: Estructura de OpenFlow

[1] Entre instituciones académicas y gubernamentales, empresas, proveedores de servicio, compañías de software y fabricantes de equipos comprenden más de 90 miembros en la ONF.

OpenFlow parte de la premisa que el plano de control debería separarse del plano de datos, un rasgo inusual en dispositivos de red común. Crea una arquitectura donde el controlador se encargará del plano de control y el switch/router, mantendrá el plano de datos, ambos intercomunicados a través del protocolo de comunicación OpenFlow, cuya estructura se muestra en la Figura 4.1 tomada de [11]. Así como el set de instrucciones de un CPU (Central Processing Unit), el protocolo especifica primitivas básicas explotables por aplicaciones de software para programar el reenvío de paquetes en dispositivos de red [9], reenvío que se maneja bajo la abstracción de flujos definidos en tablas de flujos OpenFlow. En este trabajo se propone una herramienta usando la versión 1.3 del protocolo, ya que al ser la más renombrada hasta los inicios del 2016 como muestra la Figura 4.2 (tomada de Google Trends), nos provee mayor soporte de la comunidad.

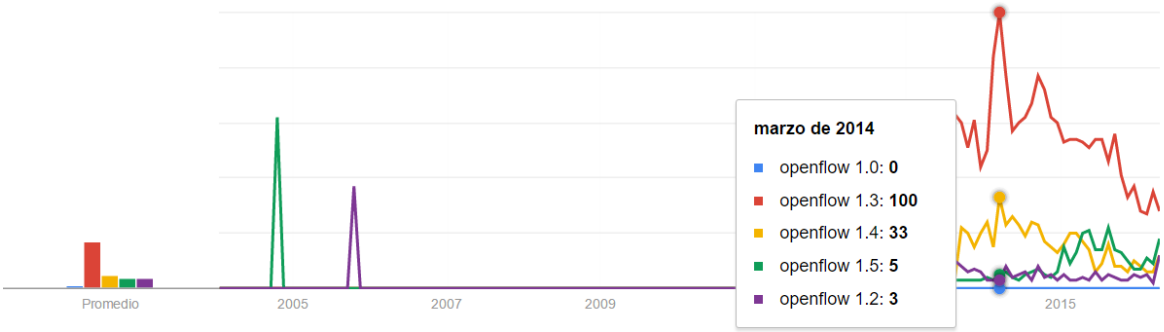

Figura 4.2: Análisis de Tendencia OpenFlow por Google

4.1 Switches

OpenFlow identificó en switches y routers de distintos proveedores, que las tablas de flujos utilizadas para la implementación de multiples servicios como firewalls, QoS (Quality of Service) y NAT comparten cierto grupo de funcionalidades, explotando estas OpenFlow crea un protocolo abierto para programar tablas de flujos [12].

Los switches OpenFlow se categorizan en dos grupos, Dedicados e Híbridos, los que entran en el primer grupo, incapaces de realizar el procesamiento común de capa 2 y capa 3[2], y los que entran en el último grupo como switches y routers comunes con funcionalidades añadidas para el soporte del protocolo.

4.1.1 Dedicado

Un switch OpenFlow dedicado, no puede procesar un pipeline distinto al que ofrece OpenFlow. Posee un solo camino de datos dirigido a un controlador remoto que dicta los posibles puertos de reenvío.

En esta situación se crea una amplia cantidad de flujos bajo las características que permita la tabla de flujos. Caminos como estos pueden ser definidos para una

[2] En el resto de la investigación nos referiremos a las capas de switches 2 y 3 por sus acrónimos conocidos en inglés L2 y L3 correspondientes.

conexión UDP (User Datagram Protocol), una dirección única, una VLAN (Virtual Local Area Network), o un puerto, incluso paquetes con cabeceras particulares fuera del estándar IPv4 (Internet Protocol version 4) [12].

4.1.2 Híbrido

Un switch OpenFlow híbrido es capaz de manejar paquetes L2 (Layer 2), permitir agrupación VLANs, paquetes L3 (Layer 3), ACL (Access Control List), QoS y OpenFlow. Tales mecanismos necesitan de un pre procesamiento que pueda dirigir los paquetes al pipeline OpenFlow o pipeline tradicional, ya sea a través de una etiqueta VLAN o incluso reenviar todos los paquetes al pipeline OpenFlow y que este se encargue de reenviar los paquetes correspondientes a los puertos designados para el pipeline tradicional.

La implementación de OpenFlow en los switches existentes se apoya en la reutilización de algunos recursos, típicamente la tabla de flujos reutiliza las TCAM (Ternary Content-Addressable Memory), así se facilita la migración de elementos principales como el protocolo al sistema operativo del switch. Aun cuando el proceso migratorio a SDN ya está en marcha, la mayoría de los switches y routers OpenFlow comerciales y disponibles son híbridos.

4.1.3 Puertos

Los puertos mostrados en la Figura 4.3 funcionan como interfaz de red para el traspaso de paquetes. Categorizados en físicos, lógicos y reservados, siendo estos dos últimos puertos, virtuales. Por lo tanto, OpenFlow define un número de puertos disponibles para el procesamiento OpenFlow que no necesariamente coincide con el número de interfaces que contiene el hardware del switch. A continuación, se describe cada uno de ellos.

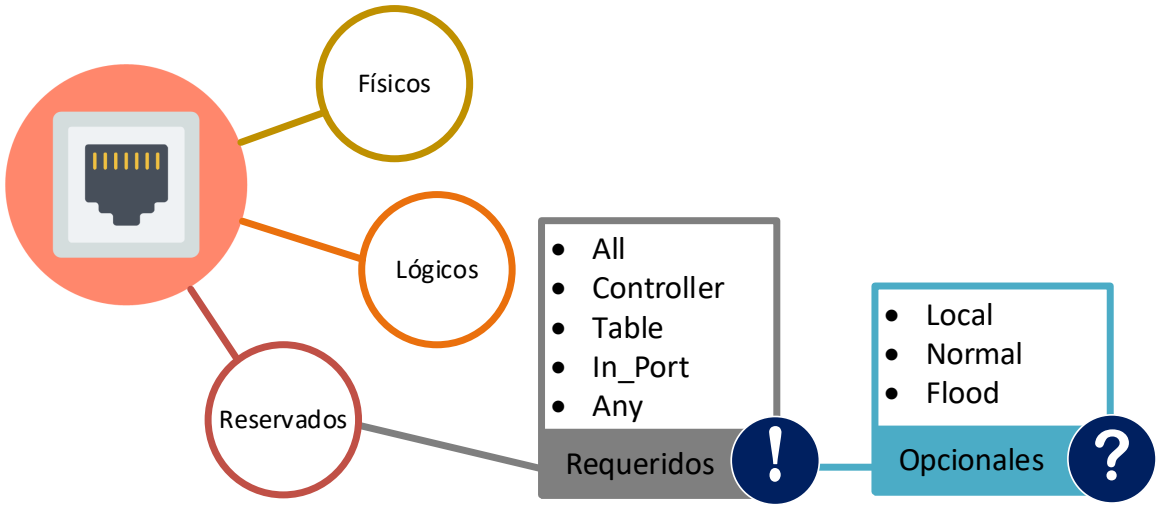

Figura 4.3: Puertos OpenFlow

4.1.3.1 Físicos

Los puertos físicos definidos en OpenFlow corresponden a las interfaces de hardware que posee el switch. En algunos casos los puertos OpenFlow son

virtualizados sobre el hardware, implicando que los puertos físicos OpenFlow sólo representan una parte virtual de la interfaz de hardware correspondiente.

4.1.3.2 Lógicos

Los puertos lógicos son puertos definidos que no pertenecen directamente a las interfaces de hardware del switch. Son abstracciones de alto nivel que funcionan de cara al procesamiento OpenFlow de la misma manera que un puerto físico. Las únicas diferencias entre éstos es que un paquete asociado a un puerto lógico posee un campo metadata adicional llamado Tunnel-ID y al recibir un paquete este debe ser enviado al controlador.

4.1.3.3 Reservados

Los puertos reservados están destinados a realizar acciones de reenvío usando métodos comunes. Entre los puertos reservados encontramos algunos con carácter obligatorio que todo switch OpenFlow debe poseer, y otros con carácter opcional dependiente del proveedor.

Requeridos:

- **ALL:** Representa todas las interfaces, excluyendo el puerto entrante y aquellos configurados con la bandera OFPPC_NO_FWD.
- **CONTROLLER:** Es un puerto bidireccional destinado a comunicar el camino de datos con el controlador usando mensajes Packet-In y Packet-Out. Es usado de manera recurrente al recibir el primer paquete de un nuevo flujo, de esta manera el controlador decide si el flujo debería ser añadido a la tabla.
- **TABLE:** Representa la tabla inicial del pipeline OpenFlow. Es usado en acciones de salida con mensajes Packet-Out y envíos de paquete a la primera tabla de flujo. Permite un pipeline multitabla (hasta 255).
- **IN_PORT:** Representa el puerto de entrada de un paquete. Pero sólo puede ser usado como puerto de salida.
- **ANY:** Simboliza cualquier puerto en el uso de algunos comandos OpenFlow cuando ningun otro puerto es especificado. Sin embargo no se puede usar como puerto de entrada o salida

Opcionales:

- **LOCAL:** Es un puerto bidireccional usado para el manejo local del switch. Permite a las aplicaciones remotas acceder a los puertos y servicios del switch a través de OpenFlow.
- **NORMAL:** Es un puerto de salida que permite al switch funcionar como un switch Ethernet tradicional usando el pipeline de reenvio tradicional (L2, VLAN, y/o L3). Este puerto sólo es de uso para switches híbridos.
- **FLOOD:** Representa la acción de inundación en el uso de pipeline tradicionales. Únicamente puede ser usado como puerto de salida, enviando generalmente el paquete a través del minimo spanning tree, sin incluir las interfaces de entrada (IEEE 802.1D) o aquellas bloqueadas con el estado OFPPS_BLOCKED. Sin embargo se puede espcificar con el campo VLAN ID cuales puertos inundar.

4.1.4 Tabla de Flujo

El switch OpenFlow contiene al menos una tabla de flujos con una acción asociada a cada entrada de flujo y un canal seguro que permite la conexión con el controlador [12]. Estas tablas contienen los flujos OpenFlow que definen el plano de datos. Para alcanzar el alto desempeño y bajo costo las entradas de flujos definidas en las tablas olvidan la habilidad de especificar arbitrariamente el manejo de cada paquete y definen un grupo de acciones más limitado pero aún, flexible.

Las tablas están enumeradas secuencialmente. El manejo de paquetes comienza en la primera tabla de la secuencia, la número cero. Acorde al tipo de paquete se compararán los campos especificados por las entradas de flujo. Estas entradas especifican los valores de los campos que aceptan, sin embargo pueden omitir campos si asi fueran definidas.

Una vez un paquete OpenFlow es recibido por el switch, los campos de cabecera son extraídos y comparados contra los campos de cabecera en las entradas flujo. Por cada paquete que sea comparado exitosamente son actualizados los contadores asociados a la entrada y el set de instrucciones en ella se ejecuta.

Algunas de las instrucciones puede redireccionar directamente el paquete (incluyendo nueva metadata y campos actualizados) a otra tabla con número mayor donde se tratará de la misma manera. Si el proceso de búsqueda en la tabla no resulta con comparaciones exitosas, las acciones del switch dependerán de las instrucciones definidas en la entrada de flujo para fallo en tabla. Esta entrada dicta la serie de acciones realizadas cuando el paquete entrante no da con ninguna de las entradas de la tabla [13].

Cada entrada de flujo consiste en:

- Campos de cabecera
- Prioridad
- Contadores
- Instrucciones
- Timeout
- Cookie

4.1.4.1 Tabla de Grupos

Las entradas de flujo se pueden agrupar, por lo tanto se puede redireccionar paquetes a estos grupos. Estos grupos estan formados por [14]:

- **Identificador de grupo:** Un identificador de 32 bits.
- **Tipo de grupo:** Tiene un papel importante al funcionar como discriminante en la semántica del grupo.
- **Contadores:** Formado de campos incrementales para el manejo de estadísticas en el procesamiento de paquetes.
- **Cubeta de acciones:** Es una lista de acciones agrupadas en una “*cubeta*” con parametros asociados.

Los tipos de grupos asi como los distintos puertos son separados en aquellos requeridos y aquellos opcionales [14]:

Requeridos [14]:

- **All:** Amerita ejecutar todas las cubetas de acciones de un grupo con un paquete, por lo que éste se clona por cada cubeta.
- **Indirect:** Implica ejecutar sólo una cubeta de acción, aquella que haya sido definida en el grupo.

Opcionales [14]:

- **Select:** A través de un algoritmo se escoge una cubeta de acciones a ejecutar (p.e. Round Robin). Si una cubeta posee un puerto inhabilitado, el switch está en la capacidad de seleccionar entre las restantes.
- **Fast Failover:** Corresponde a la ejecución de la primera cubeta de acciones activa. Determinado al evaluar la vida de un puerto/grupo específico de la cubeta. En caso de no existir una cubeta activa, el paquete es descartado.

4.1.4.2 Tabla de Medidores

Para la realización de operaciones de QoS como limitar la tasa de paquetes, se puede utilizar las entradas de medidores asociadas a los flujos. El tipo de medidor puede ser especificado en el set de instrucciones al que es atado. Las entradas de medidores están compuestas por [14]:

- **Identificador del medidor:** Un identificador de 32 bits.
- **Bandas del medidor:** Una lista desordenada de bandas que especifican la tasa y la manera de manejar el paquete.
- **Contadores:** Formado de campos incrementales para el manejo de estadísticas en el procesamiento de paquetes (ver Tabla 4.2 tomada de [14]).

Las bandas usan tasas de paquetes para indicar los limites superior/inferior en los que mantener los paquetes que pertenezcan al flujo. Se mide la tasa actual y se procesa el paquete con la banda correspondiente. La banda correspondiente es aquella con la tasa más alta que este por debajo de la tasa actual. Cada banda posee 4 tipos de campos [14]:

- **Tipo de banda:** Define como se procesará el paquete. Si bien no existe ningún tipo de banda requerido, opcionalmente puede ser:
  - **Drop:** Implica descartar el paquete. Usado como límite.
  - **DSCP remark:** Incrementa la precedencia de descarte en el campo DSCP (Differientiated Services Code Point) de la cabecera IP.
- **Tasa:** Define la tasa más baja al cual la banda puede aplicar.
- **Contadores:** Formado de campos incrementales para el manejo de estadísticas en el procesamiento de paquetes.
- **Argumentos específicos:** Algunos tipos de banda como DSCP remark tienen argumentos opcionales.

4.1.4.3 Campos de Cabecera

Los campos de cabecera poseen información a comparar con los datos encontrados en la cabecera de los paquetes, esto comprende, puerto de ingreso, direcciones origen, y metadata especificada por tablas anteriores. Los campos de cabecera usados dependen del tipo de paquete recibido [13]. Sin embargo, existen un grupo de campos que todo switch OpenFlow 1.3 debe soportar, descritos en la Tabla 4.1, tomada de [14] .

Campo	Descripción
Ingress Port	Puerto de ingreso. Puede ser un puerto físico o lógico.
Ethernet source address	Dirección Ethernet fuente.
Ethernet destination address	Dirección Ethernet destino.
Ethernet type	Tipo Ethernet del *payload* del paquete OpenFlow, despues de las etiquetas VLAN.
IP protocol number	Número del protocolo IPv4 o IPv6
IP source address	Dirección IPv4/IPv6 fuente. Puede usar máscara.
IP destination address	Dirección IPv4/IPv6 destino. Puede usar máscara.
Transport source port	Puerto TCP/UDP fuente.
Transport destination port	Puerto TCP/UDP destino.

Tabla 4.1: Lista de Campos Requeridos

El compendio de los campos restantes a la Tabla 4.1 representa protocolos y funcionalidades como VLAN, SCTP, ARP, ICMP, MPLS y PBB.

4.1.4.4 Prioridad

La prioridad es un identificador usado para comparar la precedencia de una entrada a la hora de tomar acciones sobre un paquete. Al comparar el paquete con cada una de las entradas, únicamente se debe seleccionar la entrada con mayor prioridad entre aquellas que sean capaz de aceptar el paquete. De haber múltiples entradas con la mayor de las prioridades presentes la entrada de flujo seleccionada es indefinida, caso que sólo surge cuando el controlador no coloca la bandera OFPFF_CHECK_OVERLAP y agrega entradas sobrepuestas [14].

4.1.4.5 Contadores

Los contadores son usados para recolectar estadísticas de un flujo particular, tal como número de paquetes recibidos, número de bytes o duración del flujo desde que fue creado en la tabla, tanto en segundos como nanos segundos. Sin embargo, estos también son mantenidos a lo largo del switch para generar estadísticas de tablas, puertos y colas. En la Tabla 4.2, tomada de [14], se muestra una lista de estos contadores [13].

Counter	Bit	
Per Flow Table		
Reference Count (active entries)	32	*Required*
Packet Lookups	64	*Optional*
Packet Matches	64	*Optional*
Per Flow Entry		
Received Packets	64	*Optional*
Received Bytes	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Port		
Received Packets	64	*Required*
Transmitted Packets	64	*Required*
Received Bytes	64	*Optional*
Transmitted Bytes	64	*Optional*
Receive Drops	64	*Optional*
Transmit Drops	64	*Optional*
Receive Errors	64	*Optional*
Transmit Errors	64	*Optional*
Receive Frame Alignment Errors	64	*Optional*
Receive Overrun Errors	64	*Optional*
Receive CRC Errors	64	*Optional*
Collisions	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Queue		
Transmit Packets	64	*Required*
Transmit Bytes	64	*Optional*
Transmit Overrun Errors	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Group		
Reference Count (flow entries)	32	*Optional*
Packet Count	64	*Optional*
Byte Count	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Group Bucket		
Packet Count	64	*Optional*
Byte Count	64	*Optional*
Per Meter		
Flow Count	32	*Optional*
Input Packet Count	64	*Optional*
Input Byte Count	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Meter Band		
In Band Packet Count	64	*Optional*
In Band Type Count	64	*Optional*

Tabla 4.2: Lista de Contadores OpenFlow

4.1.4.6 Instrucciones

Cada entrada de flujo tiene una serie de instrucciones asociadas. Estas pueden implicar la modificación del paquete en curso, la lista de acciones que lo acompaña o el mismo procesamiento del paquete.

Existen instrucciones requeridas, y algunas opcionales. Sin embargo, estas no pueden ser ejecutadas en cualquier orden, y tampoco pueden ser definidas de manera repetitiva en la misma entrada de flujo.

Requeridas [14]:

- **Write-Action:** Añade las acciones especificadas a la lista de acciones actual. En caso de ya existir, se sobrescriben.
- **Goto-Table:** Especifica la próxima tabla a procesar con el identificador otorgado. Aquellos switches que posean una única tabla no pueden ejecutar esta instrucción. En caso de que una entrada de flujo contenga ésta instrucción ausente, se ejecuta el grupo de acciones.

Opcionales [14]:

- **Meter:** Reenvía el paquete al medidor indicado para la evaluación de su descarte.
- **Apply-Actions:** Realiza las acciones especificadas inmediatamente, sin la modificación de la lista de acciones.
- **Clear-Action:** Limpia la lista de acciones inmediatamente.
- **Write-Metadata:** Añade el valor de metadata enmascarada al campo correspondiente. La máscara indica cuales bits del registro deben ser modificados.

4.1.4.7 Acciones

La lista de acciones puede variar dependiendo del switch con el que se trabaje, OpenFlow híbrido o dedicado puesto que los híbridos tienen características adicionales que ameritan atención para su correcto desempeño, sabiendo esto, las acciones pueden ser separadas en requeridas y opcionales.

Requeridas [14]:

- **Output:** Envío del paquete a través de uno o varios puertos OpenFlow.
- **Drop:** Descartar el paquete. Este comportamiento no se realiza a través de una acción explicitamente, se realiza cuando encontramos una entrada de flujo sin acción especificada.
- **Group:** Procesa el paquete a través de un grupo de tablas específico. En cuyo caso el procesamiento depende del tipo de grupo.

Opcionales [14]:

- **Set-Queue:** Reenvía el paquete a una cola atada a un puerto, añadiendo un *queue id* al paquete. El reenvío será dictado por la configuración de la cola. Esta acción es usada para QoS.
- **Push-Tag/Pop-Tag:** En caso de soporte VLAN, existe la habilidad de añadir y remover etiquetas VLAN con el uso de etiquetas VLAN, MPLS (Multiprotocol Label switching) y PBB (Provider Backbone Bridges). Estas acciones son:
  - Push VLAN header.

  - Pop VLAN header.
  - Push MPLS header.
  - Pop MPLS header.
  - Push PBB header.
  - Pop PBB header.
- **Set-Field:** Las acciones Set-Field están diseñadas para editar los campos identifcables por su *field type*. En el caso de modificaciones sobre campos VLAN, los cambios se realizan en la etiqueta más externa a menos que se especifique lo contrario.
- **Change-TTL:** Son distintas acciones que permiten para editar, decrementar y copiar los valores IPv4 TTL (Time To Live), IPv6 Hop Limit o MPLS TTL que se encuentren en el paquete. Estas acciones son:
  - Set MPLS TTL.
  - Decrement MPLS TTL.
  - Set IP TTL.
  - Decrement IP TTL.
  - Copy TTL outwards.
  - Copy TTL inwards.

4.1.4.8 Grupo de Acciones

Los paquetes son acompañados por un grupo de acciones, inicialmente vacío. Las entradas de flujo modifica este grupo con instrucciones capaces de añadir acciones como Write-Action. Este grupo de acciones sigue al paquete a través de tablas y puede ser modificado en cada tabla hasta que termine la navegación. Cuando termine la navegación se ejecutarán todas las acciones del grupo.

El grupo de acciones no puede contener acciones repetidas. Para estos casos se permite la ejecución de acciones con la instrucción Apply-Actions. Ya sea a excepción de la ejecución de acciones a través de Apply-Actions, las acciones se ejecutan en un orden expresado a continuación:

- Copy TTL inwards: Se aplica la acción “Copy TTL inwards” en el paquete.
- Pop: Se aplican todas las acciones “Pop” posibles en el paquete.
- Push-MPLS: Se aplica la acción “Push MPLS header” en el paquete.
- Push-PBB: Se aplica la acción “Push PBB header” en el paquete.
- Push-VLAN: Se aplica la acción “Push VLAN header” en el paquete.
- Copy TTL outwards: Se aplica la acción “Copy TTL outwards” en el paquete.
- Decrement TTL: Se aplica la acción “Decrement TTL” en el paquete.
- Set: Se aplican todas las acciones “Set-Field” posibles en el paquete.
- QoS: Se aplican todas las acciones QoS como “Set-Queue” en el paquete.
- Group: Si un grupo es especificado, se ejecutan todas las acciones de el/los grupos en este mismo orden.
- Output: Si no se especifica algún grupo de acciones, se reenvia el paquete por el puerto especificado en la acción output.

La razón de este orden es mantener la integridad del paquete con una estructura lógica de ejecución.

4.1.4.9 Lista de Acciones

La instrucción Apply-Actions y los mensajes Packet-Out incluyen una lista de acciones. Estas acciones son procesadas ordenadamente, sin embargo, en caso de reenvío, no hay garantía de que paquetes destinados al mismo puerto de salida, salgan de manera ordenada.

4.1.5 Timeout

El timeout representa el tiempo de inactividad que posee un flujo antes de ser retirado de la tabla.

4.1.6 Cookie

La cookie es un valor opaco seleccionado por el controlador. Le permite al controlador filtrar las estadísticas, modificar o eliminar flujos.

4.2 Mensajes

Los mensajes, permiten la comunicación entre el switch y el controlador OpenFlow, son intercambiados sobre un canal seguro para notificar y realizar las configuraciones junto al manejo de recursos (ubicar flujos, recolectar eventos y estadísticas).

En cada uno de los mensajes, la cabecera posee la misma estructura. Independientemente de la versión del protocolo que se use existirán 4 campos en la cabecera, estos se pueden observar en la Figura 4.4, tomada de [15].

- El campo *version* especifica la versión del protocolo.
- El campo *type* indica el modelo de mensaje y la forma en la que se tratará.
- El campo *length* señala donde termina el mensaje comenzando desde el primer byte de la cabecera.
- El campo *xid* es un indicador de transacciones usado para hacer la correspondencia entre solicitudes y respuestas.

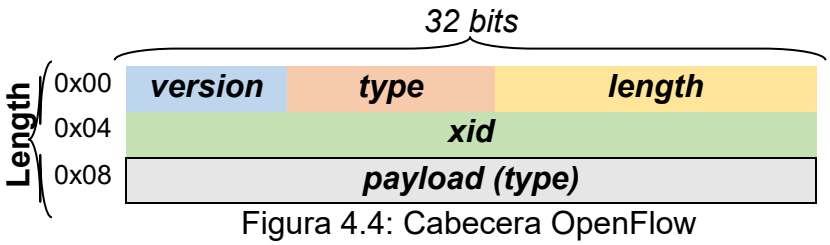

Figura 4.4: Cabecera OpenFlow

La conexión TLS (Transport Layer Security) [16], o la conexión TCP desencriptada, es iniciada por el switch al ser encendido, usualmente destinada al puerto por defecto del controlador, 6633 (modificable). La comunicación entre el switch y el controlador no es revisada contra la tabla de flujos, por lo tanto, el switch debe identificar el tráfico entrante como local antes de compararlo contra la tabla de flujo. En caso de tener una comunicación interrumpida, el switch intentará comunicarse con el o los controladores de respaldo. Después de N intentos fallidos, el switch entra en un modo de emergencia y reinicia inmediatamente la conexión, posterior

a eso el proceso de comparación en la tabla de flujos se realiza con las reglas que posean el bit de emergencia encendido como se puede observar en la Figura 4.5.

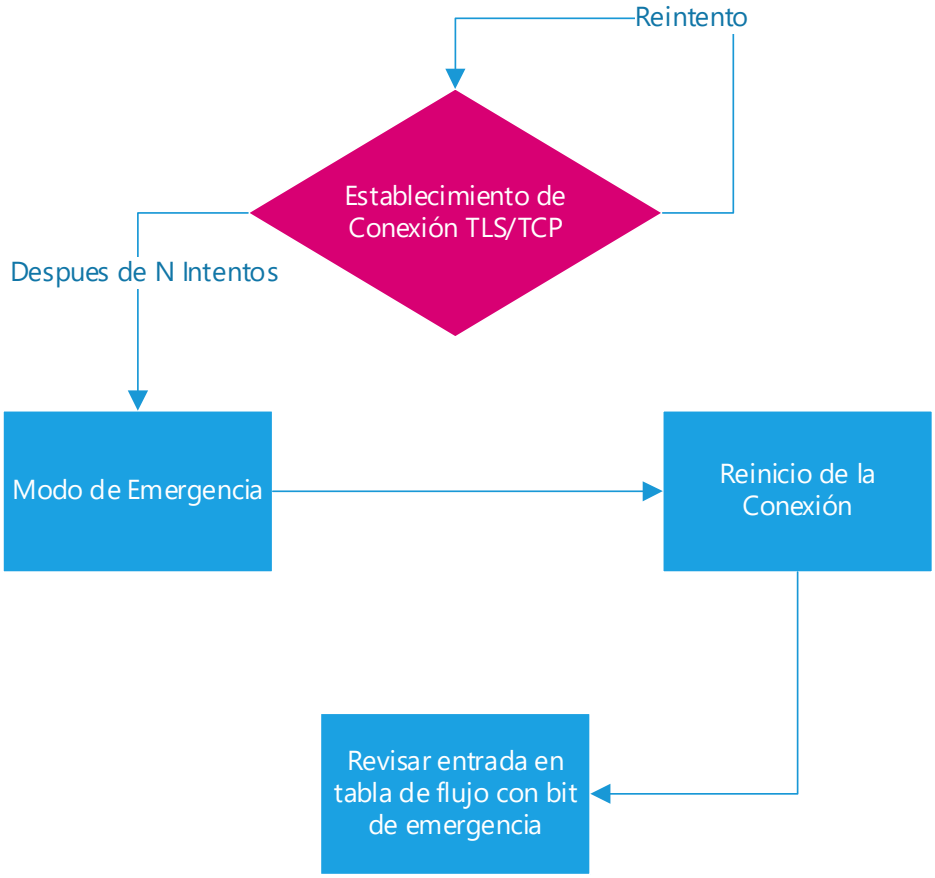

Figura 4.5: Flujo Alterno en el Manejo de la Conexión OpenFlow

Los mensajes entre el controlador y el switch inician con la cabecera OpenFlow. Esta cabecera especifica la versión del protocolo OpenFlow, el tipo de mensaje, la longitud del mensaje, y el ID de transacción del mensaje [17]. El protocolo OpenFlow define tres tipos de mensaje, cada uno con múltiples subtipos:

- Controlador-al-switch.
- Simétricos.
- Asíncronos.

4.2.1 Controlador-al-Switch

Estos mensajes son iniciados por el controlador para manejar o inspeccionar directamente el estado del switch. Si bien estos mensajes pueden o no necesitar respuesta del switch, son categorizados en los siguientes subtipos:

- Features.
- Configuration.
- Modify-State.
- Read-State.

- Packet-Out.
- Barrier.
- Role-Request.
- Asynchronous-Configuration.

4.2.1.1 Features

Una vez establecida la sesión, el controlador envía hacia el switch un mensaje (FEATURES_REQUEST) solicitando la lista de funcionalidades que ofrece. Este tipo de mensajes está conformado únicamente por la cabecera OpenFlow, con el campo *type* correspondiente. Al recibir la solicitud el switch entonces deberá responder un mensaje (FEATURES_REPLY) enumerando sus capacidades.

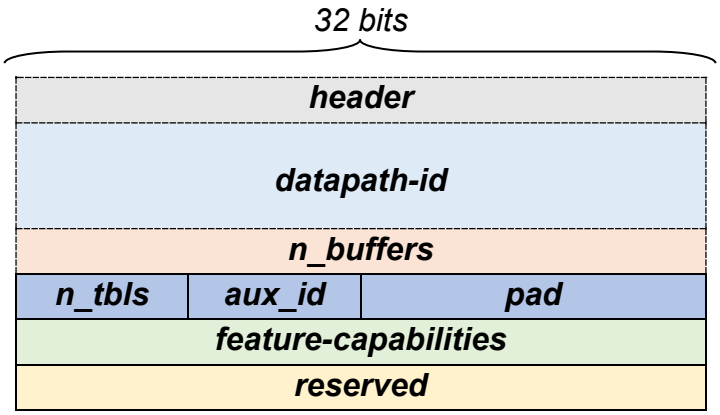

Figura 4.6: Estructura del Mensaje Features-Reply

La estructura de un mensaje FEATURES_REPLY como la desplegada en la Figura 4.6, tomada de [18], responde a la necesidad de indicar ciertas características del switch:

- El campo *datapath-id* cumple con la misma función que podría dar la dirección física de un switch Ethernet ya que funciona como un identificador único para el pipeline de procesamiento OpenFlow que maneja los paquetes.
- El campo *n_buffers* indica la cantidad de paquetes
- que el switch puede encolar.
- El campo *n_tbls* representa el número de tablas en el switch. Posterior a este se encuentra otro campo.
- El campo *aux_id* dicta la manera en la que se tratará el canal de transporte usado para comunicarse con el controlador, las opciones comprenden tratarlo como un controlador maestro o uno auxiliar.
- El campo *pad*, tiene la responsabilidad de agregar el espacio necesario para mantener el alineamiento de bytes donde se es necesario.
- El campo *feature-capabilities* enlista las capacidades que el switch ofrece usando mascaras de bits.

4.2.1.2 Configuration

El controlador está en la capacidad de consultar las propiedades del switch (GET_CONFIG_REQUEST) y recibir respuestas (GET_CONFIG_REPLY) del

mismo, incluso está en capacidad de modificar (SET_CONFIG) las propiedades del pipeline, propiedades relacionadas al manejo de paquetes fragmentados o la cantidad de bytes compartidos con el controlador en mensajes Packet-In.

El mensaje GET_CONFIG_RESQUEST comprende únicamente la cabecera OpenFlow con el valor adecuado para el campo *type*, sin embargo, los mensajes GET_CONFIG_REPLY y SET_CONFIG poseen dos campos adicionales bajo el nombre de *flags* y *miss_send_len,* de 16 bits, para indicar las operaciones activas o a activar en el procesamiento de paquetes fragmentados y la cantidad de bytes en el paquete a encapsular dentro del mensaje Packet-In, puede observarse la estructura en Figura 4.7, tomada de [19] .

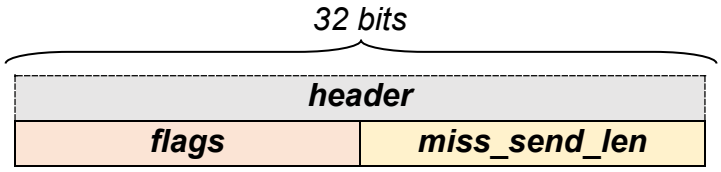

Figura 4.7: Estructura de Mensajes
Get-Config-Request, Get-Config-Reply y Set-Config

4.2.1.3 Modify-State

En este grupo se encuentran mensajes capaces de administrar el estado de los switches (añadir, modificar y/o remover entradas de flujo, grupos o puertos). Estos pueden ser:

- **ADD:** Añade un flujo/grupo. Sin embargo, antes de agregar un flujo/grupo a la tabla pueden existir condiciones que afecten el resultado como:
  a) El switch debe revisar si el mensaje contiene la bandera OFPFF_CHECK_OVERLAP encendida, esto le indica al switch si debe rechazar (en caso de estar encendida) o tolerar la posibilidad de flujos con rangos interceptados, es decir la posibilidad de que un paquete pueda coincidir con dos o más flujos de igual prioridad. El rechazo de la adición se realiza con una respuesta (OFP_ERROR_MSG) con el tipo de error OFPET_FLOW_MOD_FAILED y código OFPFMFC_OVERLAP. De caso contrario se añade el flujo en el índice más bajo.
  b) Si se añade un flujo con campos idénticos, se reemplaza el flujo, incluyendo contadores. De no haber espacio para añadir el flujo en la tabla, se responde (OFP_ERROR_MSG) con el tipo de error OFPET_FLOW_MOD_FAILD y código PFOFMFC_ALL_TABLES_FULL.
  c) Si un puerto referenciado es inaccesible, la respuesta de error (OFP_ERROR_MSG) será del tipo OFPET_BAD_ACTION y con código OFPBAC_BAD_OUT indicando así que no se pudieron realizar las acciones necesarias debido a un problema con el puerto.
  d) Si el puerto referenciado puede ser accesible en un futuro (línea de puertos físicos añadidos al dispositivo) el switch puede soltar silenciosamente los paquetes enviados a ese puerto o responder con un error (OFPBAC_BAD_PORT) y rechazar el mensaje.

- **MODIFY:** Modifica una entrada de flujo. Sin embargo, si no existe alguna entrada de flujo que coincida con la cabecera que se desea modificar, el comando MODIFY actuará como el comando ADD. De otra manera el grupo de acciones será modificado.
- **DELETE:** Elimina una entrada de flujo. Para peticiones de eliminar flujos, la acción natural en caso de haber encontrado la entrada de flujo buscada es que aquella con la bandera OFPFF_SEND_FLOW_REM activada genere un mensaje Flow-Removed en respuesta. En caso de eliminar una entrada de emergencia o de no haber hallado la entrada a eliminar, el switch se abstendrá de generar mensajes Flow-Removed.

Los comandos MODIFY y DELETE poseen versiones _STRICT. En los casos non-RESTRICT, los campos no especificados funcionan como comodines y todos aquellos flujos que coincidan con los campos especificados serán afectados. En los casos _STRICT, todos los campos inclusos aquellos con comodines y prioridades son estrictamente comparados, afectando sólo aquellos flujos que sean idénticos.

4.2.1.4 Read-State

En Read-State encontramos mensajes que permiten obtener las estadísticas que recoge el switch a partir de flujos, tablas y puertos (STATS_REQUEST y STATS_REPLY).

4.2.1.5 Packet-Out

Estos mensajes son usados por el controlador para enviar paquetes por un puerto específico (PACKET_OUT) y poder reenviar paquetes recibidos a través de mensajes Packet-In. Deben contener el paquete completo o el ID del buffer referenciando el espacio de almacenamiento del paquete, de igual manera debe contener una lista de acciones a ser aplicada ya que en caso de recibir una lista de acciones vacía el paquete será descartado.

4.2.1.6 Barrier

Los mensajes de solicitud/respuesta del tipo Barrier son usados para revisar que las dependencias de mensajes especificados hasta ahora hayan sido cumplidas (BARRIER_REQUEST) o para recibir notificaciones de operaciones completadas (BARRIER_REPLY) [14]. El mensaje funciona como un punto de sincronización generado por el controlador.

4.2.1.7 Role-Request

Los mensajes Role-Request tienen la finalidad de definir o solicitar el rol que tendrá el canal OpenFlow dentro de una estructura multi-controlador.

4.2.1.8 Asynchronous-Configuration

El controlador tiene la capacidad de añadir o solicitar filtros adicionales en los mensajes asíncronos que recibe a través del canal OpenFlow.

4.2.2 Simétricos

Los mensajes simétricos son mensajes enviado sin previa solicitud, en ambas direcciones. Estos pueden ser:

- HELLO.
- ECHO.
- Experimenter.

4.2.2.1 HELLO

Los mensajes HELLO son intercambiados entre el switch y el controlador al establecimiento de la conexión [14]. Es un mensaje utilizado para la comunicación de versiones, pues ambos deben indicar la versión soportada del protocolo más alta posible. Si la negociación falla, se envía un mensaje Error.

4.2.2.2 ECHO

Los mensajes de solicitud/respuesta del tipo Echo (ECHO_REQUEST y ECHO_REPLY) pueden ser enviados en dirección al switch o el controlador, en cualquiera de los casos, debe cumplirse el intercambio de mensajes. Estos pueden indicar latencia, ancho de banda o incluso funcionar como una señal de vida para la conexión controlador-switch.

4.2.2.3 Experimenter

Los mensajes Experimenter proveen una manera estandarizada de que switches OpenFlow ofrezcan funcionalidades adicionales, dentro del espacio permitido para futuras revisiones de OpenFlow.

4.2.3 Asíncronos

Los mensajes asíncronos son enviados al controlador sin haber sido previamente solicitados, ya que estos comunican la llegada de paquetes, cambios de estados o incluso errores. Están enumerados en la siguiente lista:

- Packet-In.
- Flow-Removed.
- Port-Status.
- Error.

4.2.3.1 Packet-In

Hay casos en que los paquetes recibido en el switch no concuerdan con alguna entrada de la tabla de flujos (fallo de tabla), o los paquetes son reenviados al controlador, para estos casos se envía un mensaje Packet-In (PACKET_IN) al controlador. Si el switch posee suficiente memoria, el switch encapsula una fracción de la cabecera del paquete (128 bytes, modificable) y el ID del buffer para enviarlos al controlador. Cuando no queda espacio disponible para el buffer, el switch reenvía el paquete completo. En el primer caso, el controlador generalmente se ve en la posibilidad de responder con mensajes Packet-Out. Hay

casos especiales como la revisión de campos TTL, que puede ocasionar el envío de mensajes Packet-In.

4.2.3.2 Flow-Removed

En cada ocasión que una entrada de flujo es añadida al switch, se especifica un *Timeout*, de igual manera que se especifica un tiempo de vida para el flujo. Si alguno de estos tiempos vence o el controlador lo solicita, se remueve el flujo. El controlador puede remover un flujo activamente enviando un mensaje Modify-State con subtipo DELETE [14].

En caso de que un flujo sea removido, el switch se ve en la necesidad de buscar la presencia de la bandera OFPFF_SEND_FLOW_REM para notificarle al controlador con un mensaje Flow-Removal (FLOW_REMOVED) que el evento ocurrido con una descripción detallada de la entrada. Esto especificado en el mensaje Modify-State subtipo ADD y DELETE.

4.2.3.3 Port-Status

En ocasiones los puertos del switch OpenFlow cambian su configuración (deshabilitados por el usuario o Spanning Tree), cuando esto suceda se espera un mensaje Port-Status (PORT_STATUS) dirigido al controlador.

4.2.3.4 Error

Este mensaje ERROR es utilizado por el switch para notificarle al controlador de problemas con la interpretación de los campos *type* y *code*.

4.3 Casos de Uso

A continuación, se explicarán algunos casos de uso relevantes en el contexto que ha traído el uso de OpenFlow con el pasar del tiempo. Extraídos de [12]:

4.3.1 Administración de Red y Control de Acceso

OpenFlow puede usarse como Ethane [20] para permitirle al administrador de red definir políticas de red en el controlador central, con las que serán influenciadas las decisiones de admisión para cada flujo nuevo. El controlador revisaría cada flujo contra un grupo de reglas como “Visitantes pueden comunicarse usando HTTP (Hypertext Transfer Protocol), pero sólo vía proxy web” o “Teléfonos VoIP no tienen permitido comunicarse con laptops” (Voice over IP). El controlador asocia los paquetes con sus emisores manejando todos los enlaces entre nombres y direcciones, prácticamente se apropia de protocolos como DNS (Domain Name System) [21], DHCP (Dynamic Host Configuration Protocol) [22] y autentica todos los usuarios, manteniendo la pista de los puertos a los que están conectados [23].

4.3.2 VLAN

OpenFlow provee la capacidad de manejar redes aisladas al igual que VLAN. La manera más sencilla sería declarar flujos que especifican los puertos accesibles del tráfico de paquetes dado un VLAN ID. El tráfico identificado sería marcado por los switches a través de una acción con el VLAN ID apropiado. Sin embargo, una

opción más dinámica podría involucrar el uso del controlador para manejar la autenticación de usuarios y usar la ubicación de los usuarios para etiquetar el tráfico en tiempo de ejecución [23].

4.3.3 Clientes VoIP Móvil Inalámbrico

Para ilustrar este caso se considera el experimento de un nuevo mecanismo de llamada manos-libres para teléfonos con WiFi. En el experimento los clientes VoIP establecen una nueva conexión sobre una red OpenFlow y el controlador esta implementado para rastrear la localización de los clientes, re-enrutando las conexiones a través de la misma reprogramación de las tablas de flujos mientras los usuarios se mueven alrededor de la red [23].

4.3.4 Red No-IP

Los ejemplos anteriores asumen el uso de redes IP, pero OpenFlow no necesita que los paquetes tengan algún formato especial siempre que las tablas de flujos puedan comparar los campos de la cabecera del paquete. Esto permite la experimentación con nuevos esquemas de direccionamiento, enrutamiento y nombramiento. Por ejemplo, un flujo puede identificarse a través de la dirección MAC usando un nuevo valor en el campo EtherType o a nivel IP, con una nueva versión IP. Esto quiere decir que se espera en un futuro, switches que permitan máscaras genéricas hechas por un controlador [23].

4.3.5 Procesamiento de Paquetes en Lugar de Flujos

Los ejemplos anteriores asumen la abstracción de flujos, donde la toma de decisiones realizadas por el controlador toma medidas cuando el flujo inicia. Existen también, experimentos interesantes que requieren el procesamiento de todos los paquetes antes de ser descartados. Por ejemplo, un sistema de detección de intrusos, un mecanismo de control de congestión, o la conversión del formato de un protocolo a otro. La manera más simple de conseguir el procesamiento de paquetes es forzar el envío de paquetes al controlador habilitando en el switch el reenvío de paquetes al controlador por defecto. Es flexible, sin embargo, permanente, trayendo consecuencias al implementarse en grandes redes. La segunda manera es enrutar los paquetes a un switch programable como NetFPGA [24] que pueda ser anexado al switch OpenFlow [23].

5. Controladores SDN

El controlador es el elemento principal de una red SDN y se considera como el sistema operativo de la misma. El controlador centraliza toda la comunicación que pasa por los dispositivos y provee una visión general de la red.

Una descripción general de un controlador SDN, sería: es el sistema de software o colección de sistemas que juntos proveen de [10]:

- Un manejo del estado de la red, y en algunos casos, la distribución de este estado, puede implicar una base de datos. Esta base de datos sirve como repositorio de información derivada del control de los elementos de red y software relacionados como información controlada por las aplicaciones SDN incluyendo el estado de la red, topología aprendida y controles de sesión de información.
- Un modelo de datos de alto nivel que captura la relación entre los recursos manejados, políticas y otros servicios provistos por el controlador. En muchos casos, estos modelos e datos son construidos usando Yang, el cual es un lenguaje de modelado para el protocolo de configuración de red o NETCONF (Network Configuration Protocol). Este lenguaje está definido en el RFC 6020 [25] de la IETF.
- Una interfaz de programación de aplicaciones la cual expone los servicios ofrecidos por el controlador para una aplicación. Esta facilita la interacción aplicación-controlador y a través del modelo de datos se describen los servicios ofrecidos por el controlador y sus características. En algunos casos, el controlador y estas APIs son parte de un mismo entorno de desarrollo que genera el código API directamente del modelo de datos. Algunos sistemas van más allá y proveen robustos entrenos de desarrollo que permiten la expansión de las capacidades de núcleo del mismo y subsecuentemente publican nuevos módulos de estas APIs, incluyendo así un soporte dinámico para la expansión de las capacidades del controlador.

El controlador SDN puede tener muchos significados diferentes y por lo tanto existen muchas diferentes formas de este actualmente. Gran parte de significado se deriva del dominio de red en el cual el controlador opera de las estrategias y el protocolo escogidos.

En el caso de este estudio, se utilizará el protocolo OpenFlow del que se habla en el Capítulo 4 para la comunicación entre el controlador y los switches, ya que el mismo es estándar entre muchos controladores, y es apoyado por la ONF.

5.1 Historia de Controladores

Actualmente se está llevando a cabo una batalla entre las diferentes empresas que proveen soluciones de red, quienes quieren proveer su propio controlador SDN para promover su propio equipamiento de red. A su vez otras empresas que buscan la incorporación de todo posible equipo sin importar su marca o tipo, en busca de obtener un universo más amplio de potenciales clientes. Al igual que en

otros ámbitos del desarrollo de software, existen dos vertientes, el software libre o de código abierto y el propietario.

El primer controlador SDN desarrollado fue NOX, un trabajo realizado inicialmente por Nicira Networks, separadamente de OpenFlow. En 2008 Nicira Networks, que fue comprada por VMWare, el cual donó NOX a la comunidad SDN, volviéndose así un software de código abierto. NOX se convirtió en la base de muchas soluciones de controladores SDN actuales. Entonces Nicira comenzó a desarrollar ONIX en cooperación con NTT Communications y Google. ONIX es la base para el controlador Nicira de VMWare y también base del controlador WAN propietario de Google, el cual uso para la migración de su red convencional a una red SDN que se concretó el año 2010.

Es aquí, entonces, cuando surge una variedad de controladores SDN de código abierto, iniciando con uno llamado POX, basado en NOX, fue la base para la creación de Beacon el cual fue uno de los controladores más populares iniciando el año 2010. Beacon es un controlador escrito en Java y basado en OpenFlow. Otro controlador SDN de relevancia es Ryu apoyado por NTT (Nippon Telegraph and Telephone) y usado por la NSA (National Security Agency) para su migración a una red SDN. Tambien existe Floodlight que es un controlador basado en Beacon y de código abierto.

Subsecuentemente empresas como Cisco Systems, HP (Hewlett-Packard), IBM (International Business Machines Corp.), VMWare y Juniper lanzaron al mercado sus propios controladores SDN. Los controladores de HP, Cisco e IBM, basados en Beacon, posteriormente se unen para formar OpenDaylight uno de los controladores con mayor aceptación en la actualidad. El controlador SDN de Juniper se convirtió en parte de su repertorio de productos luego de la adquisición de Contrail, este está disponible en su versión código abierto y en versiones comerciales.

Actualmente existen muchos controladores SDN disponibles, lo cual hace ardua la tarea de la escogencia de un controlador en específico para la implementación de una red. Para esto se deben realizar pruebas y evaluaciones de desempeño. Aspectos de suma importancia son los servicios disponibles, la posibilidad de agregación de módulos que amplíen los servicios prestados y lo más importante su operatividad y desempeño. En la Figura 5.1, tomada de [26], se muestra una comparación de varios controladores, tomando en cuenta aspectos básicos que ofrecen.

	Beacon	NOX	POX	ODL	Floodlight	Ryu	OpenMUL
Soporte OpenFlow	OF v1.0	OF v1.0	OF v1.0	OF v1.3 y v1.4	OF v1.3 y v1.4	OF v1.0, v1.2, v1.3, v1.4 y extensiones Nicira	OF v1.0, v1.2, v1.3 y v1.4
Virtualización	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch	Mininet y Open vSwitch
Lenguaje de desarrollo	Java	C++	Python	Java	Java	Python	C, Python
Provee REST API	No	No	No	Sí	Sí	Sí (Básica)	Sí
Interfaz Gráfica	Web	Python+, QT4	Python+, QT4, Web	Web	Web	Web	Web
Soporte de plataformas	Linux, Mac OS, Windows y Android	Linux	Linux, Mac OS, Windows	Linux, Mac OS, Windows	Linux, Mac OS, Windows	Linux	Linux, Mac OS, Windows
Soporte de OpenStack	No	No	No	Sí	Sí	Sí	Sí
Multiprocesos	Sí	Sí	No	Sí	Sí	No	Sí
Código Abierto	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Tiempo en el mercado	6 años	8 años	3 años	3 años	4 años	3 años	3 años

Tabla 5.1: Características de Controladores SDN

En las siguientes secciones se realizará una descripción de cuatro controladores los cuales serán utilizados durante las pruebas de la herramienta de medición de desempeño propuesta. Los controladores escogidos para su evaluación son OpenDaylight, Floodlight, Ryu y OpenMUL, la elección fue basada en la competitividad comercial, el uso actual en proyectos en desarrollo o en redes ya establecidas para producción y en su proyección a futuro.

5.2 OpenDaylight

OpenDaylight u ODL es un proyecto colaborativo de código abierto, llevado por Linux Foundation en colaboración con numerosas empresas del área de computación y redes tales como Cisco Systems, Dell, Ericsson, HP, Intel, etc. ODL puede ser el componente central de cualquier arquitectura SDN, permitiéndole a los usuarios reducir la complejidad operacional, extender el tiempo de vida de las redes existentes y habilitar nuevos servicios y capacidades únicamente disponibles con SDN.

El principio fundamental del proyecto OpenDaylight es facilitar una comunidad soportada por la industria, que ofrezca un framework abierto, incluyendo código y arquitectura, para acelerar el avance hacia una plataforma SDN común y robusta. De tal forma OpenDaylight permite que cualquiera contribuya con código para el proyecto, mientras este pase las pruebas del TSC (Technical Steering Committee).

El proyecto OpenDaylight es uno de los más recientes. ODL tiene varias distribuciones, la más actual disponible es la versión Beryllium la cual está desarrollada con una serie de complementos cuyo objetivo es permitirnos interactuar con el controlador de forma más transparente. De esta forma se trabaja con Karaf; una plataforma genérica que proporciona funciones y servicios de alto nivel diseñados específicamente para la creación de servidores basados en OSGi (Open Service Gateway Initiative). OSGi permite la programación en Java e instalar, arrancar o detener paquetes Java de manera sencilla. Esta distribución viene con una serie de proyectos incluidos que son fáciles de instalar gracias a Karaf.

Por defecto, ODL no tiene funcionalidades específicas instaladas, luego de la instalación del controlador, son instaladas las funcionalidades utilizando la consola Karaf aquellas seleccionadas por el usuario para cumplir con sus necesidades y requerimientos de red, Karaf ofrece una lista de funcionalidades que se adecuan a la mayoría de los ambientes de red. OpenDaylight ofrece las siguientes diferencias con las otras opciones de controladores SDN:

- Una arquitectura basada en micro servicios, es una serie de protocolos o servicios en particular que el usuario puede habilitar durante la instalación del controlador, por ejemplo; un plugin que proporciona conectividad a los dispositivos a través de OpenFlow o protocolos BGP (Border Gateway Protocol), un servicio AAA (Authentication Authorization and Accounting), etc.
- Soporte para una amplia y creciente gama de protocolos de red más allá de OpenFlow incluyendo SNMP (Simple Network Management Protocol), NETCONF, OVSDB (Open vSwitch Database Management), BGP, PCEP (Path Computation Element Communication Protocol), LISP, y más.
- Soporte para el desarrollo de nuevas funcionalidades abarcando protocolos y servicios de red adicionales.

5.2.1 Arquitectura

El controlador ODL es una infraestructura de alta disponibilidad, extensible, escalable y multiprotocolo, es un software que corre en la máquina virtual de Java lo que le otorga la capacidad de poder ejecutarse en cualquier sistema operativo y hardware que tenga soporte de Java. En la Figura 5.1, tomada de [27], se muestra gráficamente como es la estructura del controlador ODL.

En el southbound, en la frontera con los elementos de red, tenemos que puede trabajar con múltiples protocolos, tales como OpenFlow 1.0 y 1.3, BGP-LS, etc. Otra característica importante de su arquitectura es la capa de abstracción de servicio o SAL (Service Abstraction Layer). Esta capa SAL expone servicios a los módulos superiores, y cumple con los servicios solicitados, independientemente del protocolo subyacente que se utiliza entre el controlador y los dispositivos de red. Proporcionando así, protección a cambios de protocolo o de versiones del mismo protocolo con el tiempo.

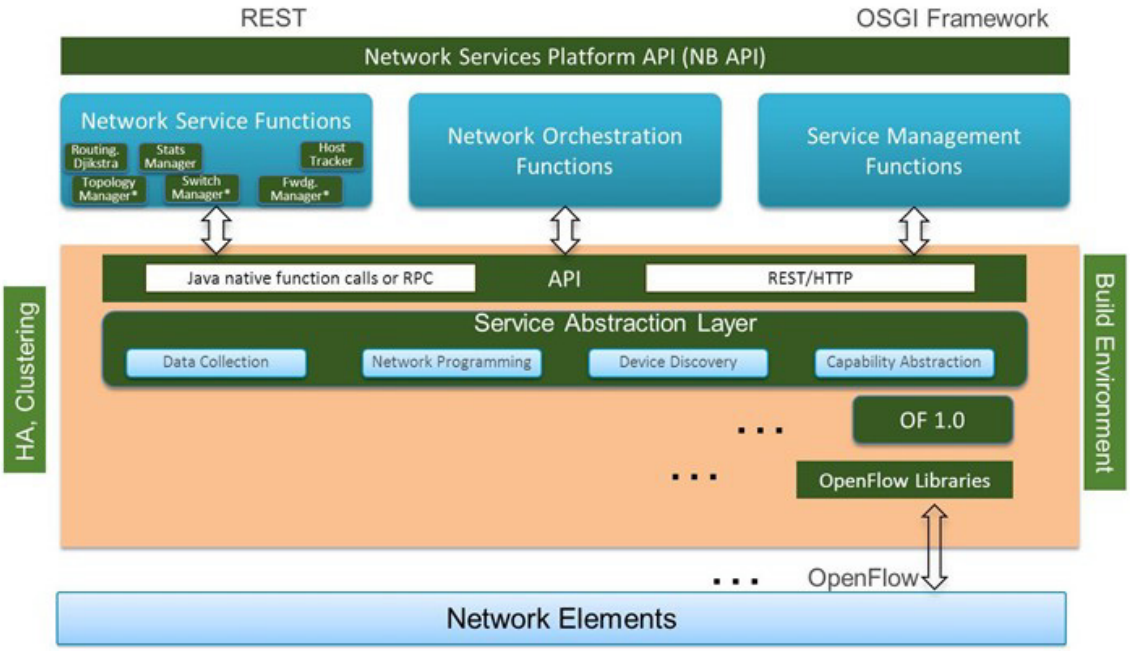

Figura 5.1: Arquitectura de OpenDaylight

Para poder ODL controlar los dispositivos de red en su dominio, debe mantener datos de los dispositivos, tales como, sus capacidades, accesibilidad, etc. Esta información es almacenada y manejada por el Topology Manager. Los otros componentes como Host Tracker, Device Manager y switch Manager, ayudan a la generación de bases de datos de la topología para el Topology Manager.

El controlador expone APIs hacia el northbound y el framework OSGi a disposición de las aplicaciones, incluso aquellas que se deseen ejecutar en el mismo espacio de direcciones que el controlador ODL. Mientras que el framework REST (Representational State Transfer), basado en web, es usado para aplicaciones que no se ejecutan en el mismo espacio de direcciones, o incluso en la misma máquina.

La lógica de negocio y algoritmos residen en las aplicaciones. Estas aplicaciones utilizan el controlador para reunir información de la red, y posteriormente siguen su algoritmo propio para analizar los datos obtenidos y luego usar el controlador para orquestar nuevas reglas en toda la red de su dominio.

La capa de abstracción del servicio (SAL) es el corazón del diseño modular del controlador ODL, permitiéndole dar soporte a múltiples protocolos en el southbound y provisionando de un servicio de consistencia para módulos y aplicaciones. En la Figura 5.2, tomada de [27], se muestra gráficamente la estructura de esta capa. La SAL provee servicios básicos como Device Discovery, los cuales son usados por módulos como Topology Manager para construir la topología de red y las capacidades de cada dispositivo.

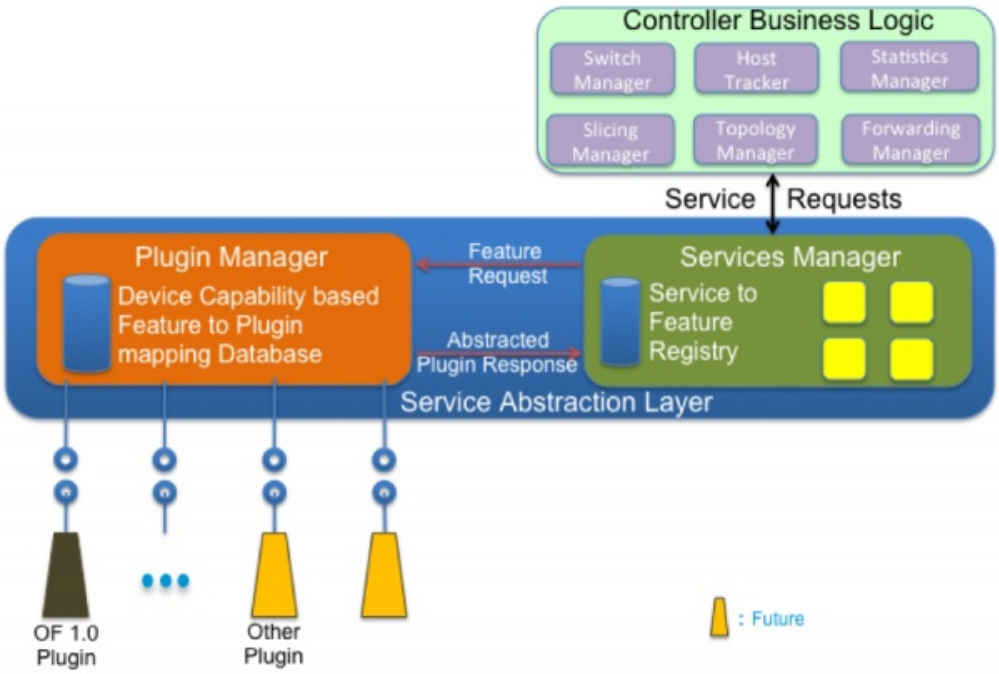

Figura 5.2: Capa de Abstracción de Servicio

Los servicios son conformados con las características y capacidades de los dispositivos, expuestas por los plugins. En base a la solicitud de servicio, la SAL se correlaciona con el plugin adecuado, entonces usa el protocolo de southbound más apropiado para la comunicación con el dispositivo de red en cuestión. Cada plugin es independiente de los otros y están débilmente acoplados con la SAL.

La SAL evolucionó hacia un enfoque basado en modelos, en donde se proporciona un framework para modelar la red, sus propiedades y los dispositivos de red de forma dinámica, para posteriormente establecer una relación servicios/aplicaciones utilizando las APIs del northbound, y plugins, protocolos proporcionados por el southbound. La Figura 5.3, tomada de [27], muestra cómo los plugins proporcionan las partes del árbol del modelo de red global en la capa de abstracción. Mientras que Figura 5.4, tomada de [27], muestra como las aplicaciones pueden acceder a la información existente en el modelo de red, a través de los APIs existentes en el northbound.

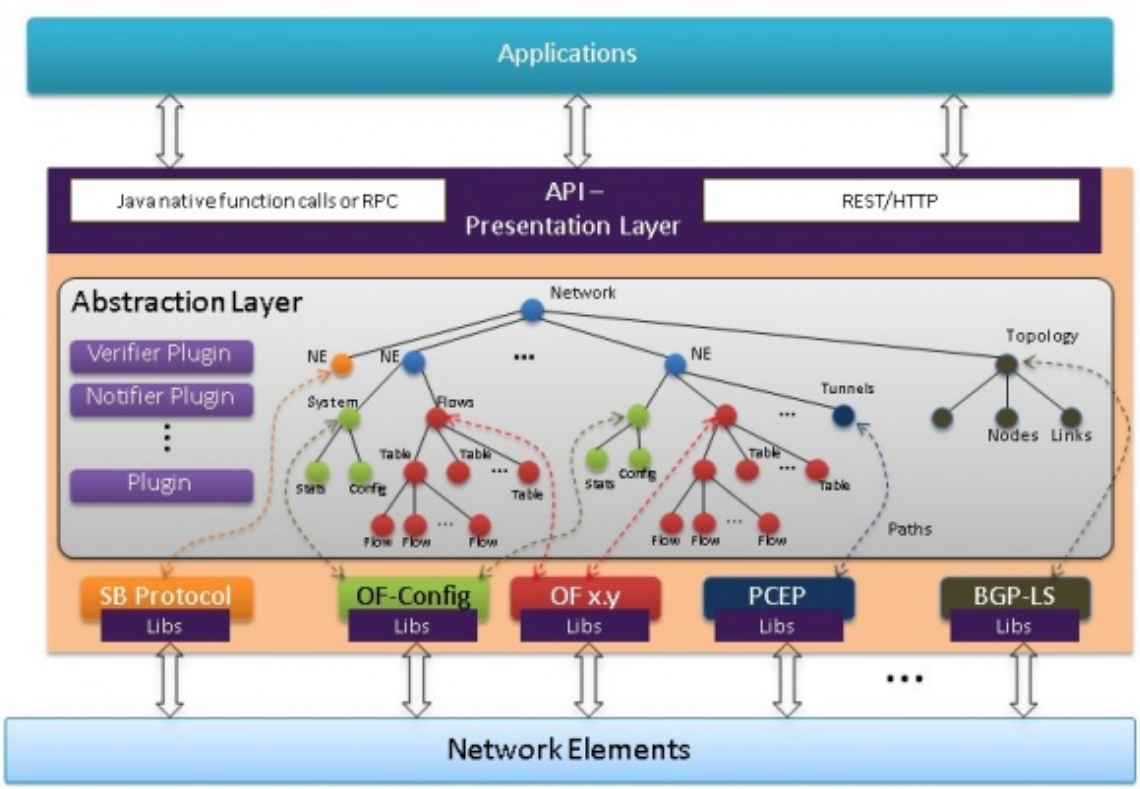

Figura 5.3: Formación del Árbol de Modelo de Red

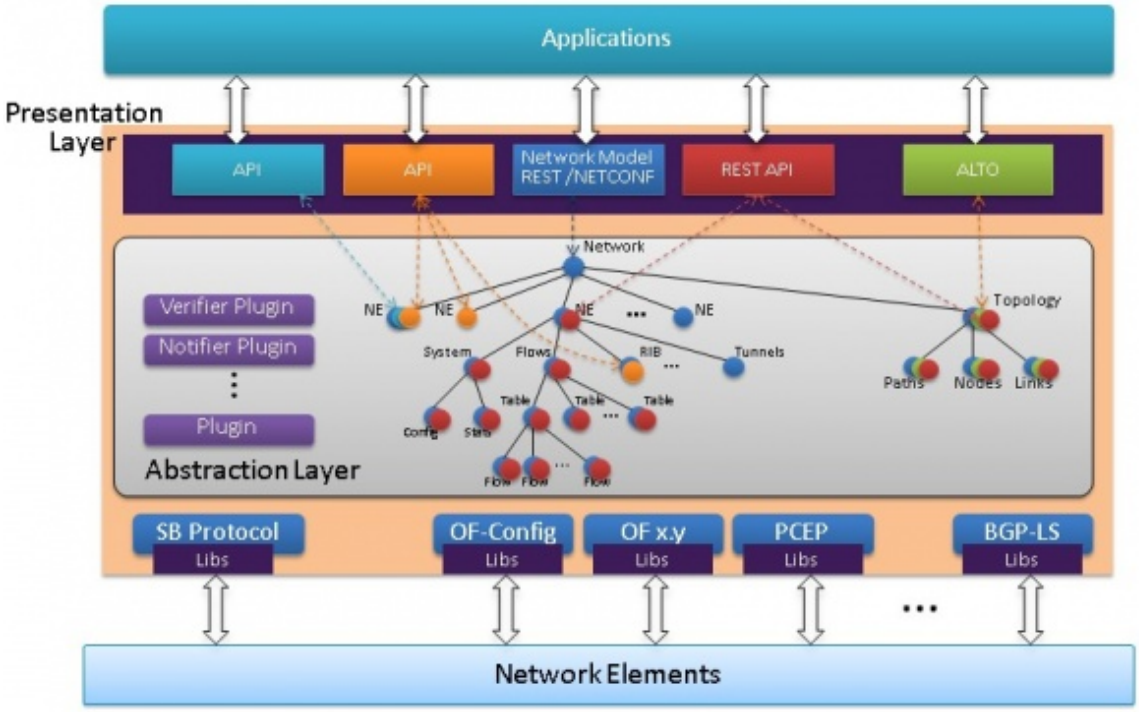

Figura 5.4: Acceso a la Información del Modelo de Red

5.2.1.1 Capacidades del Northbound

OpenDaylight ofrece un amplio conjunto de APIs disponibles en el northbound del controlador, en su mayoría concebidas para el acceso, configuración, monitoreo y control de los servicios de red básicos del controlador representados en la Figura 5.1.

Es importante destacar que el framework de este controlador es abierto y flexible para permitir la extensión de las funcionalidades del controlador ya existentes o para crear nuevas APIs REST que expongan nuevos servicios de red definidos por el usuario. La Tabla 5.2, tomada de [28], lista un resumen de las APIs REST disponibles en el northbound del controlador OpenDaylight.

Grupo	API del Northbound	Descripción
Funciones de Gestión	Host Tracker REST APIs	APIs utilizadas para realizar un seguimiento de hosts y ubicarlos en la red subyacente. La ubicación del host es representada lógicamente como una entidad que puede ser un switch o un puerto.
	Statistics REST APIs	APIs que proporcionan un conjunto de información y estadísticas expuestas suministradas por los plugins de protocolos del southbound, tal como la de OpenFlow. Información y estadísticas tales como número de paquetes procesados, estado de puertos, tablas de flujos, etc.
	User Manager REST APIs	Esta API proporciona primitivas para gestionar los usuarios que se registrarán en el controlador.
	Connection Manager REST APIs	API que permite gestionar las conexiones con los nodos bajo el control de OpenDaylight.
Control de Nodos y Flujos	Container Manager REST APIs	Esta API proporciona primitivas para crear, eliminar y gestionar los nodos de la red. Esto se realiza a través de la gestión de contenedores que representan grupos de estos nodos.
	Topology REST APIs	Estas APIs dan acceso a la topología almacenada en el controlador, previamente generada y actualizada por el módulo Topology Manager.
	Static Routing REST APIs	Esta API es para la gestión de rutas estáticas L3 de la red subyacente.
	Subnets REST APIs	Esta API permite el manejo de las subredes L3 de un contenedor dado.
	Switch Manager REST APIs	Esta API permite el acceso a un nodo especifico, para la visualización de sus características, conexiones y capacidades.
	Flow Programmer REST APIs	Esta API provee la capacidad de programar los flujos en la red OpenFlow subyacente.
Integración Open vSwitch	Bridge Domain REST APIs	Esta API del northbound da acceso a algunas primitivas del protocolo OVSDB para programar un Open vSwitch. Por ejemplo, para gestionar bridges virtuales.
Integración OpenStack	Neutron / Network Configuration APIs	Esta API adapta la API Neutron para proporcionar una integración estable del controlador OpenDaylight con OpenStack.

Tabla 5.2: APIs REST Disponibles en el Northbound de OpenDaylight

5.3 Ryu

El término “Ryu” significa *flujo* o *dragón japonés*, en este caso, Ryu es usado para manejar el control de flujos que permita redes inteligentes [29] como framework. Es conocido como un framework de código abierto, pero también lo es como un software llamado NOS (Network Operating System) que no sólo soporta el manejo de dispositivos OpenFlow, sino que de igual manera es capaz de trabajar con dispositivos NETCONF, OVSDB, xFlow, VRRP (Virtual Router Redundancy Protocol) y SNMP.

Ryu posee un diseño inspirado en la deconstrucción de controladores monolíticos, con una arquitectura basada en componentes que ofrece una mayor escalabilidad al costo de eficiencia al ser implementado en el lenguaje de programación Python. Para nuestro beneficio Ryu soporta una variedad de versiones OpenFlow de la cual nos estaremos aprovechando, dicho esto las versiones son comprendidas en el rango [1.0, 1.4].

Ryu tiene como objetivos: convertirse en el controlador de red por defecto para distribuciones Linux como RHEL (Red Hat Enterprise Linux), Fedora o Ubuntu, y ser el controlador de red estándar de orquestadores de nube como OpenStack.

5.3.1 Arquitectura

Idealmente Ryu fue diseñado para explotar la mayor cantidad de servicios ofrecidos (firewall, switch L2, controlador OpenFlow, ver Figura 5.5, tomada de [29]) con enlaces de comunicación RESTful.

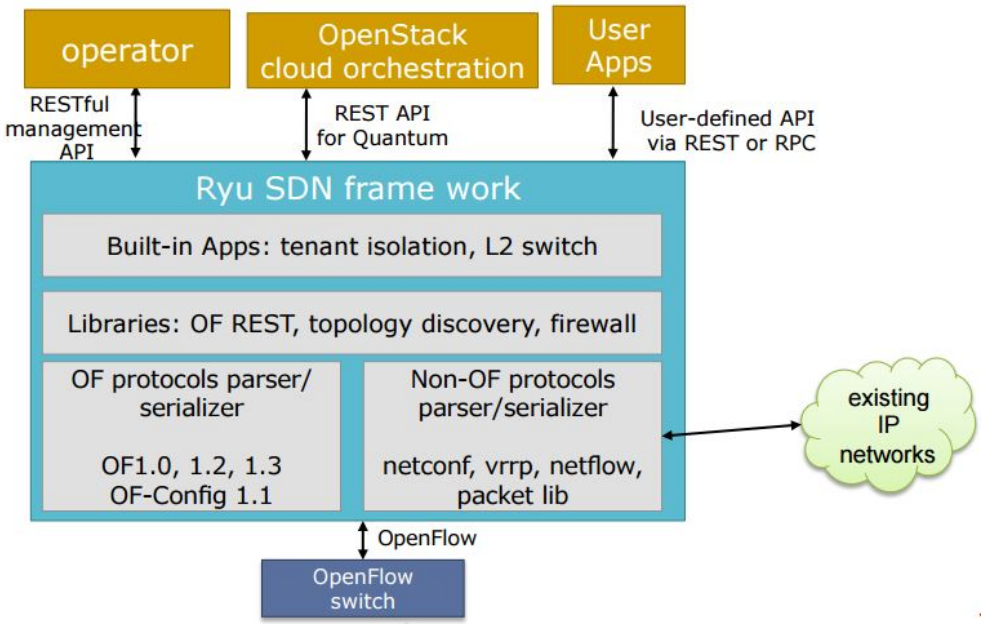

Figura 5.5: Arquitectura de Ryu

Las aplicaciones Ryu son entidades que implementan múltiples funcionalidades de Ryu basadas en componentes y librerías, como el controlador OpenFlow,

visualizador de topología, switch L2, firewall, soporte RESTful para operaciones OpenFlow, etc. El framework ofrece un grupo útil y amplio de estos componentes, componentes modificables y agrupables capaces de comunicarse con nuevos componentes. La creación de nuevos componentes no ata el componente a un sólo lenguaje ya que la comunicación es por medio de mensajes estándar.

5.3.2 Componentes

Un componente es una unidad aislada que provee una interfaz de control y estado, capaz de generar eventos y comunicarse con otros componentes a través de ellos. La comunicación de estos componentes se realiza con pase de mensajes [30].

Los eventos se agrupan en una cola administrada por la aplicación. La cola es de comportamiento FIFO (First In First Out), es decir, se conserva el orden de llegada de los eventos para su posterior procesamiento. Existe un hilo para el procesamiento de eventos, que está en la potestad de recibir los eventos encolados y desencolarlos una vez se ha despachado el manejador correspondiente. Estos manejadores son despachados en el mismo contexto del hilo, de aquí el que se infiera tener cuidado con el bloqueo. Los componentes pueden consistir de hilos Python o procesos del sistema operativo. En la Figura 5.6, tomada de [30], puede observar una lista de ellos.

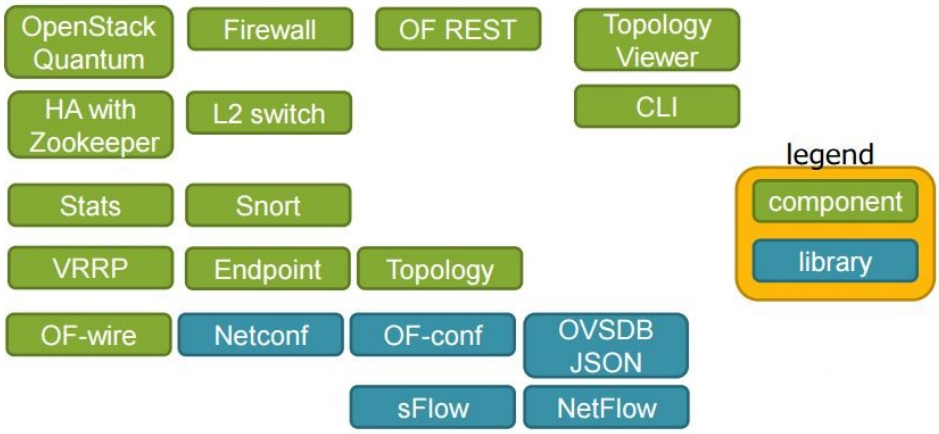

Figura 5.6: Lista de Componentes y Librerías

- **OF-Wire:** Provee las definiciones correspondientes a las versiones OpenFlow 1.0, 1.1, 1.2, 1.3, 1.4, codifica y decodifica los mensajes del protocolo OpenFlow, y las extensiones Nicira.
- **Topology:** Permite la construcción de topologías y rastreo de enlaces.
- **VRRP:** Provee soporte del protocolo VRRPv3 a switches OpenFlow.
- **OF REST:** Provee la capacidad de configurar switches OpenFlow usando APIs REST, ver Figura 5.7, tomada de [29].

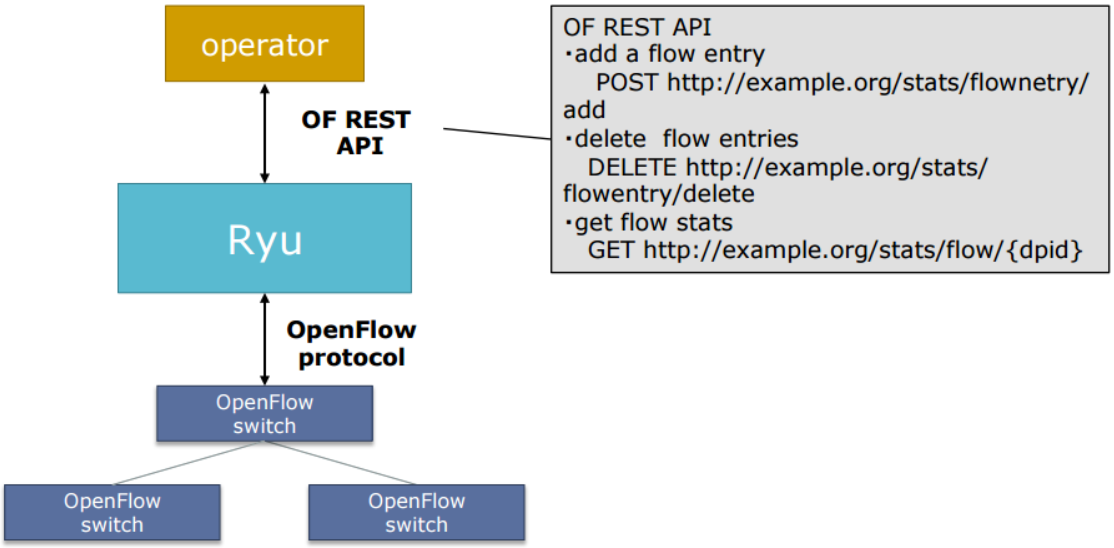

Figura 5.7: Configuración OpenFlow vía API REST

- **OpenStack Quantum:** Permite aislar entidades de red como un servicio (NaaS), de dos maneras, VLAN o túnel GRE. OpenStack Quantum posee un plugin para comunicarse con el uso de un API REST a aplicaciones Ryu, ver Figura 5.8, tomada de [30].

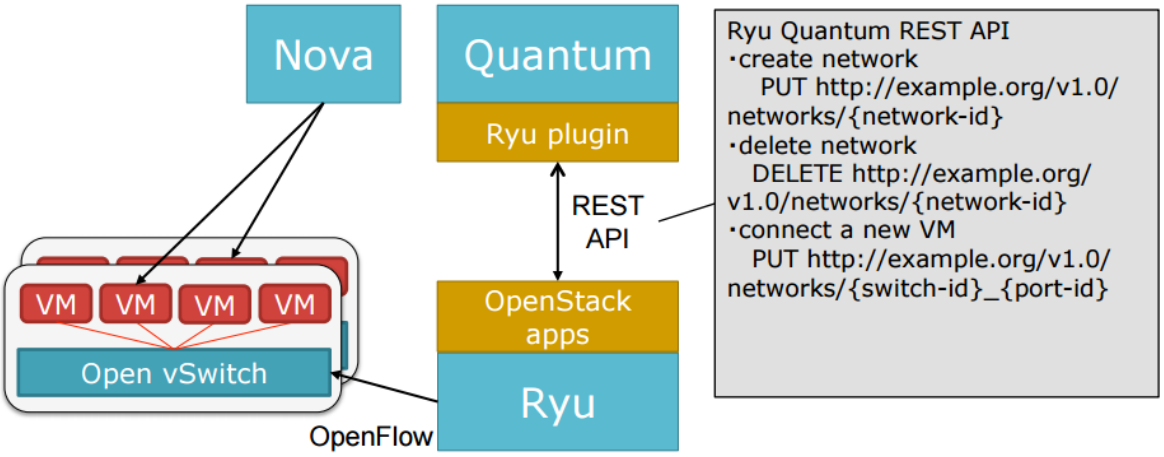

Figura 5.8: Comunicación Ryu - OpenStack Quantum

- **Stats:** Provee análisis y visualización de estadísticas en puertos, ver Figura 5.9, tomada de [30].

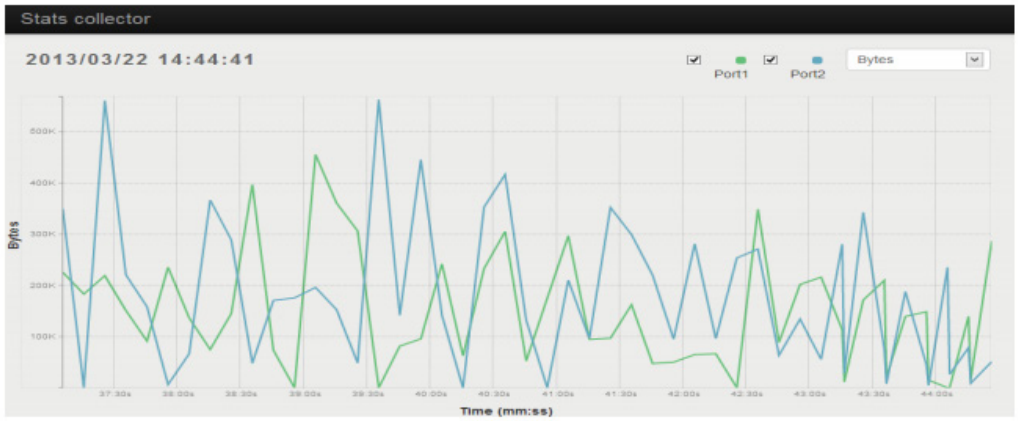

Figura 5.9: Visualización de Estadísticas

- **Topology Viewer:** Permite visualizar la topología y flujos dinámicamente, ver Figura 5.10, tomada de [29].

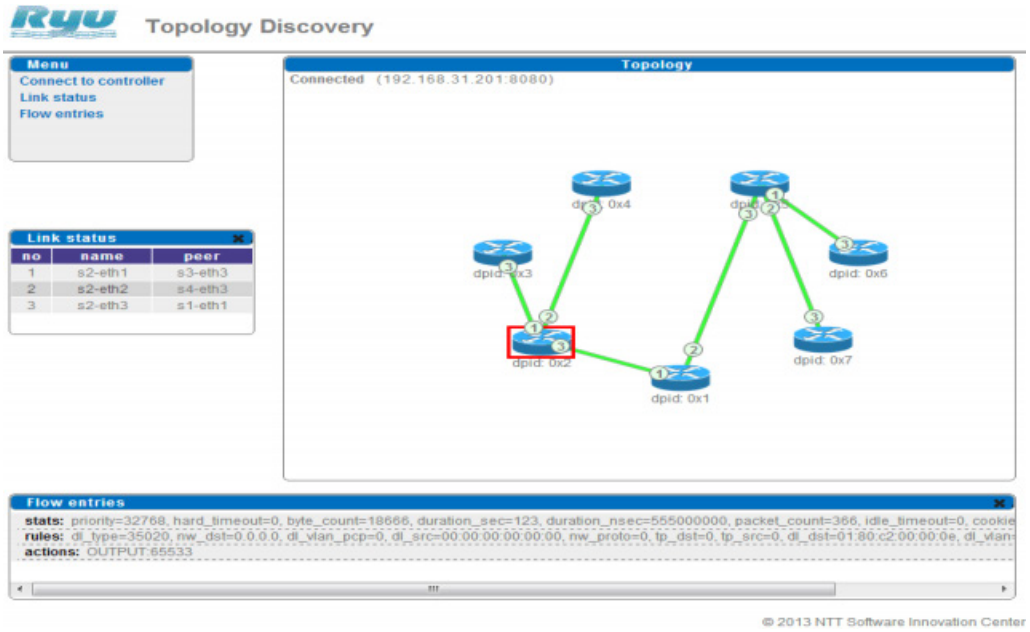

Figura 5.10: Visualización de Topología

- **HA with Zookeeper:** Provee medios para evitar puntos de falla centralizados. Ya que un controlador puede ser un punto peligroso para la red en caso de falla, ZooKeeper permite alta disponibilidad (High Availability también conocido como HA) con el manejo de puntos redundantes, ver Figura 5.11, tomada de [29].

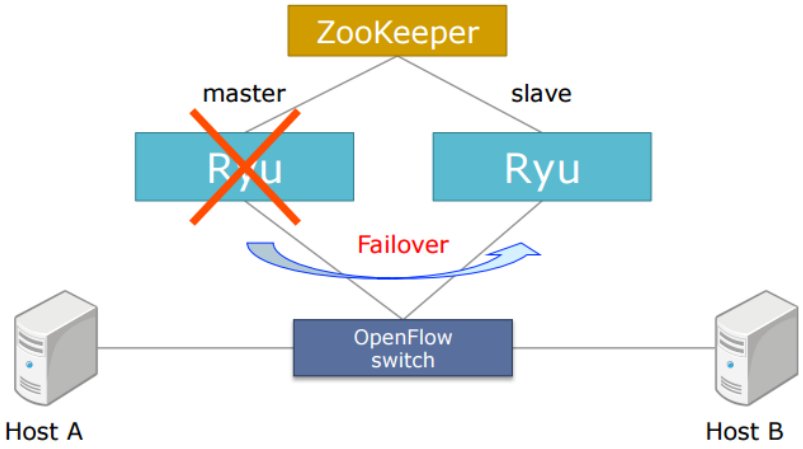

Figura 5.11: Manejo de Redundancia con ZooKeeper

- **Firewall:** Permite el manejo de reglas Firewall con el uso de API REST.
- **Snort:** Permite integrarse con sistemas de detección de intrusos para la notificación de amenazas en tiempo real, ver Figura 5.12, tomada de [29].

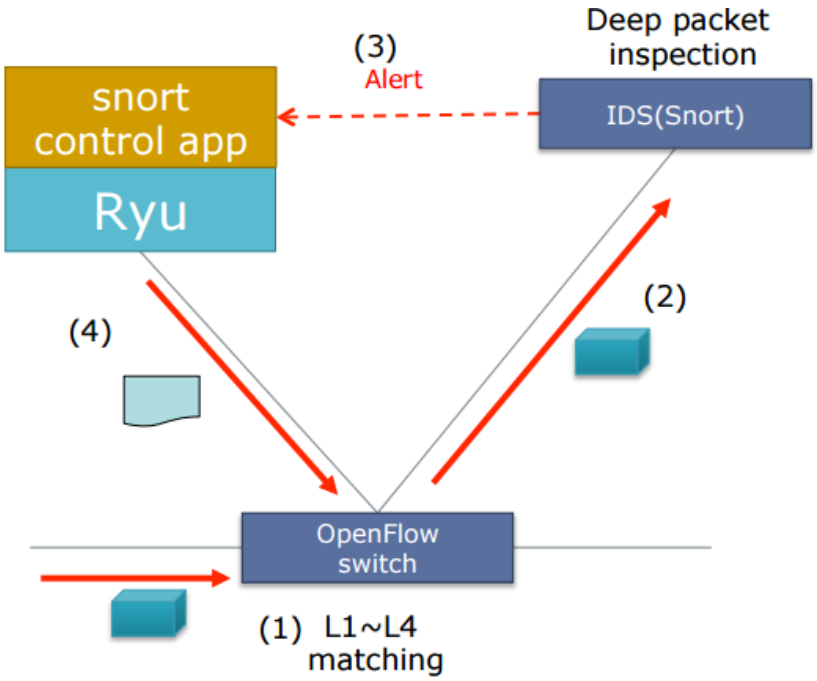

Figura 5.12: Detección de Intrusos con Ryu

- **CLI:** Permite la administración vía comandos de terminal.
- **L2 Switch:** Provee soporte a la implementación de switches L2.

Las librerías por otra parte son extensiones con soporte a la construcción de paquetes de otra gran variedad de protocolos importantes.

5.4 Floodlight

Floodlight está basado en Beacon, que es un controlador SDN desarrollado por David Erickson de la Universidad de Stanford. Floodlight es un controlador OpenFlow escrito en Java con licencia Apache que está soportado por una comunidad de desarrolladores, incluyendo algunos ingenieros de Big Switch Networks. Cualquier aplicación escrita para Floodlight por cualquier desarrollador, puede ser certificada y calificada por la compañía.

Floodlight es una colección de aplicaciones construidas encima del controlador mismo. El controlador realiza una serie de funcionalidades comunes para controlar e investigar la red OpenFlow, mientras que las aplicaciones realizan diferentes actividades para resolver requerimientos del usuario sobre la red.

Debido a que está escrito en Java, este corre en la máquina virtual de Java (JVM). Como otros controladores, Floodlight ofrece sus servicios a través del southbound y northbound. En el northbound las APIs son expuesta a todos los módulos que se encuentren en ejecución a través de un puerto REST, cualquier aplicación puede interactuar con el controlador enviando comandos HTTP-REST, permitiendo la solicitud de información o la invocación de servicios.

Por otro lado, en el southbound, el módulo proveedor de Floodlight inicia escuchando en el puerto TCP específico del protocolo OpenFlow 1.0/1.3/1.4, para iniciar conexiones con los switches. En la siguiente sección se ahondará más acerca de esta arquitectura.

5.4.1 Arquitectura

La arquitectura de Floodlight es modular, tal como se muestra en la Figura 5.13, tomada de [28]. Incorpora numerosos módulos de propósito específico que proveen la base para las funciones de red, tales como Topology Manager, Link Discovery, Device Manager, Performance Monitor, etc. Estos son necesarios para ofrecer los servicios a aplicaciones y tomar acciones en la red subyacente. Esta arquitectura comprende al controlador Floodlight, el cual ofrece servicios a través de 3 fronteras; al norte ofrece servicios a las aplicaciones, al sur controla los dispositivos de la red OpenFlow y en la frontera sur integra módulos extras para ampliar su funcionalidad.

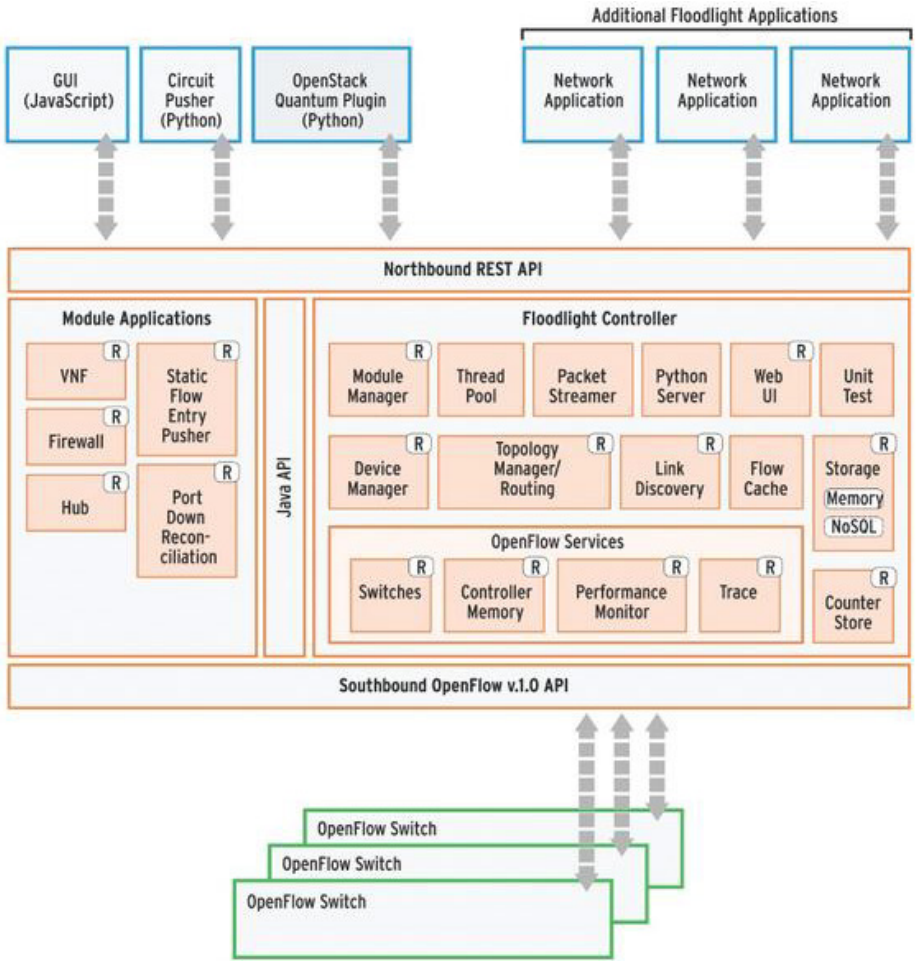

Figura 5.13: Arquitectura del Controlador Floodlight

En el westbound, la API Java permite el desarrollo de módulos modificados en lenguaje Java y su rápida interacción con el controlador central. Los módulos son cargados por un sistema durante el inicio del controlador e integrados al mismo, permitiendo así la comunicación con las funciones de red básicas y otorgando una respuesta rápida a eventos de la red, tales como la aparición de nuevos paquetes o flujos.

La API REST en el northbound permite la integración de aplicaciones externas en cualquier lenguaje a través de JSON. Comprado con la API Java comentada anteriormente, esta API es relativamente lenta. La reacción a eventos en tiempo real no es posible. Por lo tanto, la API REST ofrece un servicio de consultas e información de estatus, para una posible modificación de las reglas que controlan el protocolo OpenFlow.

Un ejemplo claro de estas aplicaciones que utilizan los servicios de la API REST, es la propia interfaz de usuario de Floodlight, mostrada en la Figura 5.14, tomada de [31]. En esta interfaz, se muestra la topología de la red, cuya información fue obtenida a través de un servicio abstraído por la API REST, aplicación que viene incluida en la instalación del controlador, y mostrada vía HTTP. Se puede ver la

MAC de cada uno de los dispositivos y su interconexión. Esta información no es mostrada en tiempo real y su actualización con respecto a servicios ofrecidos por el API de Java es lenta.

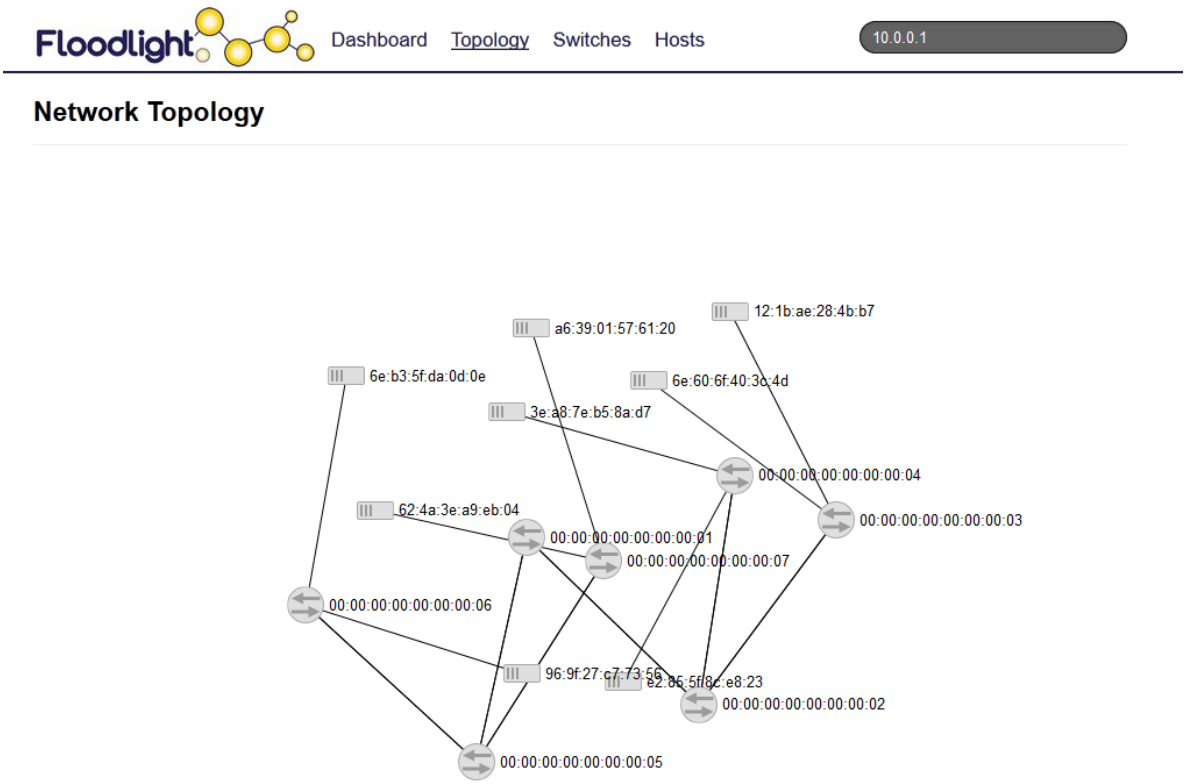

Figura 5.14: Topología Mostrada en Interfaz Web del Controlador Floodlight

Para la construcción de la topología mostrada anteriormente, el controlador Floodlight, basándose en la información del mecanismo de descubrimiento de links, ejecuta el servicio de topología, el cual procesa el mapa de la topología y lo almacena en forma de un grafo. Este grafo contiene toda la información de relevancia acerca de la interconexión de switches, los cuales pueden ser utilizados por otras aplicaciones.

5.4.2 Capacidades del Northbound

Floodlight expone una interfaz abierta REST en su northbound, la cual provee un conjunto de primitivas para acceder a los servicios básicos de red implementados en el controlador, estas primitivas están relacionadas a los módulos de servicio de aplicaciones (todos los módulos marcados con R en la Figura 5.13). El framework de la API REST es extensible a dar soporte y exponer nuevos servicios e interfaces, los desarrolladores del proyecto proporcionan documentación para realizar este tipo de agregación de funcionalidades. En la Tabla 5.3, tomada de [28], se muestra un resumen de las APIs REST ofrecidas por el controlador.

Grupo	API del Nortblound	Descripción
Funciones de Gestión	Device Manager REST API	Esta API REST es usada para recuperar la lista de dispositivos registrados por el controlador Floodlight. La información recuperada incluye campos como dirección MAC, dirección IP y puntos de conexión.
	Statistics REST API	Esta API REST expone funcionalidades de seguimiento y proporciona información acerca del funcionamiento del controlador Floodlight, información tal como el uso de memoria del controlador, estado del a API, resumen del controlador, etc.
Control de Nodos y Flujos	Topology Service REST API	Esta API otorga acceso a la topología desarrollada por el servicio base Topology Manager, y es usada para recuperar información acerca de los switches, clusters y enlaces externos e internos.
	Static Flow Pusher REST API	Esta API permite la inserción manual de flujos dentro de la red OpenFlow, así como también su eliminación y actualización.
	Virtual Network Filter REST API	Esta API REST es usada para gestionar múltiples nodos de la red OpenFlow subyacente, permitiendo así crear, eliminar y actualizar redes virtuales y adjuntar o separar puertos y hosts.
Integración OpenStack	Neutron REST Proxy Plugin	Un plugin Neutron dedicado está disponible para Floodlight. Este plugin actúa como un traductor entre la API Neutron y la Virtual Network Filter REST API descrita arriba.
Otros	Firewall REST API	Esta API es expuesta por la aplicación Firewall Module y permite habilitar o deshabilitar esta aplicación, así como también crear, eliminar y actualizar las reglas de esta aplicación.

Tabla 5.3: Resumen de Funcionalidades de la API REST de Floodlight

5.5 OpenMUL

MUL significa *base* o *raíz* en Sanscrito. Es un controlador SDN de código abierto, desarrollado mayormente en C por la fundación OpenMUL con una infraestructura multi-hilo que promete sobresaltar en rendimiento, confiabilidad y capacidad de hospedar aplicaciones modulares. Provee una gran variedad de aplicaciones en sus interfaces northbound y es capaz de trabajar con múltiples protocolos southbound como OpenFlow 1.0/1.3/1.4, OVSDB, NETCONF, etc.

5.5.1 Arquitectura

OpenMUL posee una arquitectura modular con miras a proveer funcionalidades flexibles para la orquestación de redes a través de una interfaz usable con múltiples puntos de accesos. MUL posee múltiples APIs para una gama de lenguajes y RESTful APIs para aplicaciones web en el northbound. Los grupos de funcionalidades se pueden observar en Figura 5.15, tomada de [32].

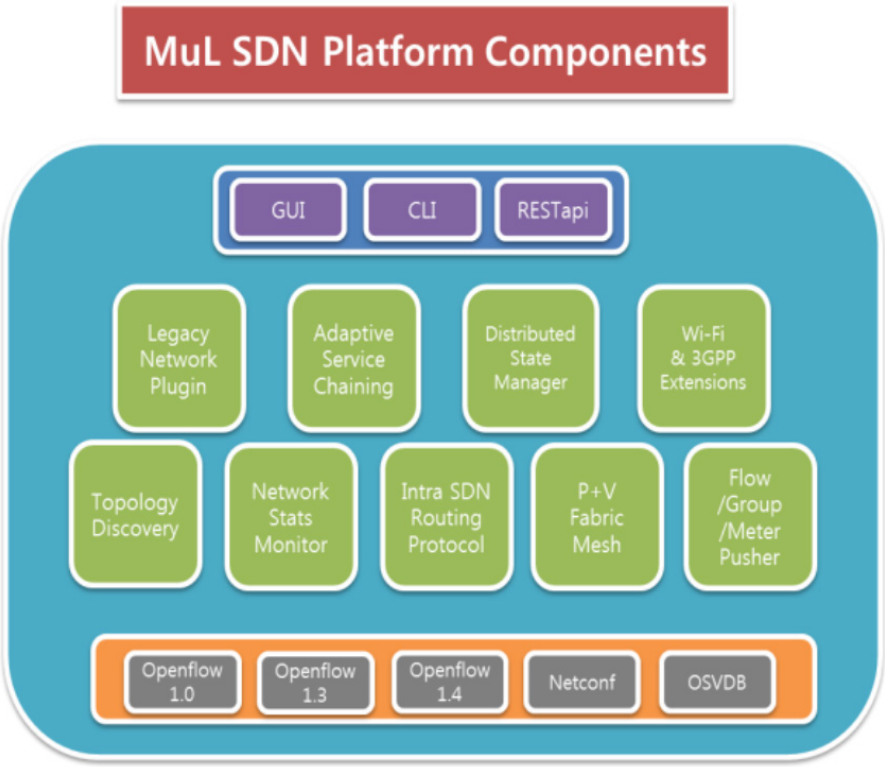

Figura 5.15: Arquitectura del Controlador OpenMUL

5.5.2 Componentes

Dividida en tres secciones, MUL combina componentes inherentes al núcleo del controlador, servicios y aplicaciones.

La mayoría de los controladores tiene inconvenientes con la sostenibilidad del rendimiento a ofrecer una variedad de servicios en forma de aplicaciones, esto se debe a que ubican estos servicios en el mismo espacio de direcciones. Sin embargo, los servicios de MUL residen como entidades independientes en una arquitectura distribuida, interconectada a través de MLAPIs (Medium-Level Application Programming Interface) [32]. Con la implementación de diversos APIs, tenemos una línea de comunicación vertical donde el plano de datos se comunica con cierto protocolo southbound al núcleo del controlador, el núcleo del controlador se las apaña con las interfaces MLAPIs para comunicarse con las aplicaciones internas y finalmente estas expuestas a través de APIs RESTful. Un ejemplo gráfico se encuentra en la Figura 5.16, tomada de [32].

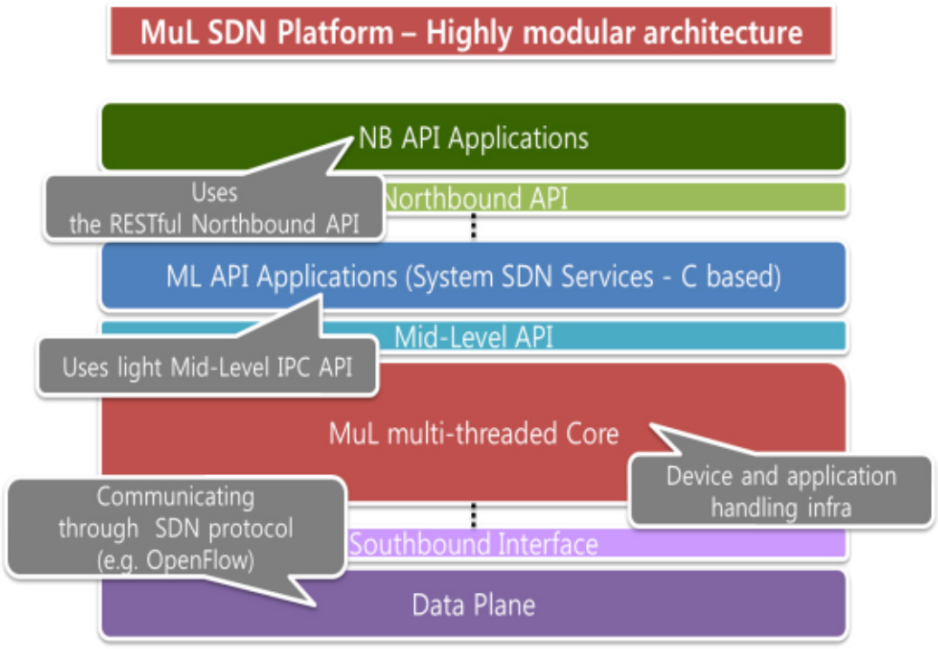

Figura 5.16: Esquema de Comunicación OpenMUL

El núcleo del controlador OpenMUL representa al componente principal. Maneja todas las conexiones de bajo nivel con los switches, se encarga del procesamiento OpenFlow y otros protocolos southbound de acceso a los dispositivos como NETCONF y OSVDB garantizando que los flujos, grupos, medidores, etc. Estén sincronizados entre switches y controlador incluso a prueba de reinicios y fallas.

Los servicios de OpenMUL en cambio, residen separados sobre el núcleo del controlador, pero interconectados a través de manejadores de eventos MLAPIs dispuestos a ser servidos hacia las diversas aplicaciones SDN:

- **Topology Discovery:** Automáticamente detecta los dispositivos de red presentes con el uso de paquetes LLDP para crear una representación de la topología actual, con la capacidad adicional de detectar y prevenir bucles de red en demanda con el uso de STP (Spanning Tree Protocol) [33].
- **Path Finding:** Permite detectar el camino más corto nodo a nodo usando el algoritmo Flloyd-Warshall, con la posibilidad de permitir influenciar la selección de la ruta usando parámetros externos como velocidad del enlace, latencia, etc.
- **Path Connector:** Es un servicio que entrega a disposición una interfaz maleable para la inserción de flujos a través de un camino. Las aplicaciones SDN suelen manejar SDN como una única área, área donde a veces creamos flujos entre par de puntos, flujos que solicitan mantenimiento, cálculo de ruta, etc. Path Connector abstrae al desarrollador para el desarrollo de aplicaciones.
- **L2/L3 Host Tracker:** Permite rastrear hosts usando combinaciones IP, MAC.
- **NAT:** Permite el uso de NAT (Network Address Translation) [34].

En adición a los servicios consumibles para el desarrollo de aplicaciones, OpenMUL ofrece aplicaciones que mejoren la experiencia de usuario a través de ciertas comodidades:

- **L2 Switch:** Permite acoplar aplicaciones que modelen la lógica de aprendizaje de un switch L2 tradicional.
- **NBAPI:** Provee un API RESTful a través de un servidor web escrito en Python.
- **CLI:** Dirigido al usuario final, entrega una consola de línea de comandos para el manejo de todos los componentes MUL.

El conglomerado de beneficios que ofrece OpenMUL puede ser apreciado en la Tabla 5.4, tomada de [32].

Beneficio	Descripción
Southbound Protocols	OpenFlow 1.4, 1.3.x, 1.0, NETCONF, OVSDB
Multi-Version	Soporta diferentes versiones simultáneamente
Flow Download Rate	10.5 Millones/sec
FlexPlug™	Provee la flexibilidad de añadir y remover cualquier aplicación.
FirmFlow™	Coherencia de flujos en el reinicio de aplicaciones, controladores y switches.
SSL	Soporta el protocolo TLS 1.2
Learning Module	Módulo de aprendizaje de switch L2
Multi-Mode OpenFlow	Provee disponibilidad en mezclas heterogéneas de switches OpenFlow
Topology Discovery	Descubrimiento de topología automático.
Path Compute	Cálculo de caminos con el algoritmo Floyd-Warshall
APIs	Set de APIS: RESTful, Python y C
P2P Networks	Soporte multi-inquilino P2P fabric usando el modelo P+V
Statistitcs	Recolección de estadísticas OpenFlow
Hot standby HA	Disponibilidad del controlador para mitigar cortes de enlaces.
Loop detection	Rápida detección de bucles con xSTP.
Address space protection	Las aplicaciones del sistema corren en un espacio de direcciones distinto al del controlador.
CLI	Soporte de usuario a través de CLI

Tabla 5.4: Beneficios de OpenMUL

5.6 Evaluación de Controladores

Como hemos visto en capítulos anteriores, las SDNs se presentan como un nuevo paradigma para cubrir las nuevas necesidades, y promete transformar las arquitecturas y la gestión de las redes que se conocen en la actualidad. Pero ante la existencia de una cantidad tan amplia de controladores SDN que ofrecen diversas funcionalidades, la escogencia de uno de estos para el despliegue de una red, es una tarea ardua, en la cual se debe de evaluar características tales como; las funcionalidades ofrecidas por cada uno de los controladores, las facilidades de desarrollo y de administración del controlador, pero la más importante es el desempeño del controlador en diferentes ambientes. Entre las características que se consideran que se deben tener en cuenta al evaluar un controlador SDN están:

1. **Soporte OpenFlow:** Al elegir un controlador es necesario conocer las características de las versiones de OpenFlow que el controlador soporta, así como las posibilidades que ofrece el proveedor para migrar a las nuevas versiones del protocolo, tales como la versión 1.3, 1.4 y 1.5. Una razón por la que esto es necesario, es que algunas funciones importantes como, por ejemplo, el soporte de IPv6 no es parte de OpenFlow 1.0 pues se incluyen a partir del estándar OpenFlow 1.2.
2. **Virtualización de red:** Debido a los beneficios que ofrece la virtualización de red, un controlador SDN debe soportar esta característica la cual permite la creación dinámica de las redes virtuales, en base a políticas, disociadas de las redes físicas, para satisfacer una amplia gama de requisitos como, por ejemplo; la ampliación horizontal de la capacidad, sin afectar los flujos existentes.
   Otra de las muchas ventajas de la virtualización de red es que permite un completo aislamiento entre cada segmento de red lo que es muy útil por razones de seguridad como, por ejemplo, mantener aislados los datos generados por un grupo de usuarios de otros usuarios y permitir a los desarrolladores de aplicaciones ejecutar las mismas en un entorno de trabajo sin afectar el tráfico. Para cumplir estos requisitos de manera eficiente, en los controladores SDN se deben configurar las redes virtuales de forma centralizada, con total aislamiento unas de otras, y dichas configuraciones deben estar automatizadas.
3. **Funcionalidad de la red:** Para lograr mayor flexibilidad en términos de cómo los flujos son enrutados, es importante que el controlador SDN pueda tomar decisiones de enrutamiento basado en múltiples campos de la cabecera de OpenFlow. También es importante que el controlador pueda definir los parámetros de QoS flujo por flujo.
   Otra importante funcionalidad en un controlador SDN es su capacidad para descubrir múltiples caminos desde el origen del flujo a su destino y para dividir el tráfico de un flujo dado a través de múltiples enlaces. Esta capacidad elimina la necesidad de STP (Spanning Tree Protocol) y aumenta el rendimiento y la escalabilidad de la red permitiendo, también, eliminar la necesidad de añadir a la complejidad de la red nuevos protocolos como TRILL (Transparent Interconnection of Lots of Links) o SPB (Shortest Path Bridging).
4. **Escalabilidad:** Una consideración fundamental con respecto a la escalabilidad de una red SDN es el número de conmutadores o switches que un controlador SDN puede soportar. En la actualidad se debe esperar que los controladores soporten un mínimo de 100 switches, pero en última instancia esto depende de las aplicaciones que soportan. Otro factor que limita la escalabilidad de una red SDN es la proliferación de entradas en la tabla de flujo, ya que, sin algún tipo de optimización, se requiere de una entrada salto por salto para cada flujo.
   Al evaluar los controladores SDN, es necesario asegurarse que el controlador puede disminuir el impacto de sobrecarga de difusión de red, la cual limita la escalabilidad de la arquitectura de red implementada y reducir al mínimo la proliferación de las entradas de la tabla de flujo. Otro aspecto de la

escalabilidad es la capacidad del controlador SDN para crear una SDN que pueda abarcar múltiples dominios. Esta capacidad permite el movimiento de máquinas virtuales y el almacenamiento virtual entre dominios. Para maximizar el beneficio de esta capacidad, el controlador SDN debe permitir que las políticas de red para el enrutamiento y reenvío se apliquen automáticamente para la migración de servidores y/o almacenamiento.

5. **Rendimiento:** Una de las principales funciones de un controlador SDN es establecer flujos. Por ello, dos de los indicadores claves de rendimiento asociados con un controlador SDN son el tiempo de conformación de flujo y el número de flujos por segundo que puede establecer el controlador. Estas métricas de desempeño influyen en gran medida cuando se requiere añadir controladores como, por ejemplo, cuando los switches inician más flujos de los que pueden ser soportados por el controlador o los controladores SDN existentes.
6. **Programación de red:** Una de las características fundamentales de las SDNs es la existencia de interfaces para la programación del controlador, lo que posibilita que este ofrezca varias funcionalidades. Ejemplos de programación que se deben buscar en un controlador SDN son la capacidad de redirigir el tráfico (por razones de seguridad se puede disponer que el tráfico entrante a un servidor pase a través de un firewall, pero para no consumir los recursos del servidor de seguridad con tráfico limpio, se puede decidir que no pase por el firewall el tráfico saliente del mismo servidor) y la posibilidad de aplicar filtros sofisticados a los paquetes (que pueden ser pensados como ACLs dinámicas e inteligentes como combinaciones complejas de múltiples campos de cabecera de paquetes).
   Un controlador SDN también puede soportar la programación, proporcionando plantillas que permitan la creación de secuencias de comandos CLI con las que es posible la programación dinámica de la red.
7. **Confiabilidad:** Una de las técnicas que un controlador SDN puede utilizar para aumentar la fiabilidad de la red, es la capacidad de descubrir múltiples caminos desde el origen hasta el destino lo cual puede realizar si continuamente controla la topología de la red. Si el controlador SDN establece varias rutas entre el origen y el destino, la disponibilidad de la solución no se ve afectada por la interrupción de un sólo enlace. Alternativamente, si el controlador SDN únicamente establece una ruta del origen al destino, cuando ocurra un fallo en un enlace, el controlador debe ser capaz de redirigir el tráfico rápidamente a un enlace activo.
8. **Seguridad de la red:** Con el fin de proporcionar seguridad a la red, un controlador SDN debe ser capaz de soportar la autenticación y autorización. Debido a que un controlador SDN es candidato para un ataque malicioso, necesita poseer la capacidad de limitar las comunicaciones de control, y ser capaz de detectar cuando la red está experimentando los inicios de un ataque.
9. **Monitorización centralizada y visualización:** Un controlador SDN tiene que ser capaz de utilizar los datos ofrecidos por el protocolo OpenFlow para identificar los problemas en la red y, automáticamente, cambiar la ruta que toma un flujo determinado. El controlador también debe poder definir qué tipo

de tráfico controlar. Por ejemplo, se puede optar por no controlar el tráfico de replicación.

Ser capaz de visualizar una red tradicional siempre ha sido a la vez importante y difícil lo que se incrementa en un entorno en el que múltiples redes virtuales se ejecutan en la parte superior de una red física donde es necesario que el controlador SDN sea capaz de descubrir y presentar la visualización de los enlaces físicos de la red y de las múltiples redes virtuales que se ejecutan en la red física.

El controlador también debe permitir ver los flujos, tanto de la perspectiva de la red física como de la virtual, y obtener información detallada sobre los mismos. También, debería ser posible monitorear el controlador SDN utilizando protocolos y técnicas estándares de gestión tales como SNMP. Adicionalmente el controlador SDN debe ofrecer soporte para una amplia gama de MIBs (Management Information Bases) estándares y MIBs privadas para poder controlar los elementos de la red virtual. Lo ideal sería que el controlador SDN ofrezca acceso a la información de red, por ejemplo, a los dispositivos conectados, al estado del puerto, y al estado de enlace a través de una API REST.

10. **Soporte de plataformas:** Los controladores SDN corren sobre sistemas operativos siendo importante que el que se esté evaluando sea multiplataforma pues ello permite mayor flexibilidad e independencia al implantarlo. En muchas entidades es de gran interés que el controlador corra sobre plataformas de código abierto.
11. **Procesamiento:** Al evaluar un controlador se debe conocer si el mismo soporta procesos múltiples o no, pues esto puede repercutir en la escalabilidad de los núcleos del CPU. No tendría sentido que un controlador monoprocesador se ejecute sobre un hardware que posee múltiples CPUs pues no se estaría usando adecuadamente el mismo o que un controlador que soporte procesos múltiples se esté ejecutando sobre un hardware de un solo CPU. Por otra parte, los controladores de procesos múltiples deben ser los utilizados en redes empresariales mientras que los monoprocesadores se pueden emplear en red pequeñas.

Estudiar las características innatas de los diferentes controladores y las funciones que los mismos ofrecen es importante para ponderar el valor que ofrecen dentro del espectro para el cual fueron diseñados.

6. Herramientas de Interés

Junto al paso del tiempo, se ha hecho un escrutinio cada vez más detallado sobre SDN y las potenciales oportunidades para la implementación de este enfoque. Su implementación arroja el desarrollo de nuevas investigaciones, nuevas herramientas, nuevas librerías y nuevos servicios de una creciente flexibilidad sólo con el fin de ofrecer una mejor experiencia en el desarrollo de aplicaciones. Por esta razón el enfoque de este capítulo es describir diversas herramientas junto a sus servicios, módulos y funciones principales útiles bajo el contexto SDN.

6.1 Herramientas de Virtualización

En el presente trabajo se instanciarán switches basados en software con el fin de la realización de pruebas en escenarios lo más cercanos a la realidad posible. En estas pruebas se virtualizará una red SDN compuesta por un número específico de switches y un controlador. Con este fin, se estudian Open vSwitch y Mininet, dos herramientas que tienen la capacidad de ejercer las funciones de capa de enlace de datos y de red de un switch OpenFlow

6.1.1 Open vSwitch

Open vSwitch es un software de código abierto Apache2, cuya finalidad es implementar un switch con calidad de producción que pueda soportar interfaces administrables, abriendo la posibilidad a la programación y control remoto. Está bien adaptado para funcionar como un switch virtual en ambientes implementados con máquinas virtuales. Además de exponer interfaces estándar de control y visibilidad con la capa de red virtual, fue diseñado para soportar una distribución a través de múltiples servidores físicos. La versión actual de Open vSwitch es compatible con las siguientes características:

- Modelo de VLAN estándar 802.1Q con “trunk” y puertos de acceso.
- Unión de NIC con o sin LACP (Link Aggregation Control Protocol) en el switch de subida de enlace.
- NetFlow, sFlow, SPAN (Switched Port Analyzer), RSPAN (Remote Switched Port Analyzer) y ERSPAN (Encapsulated Switched Port Analyzer) para el incremento de visibilidad.
- Configuración de QoS (Calidad de Servicio) y políticas de dicho proceso:
    - Túneles con soporte de CAPWAP (Control and Provisioning of Wireless Access Points), GRE (Generic Routing Encapsulation) y GRE sobre IPsec.
    - Conectividad de gestión de fallos bajo el estándar 802.1ag
    - Soporte de OpenFlow y otras extensiones.
    - Configuración de bases de datos de forma transaccional para enlaces con C y Python.
    - Capa de compatibilidad para realizar puentes con Linux
    - Transmisión de alto rendimiento utilizando el módulo kernel de Linux.

El módulo de kernel Linux incluido en Open vSwitch soporta Linux 2.6.18 y versiones superiores, también tiene soporte para Citrix XenServer y hosts de la versión de Linux Red Hat Enterprise. Open vSwitch puede operar enteramente dentro de espacio de usuario sin ayuda de un módulo del kernel, pero esto influye en el rendimiento del switch virtual como ya se ha hablado anteriormente. En la Figura 6.1, tomada de [35] se muestra la arquitectura de Open vSwitch y a continuación una breve descripción de sus componentes principales.

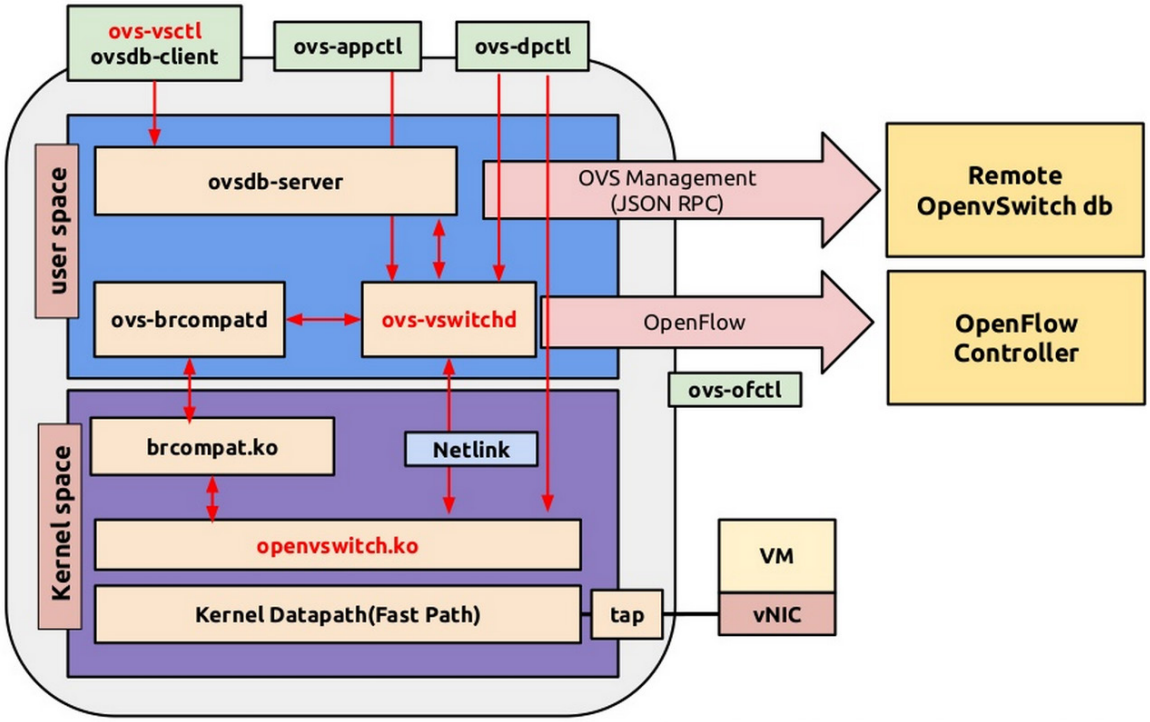

Figura 6.1: Arquitectura de OpenvSwitch

- **ovs-vswitchd:** Es un demonio implementado por el switch, junto a un módulo kernel de Linux para el reenvío basado en flujo.
- **ovsdb-server:** Servidor de base de datos ligera que realiza consultas al ovs-vswitchd para obtener información de su configuración.
- **ovs-brcompatd:** Es un demonio que permite al ovs-switch actuar como reemplazo de un bridge Linux en distintos ambientes, junto a un módulo de kernel Linux para interceptar las llamadas ioctls del bridge.
- **ovs-dpctl:** Es una herramienta para configurar el módulo kernel del switch.
- **Scripts y especificaciones:** Para construir RMPs (Red Hat Package Managers) para servidores Citrix XenServer y Linux Red Hat Enterprise. El RPMs de XenServer le permite a Open vSwitch ser instalado en un servidor Citrix Xen como reemplazo de switch, con funcionalidades adicionales.

- **ovs-vsctl:** Es una utilidad para consultas y actualizaciones de la configuración del ovs-vswitchd.
- **ovs-appctl:** Es una utilidad que envía comandos para los daemons (“demonios”) del Open vSwitch.
- **ovsdbmonitor:** Herramienta GUI para supervisión remota de bases de datos OVS y tablas de flujos OpenFlow.
- **ovs-controller:** Controlador de OpenFlow con funcionalidades limitadas.
- **ovs-ofctl:** Es una utilidad para consultas y control de switches y controladores OpenFlow.
- **ovs-pki:** Es una utilidad para crear y administrar la infraestructura de llave pública para switches OpenFlow.
- **TCPDump:** Le permite a Open vSwitch el análisis de mensajes OpenFlow.

6.1.2 Mininet

Es un simulador de red que permite la creación de escenarios virtuales completos, que incluye controlador, switches, hosts y enlaces. Su principal característica es que los switches que virtualiza tienen soporte del protocolo OpenFlow, permitiendo así la creación de escenarios experimentales de SDN.

Los hosts virtualizados son accesibles individualmente, mediante una conexión ssh, a su vez cada host puede ejecutar aplicaciones incluidas en el sistema Linux. Se puede realizar envío de paquetes entre los hosts virtualizados, los enlaces que comunican los hosts simulan el protocolo Ethernet y se puede especificar valores para el enlace, tales como la latencia. Por lo tanto, Mininet cumple con las siguientes características [36]:

- **Flexible:** Permite interpretación de nuevas topologías y funcionalidades recordando que la principal utilidad se da con redes definidas en software, mediante el uso del lenguaje de programación Python en plataformas como Linux y Windows.
- **Desplegable**: La posible implementación de un prototipo funcional no debería exigir cambios en el código o en la configuración en un dispositivo final.
- **Interactivo:** La administración y operación de una red emulada se realiza en tiempo real, como si se estuviese trabajando sobre una red física.
- **Escalable**: En una máquina física se permite la escalabilidad con cientos y miles de switches.
- **Realista:** El comportamiento de un prototipo desarrollado en Mininet es el mismo que con un dispositivo real.
- **Compartible:** Experimentación sencilla de prototipos independientes, poder realizar pruebas y compartir estas ideas desarrolladas de una manera fácil.
- **Código abierto:** Mininet es un proyecto de código abierto, por lo que se les anima a los usuarios a examinar su código fuente, modificarlo, corregir errores, y enviar solicitudes de parches.

El código de Mininet en su mayoría está escrito en Python, con una pequeña parte de esta en C. Posee una interfaz de línea de comandos para crear, gestionar y compartir los escenarios creados. Las principales características que destacan de Mininet son las siguientes [36]:

- Permite probar una topología de red compleja sin la necesidad de utilizar una red física.
- Ofrece la facilidad de realizar pruebas de aplicaciones OpenFlow, simplificando así el desarrollo de estas.
- Permite utilizar topologías predeterminadas o cargarlas en un archivo.
- Proporciona una API de Python para facilitar la creación de redes y su posterior experimentación.
- Su interfaz de línea de comandos proporciona comandos de diagnóstico de la red virtualizada.
- Incluye una interfaz de línea de comandos, que proporciona comandos de diagnóstico, así como la capacidad para enviar una orden a un nodo de la red.
- Compatibilidad con topologías personalizadas.
- Una de las características especiales de Mininet es la posibilidad de ejecutar topologías con un número determinado de nodos o dispositivos.

Como se mencionó, Mininet permite la carga de una topología desde un archivo. Este archivo puede ser generado por una herramienta para la creación de estas topologías SDN llamada VND (Virtual Network Description) con la que se puede diseñar todo tipo de topologías, personalizando cada uno de los elementos de la red. Además, permite acoplar controladores con soporte OpenFlow.

Mininet proporciona un procedimiento para la interacción con una red SDN, dicho procedimiento es para crear, personalizar y compartir una SDN, el proceso es el siguiente [36]:

1. **Creación:** Mininet emula los diferentes elementos, links, hosts, switches y controladores, para crear una red SDN, este proceso se hace a través de la virtualización del SO Linux.
    - **Link**: Es un enlace de Ethernet virtual con la función de actuar como un cable de red conectando dos interfaces virtuales, pudiendo enviar paquetes desde una interfaz con destino de la otra, el comportamiento de este link es como el de un puerto de Ethernet real.
    - **Host:** En Mininet un host es un proceso simple de Shell situado dentro de su propio espacio de nombres de red, cada host posee su propio enlace virtual.
    - **Switch:** Es un dispositivo de red capaz de realizar retransmisión de paquetes, con software OpenFlow.
    - **Controlador**: Un controlador puede estar en algún lugar de la red, sólo que este siempre tenga conectividad a nivel IP con otros dispositivos (switch) cuando estos estén en ejecución.

2. **Iteración**: La iteración en Mininet se realiza cuando se empieza la red, se pueden ejecutar diferentes comandos para los distintos elementos que la conforman, para esto Mininet provee un interfaz CLI, permitiendo a los usuarios poder controlar y gestionar la red.

3. **Personalización**: La posibilidad de exportar un archivo ".py", en donde el contenido de este sea una serie de líneas de comandos escritas en lenguaje Python, pocas líneas de código permiten la posibilidad de crear una topología y la personalización de los nodos.

4. **Ejecución en hardware**: En Mininet cada elemento se comporta de la misma forma que lo hace uno físico, esto para realizar la adecuación al hardware, componentes como la topología debe ser la misma de forma virtual a la física, los elementos emulados deben ser reemplazados a su equivalente físico, cada switch con OpenFlow debe ser reemplazado por uno real, ya configurado para conectarse simplemente al controlador.

6.2 Herramientas de Desarrollo

En esta sección se describirán algunos frameworks y herramientas utilizadas en programas como Cbench, que fueron diseñados para optimizar el desarrollo de aplicaciones OpenFlow desde distintas perspectivas.

6.2.1 Libpcap

Es una librería de código abierto que aporta principalmente una interfaz de alto nivel para la captura de paquetes. Fue desarrollado por investigadores de Lawrence Berkeley National Laboratory en 1994 como parte de un proyecto de investigación para analizar y mejorar TCP y el rendimiento de las puertas de salida en Internet [37]. El principal objetivo de los autores (McCanne, Leres y Jacobson) fue crear una plataforma independiente en forma de API para eliminar la necesidad de sistemas dependiente de módulos desarrollados para la captura de paquetes, en cada aplicación [37].

El API de Libpcap fue diseñado para ser usado en C y C++ por procesos que realizan llamadas a funciones en nivel de usuario. Sin embargo, existe una variedad de contenedores que permiten utilizarlo desde lenguajes de programación como Perl, Python, Java, C# o Ruby. Libpcap corre en la mayoría de SO parecidos a Unix (Linux, Solaris, BSD, HP-UX, etc). También existe una versión para Windows llamada Winpcap.

En el caso de la captura de paquetes, no importa lo complicado que se torne un programa escrito con Libpcap, todos tienen un esquema básico ilustrado en la Figura 6.2, posteriormente encontraremos funciones importantes en las fases S0, S1, S2, S3 y S4 mostradas en la Figura 6.2.

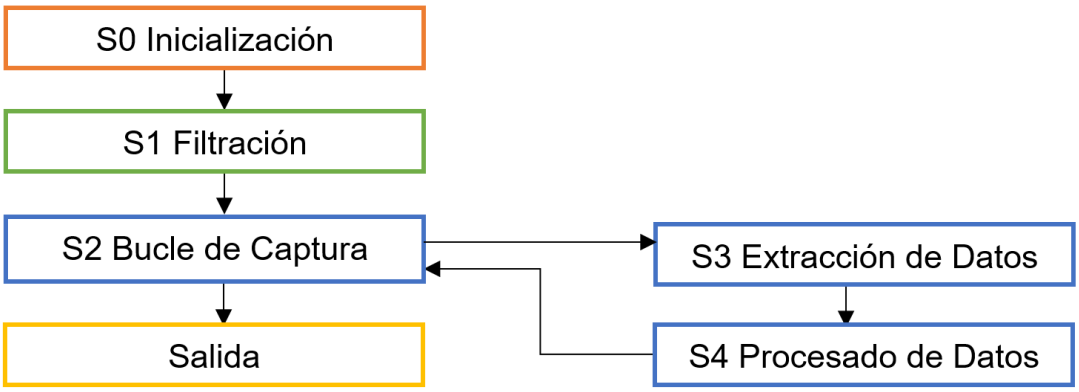

Figura 6.2: Arquitectura de Programas con Libpcap

0. **Inicialización:** Esta fase engloba funciones capaces de obtener información del sistema: Interfaces de red instaladas, configuración de estas interfaces (dirección, máscara), etc.

1. **Filtración:** Aun cuando los procesos que ejecutan las funciones de Libpcap se encuentran a nivel de usuario, la captura real de paquetes se realiza en el kernel. Por esta razón se diseñó un mecanismo para intercambiar de espacio de manera eficiente y segura [38].

   Para evitar que todos los paquetes, incluyendo aquellos no deseados generen un evento de traspaso entre los espacios de usuario y kernel, se crearon filtros. Estos filtros varían en implementaciones y son prácticamente dependientes del SO [38].

   Básicamente los filtros intervienen en el paquete después de que el driver de red reciba el paquete, y antes de enviarse a la pila de protocolos. Donde se compara el paquete con los distintos filtros. Si el paquete coincide con algún filtro, se enviará a la pila de protocolos, resultando finalmente en cambio de espacios necesario.

   Los filtros se configuran con un lenguaje de bajo nivel, sin embargo, Libpcap ofrece un lenguaje de alto nivel para programar filtros. Actualmente convertido en estándar para definir filtros de captura. Los filtros evalúan direcciones, protocolos (tcp, udp, ip, ether, arp, rarp, fddi) y tipo de dirección (host, red o puerto) [38]. De esta manera se pueden crear múltiples filtros, incluyendo filtros personalizados que evalúen partes de los campos, con el uso de expresiones regulares.

2. **Captura de Paquetes:** Una vez se tienen las interfaces listas y preparadas para la captura de paquetes comienza otro grupo de funciones. Diferentes entre ellas por la cantidad de paquetes, modo de captura y manera en la que se han definido.

3. **Extracción de Datos:** Al recibir un paquete el programa recibe el contenido bruto del paquete, por lo que debe realizar el proceso de extracción que la pila de protocolos hubiera hecho. Esto implica remover las cabeceras e interpretar los campos de interés. En el caso de un paquete TCP los pasos serían [38]:
   - Extraer cabecera Ethernet
   - Consultar el campo ether_type, para saber si es un paquete IP el que viene a continuación.
   - En caso de ser IP, consultar el campo protocol.
   - Revisar en /usr/includes/netinet y consultar cómo es la cabecera del protocolo.
   - Consultar el tamaño la última cabecera y el payload si lo lleva.

4. **Procesado de Datos:** Libpcap no sólo nos permite interpretar paquetes a través de sus funciones, permite guardar los datos en un archivo para su posterior uso.

6.2.2 NetSNMP

SNMP son las siglas de Simple Network Management Protocol [39] y es un protocolo muy extendido que trabaja en el nivel de aplicación. Mediante SNMP se puede realizar tareas de monitorización de red, configuración de dispositivos, detección de problemas y otras muchas cosas más.

Para gestionar un dispositivo mediante SNMP, éste debe tener un agente que gestione la información contenida en la MIB-II (Management Information Base). El protocolo funciona mediante un mecanismo de petición-respuesta, es decir, se realiza el envío de una petición u orden al agente que gestiona un dispositivo concreto y éste nos responde con la información requerida o el resultado de la orden, o con un mensaje de error si la petición fue incorrecta. Actualmente existen tres versiones de este protocolo: SNMPv1, SNMPv2 y SNMPv3; cada una con sus propias características y limitaciones.

NET-SNMP es un conjunto de aplicaciones usado para implementar el protocolo SNMP usando IPv4 e IPv6. Incluye:

- Aplicaciones de línea de comandos para:
  - tomar información de dispositivos capaces de manejar el protocolo SNMP, ya sea usando peticiones simples (snmpget, snmpgetnext) o múltiples (snmpwalk, snmptable, snmpdelta).
  - manipular información sobre la configuración de dispositivos capaces de manejar SNMP (snmpset).
  - conseguir un conjunto de informaciones de un dispositivo con SNMP (snmpdf, snmpnetstat, snmpstatus).
  - traducir entre OIDs numéricos y textuales de los objetos de la MIB, y mostrar el contenido y estructura de la MIB (snmptranslate).
- Un navegador gráfico de la MIB (tkmib), usando Tk/Perl.

- Un demonio para recibir notificaciones SNMP (snmptrapd). Las notificaciones seleccionadas pueden guardarse en un log (como syslog o un archivo de texto plano), ser reenviadas a otro sistema de gestión de SNMP, o ser pasadas a una aplicación externa.
- Un agente configurable para responder a peticiones SNMP para información de gestión (snmpd). Incluye soporte para un amplio rango de módulos de información de la MIB, y puede ser extendido usando módulos cargados dinámicamente, scripts externos y comandos.
- Una biblioteca para el desarrollo de nuevas aplicaciones SNMP, con APIs para C y Perl.

6.2.3 Sockets

Los sockets no son más que puntos o mecanismos de comunicación entre procesos, que permiten que un proceso envíe o reciba información con otro proceso, incluso estando estos procesos en distintas máquinas. Esta característica de ínter conectividad entre máquinas hace que el concepto de socket sea de gran utilidad.

La forma de referenciar un socket es mediante un descriptor, del mismo tipo que el utilizado para referenciar archivos. La comunicación entre procesos, a través de sockets, se basa en la filosofía cliente servidor. Un proceso en esta comunicación actuará de proceso servidor, creando un socket, cuyo nombre conocerá el proceso cliente, el cual podrá establecer comunicación con el proceso servidor, a través de la conexión con dicho socket nombrado. El proceso cliente crea un socket sin nombre cuyo valor de vuelta es un descriptor sobre el que se leerá o escribirá, permitiéndose una comunicación bidireccional, característica propia de los sockets y que los diferencia de los pipes, o canales de comunicación unidireccional entre procesos de una misma máquina. El mecanismo de comunicación vía sockets tiene los siguientes pasos:

1. El proceso servidor crea un socket con nombre y espera la conexión.
2. El proceso cliente crea un socket sin nombre.
3. El proceso cliente realiza una petición de conexión al socket servidor.
4. El cliente realiza la conexión a través de su socket mientras el proceso servidor mantiene el socket servidor original con nombre.

Es muy común en este tipo de comunicación lanzar un proceso hijo, una vez realizada la conexión, que se ocupe del intercambio de información con el proceso cliente, mientras el proceso padre servidor sigue aceptando conexiones. Para eliminar esta característica se cerrará el descriptor del socket servidor con nombre en cuanto realice una conexión con un proceso socket cliente.

6.2.4 Gnuplot_i

Gnuplot [40] es un programa de visualización gráfica de datos científicos. Permite realizar gráficos 2D y 3D de curvas, líneas de nivel y superficies, tanto a partir de funciones como de datos discretos. Es software libre disponible para casi todas las plataformas incluyendo Linux, Irix, Solaris, Mac OS X, Windows y DOS. Gnuplot

requiere las mínimas capacidades gráficas y puede usarse aún en una terminal de tipo vt100. Tiene una amplia variedad de opciones de salidas para que el usuario pueda usar las gráficas resultantes como lo desee, ya sea para ser visualizadas o para incluirlas en sus propios documentos.

La librería Gnuplot_i proporciona una interfaz con el programa Gnuplot. Es un módulo que permite el envío de peticiones de despliegue de gráficos a través de una sesión Gnuplot en el lenguaje C.

7. Trabajos Relacionados

A lo largo de este capítulo se encontrará descrito un grupo de trabajos relacionados al análisis de OpenFlow. Estos estudian casos particulares desde distintos enfoques, motivaciones, justificaciones y herramientas que entre sí generan un espacio compartido en la evaluación de controladores OpenFlow.

7.1 Performance Comparison of the State of the Art OpenFlow Controllers

Las redes en centros de datos han demostrado una integración exitosa con el protocolo OpenFlow, logrando redes más consistentes con la rápida expansión de máquinas virtuales. Pero con el creciente tráfico de los centros de datos, la necesidad por controladores con altos desempeños incrementa [41]. Por esa razón los autores de [41] pusieron a prueba con el programa Cbench, seis controladores: NOX, Beacon, Floodlight, Maestro, OpenMul y OpenIRIS (algunos de ellos mencionados previamente en este documento).

7.1.1 Metodología

Las pruebas buscaron medir el rendimiento y la latencia con distintas cantidades de switches conectados al controlador, cantidades que iban desde 1 switch hasta 128 switches. En las pruebas de rendimiento se usaba el envío de mensajes Packet-In para medir la capacidad de estrés soportado por controlador. En las pruebas de latencia se envió un solo paquete al controlador, se esperaba por una respuesta y reiniciaba el ciclo de nuevo. Con las pruebas de rendimiento se trata de obtener la mayor cantidad de mensajes manejables por el controlador.

Se consideró un grupo de parámetros específicos para las evaluaciones:

- Número de switches OpenFlow.
- Número de hosts (flujos).
- Versión OpenFlow.
- Repeticiones de prueba.
- Duración de prueba.

7.1.2 Conclusiones

Tras una serie de pruebas, emergieron dos switches, OpenIRIS y OpenMul, cada uno sobresaliente en distintas áreas. OpenIRIS es el controlador con la menor latencia en manejo de flujo con 5µs por flujo, en cambio OpenMul es el controlador con mayor rendimiento, alcanzando manejar casi 2.7 millones de flujo por segundo. Aunque todos los controladores puestos a prueba tienen soporte multi-hilo, Hassan y Ahmed explican [41] que estos difieren en sus comportamientos al escalar, “La diferencia en rendimiento puede ser causada por dos factores: el primero son los algoritmos que los controladores usan para distribuir los mensajes entrantes entre los hilos, y el segundo son los mecanismos y librerías usadas para la interacción entre la red y el controlador”. factores importantes a considerar en la implementación de la investigación que referencia este trabajo.

7.2 OpenFlow Switching Performance

OpenFlow intenta solucionar los problemas de escalabilidad y flexibilidad entregándole al usuario de manera simple una parte de la red sin estorbar el tráfico de existente, aun así, OpenFlow puede percibirse de otras maneras, una de ellas hace referencia a la funcionalidad del switch, donde OpenFlow sólo sigue reglas predefinidas y reenvía el paquete al destino especificado, similar al filtro de un firewall o NAT. La segunda forma se da durante la virtualización, donde OpenFlow puede aislar flujos con configuraciones predefinidas, similares a VPNs (Virtual Private Network) [42] en donde podemos encontrar VLANs Ethernet que trabajan en L2, tecnologías MPLS que trabajan en L2/L3 o tecnologías obsoletas de circuitos pseudocableados como FR (Frame Relay) o ATM (Asynchronous Transfer Mode). El problema es que son tecnologías con sistemas de manejo complicados e inflexibles en un sentido multi-protocolo [23]. La cuestión reside ¿OpenFlow puede suplantar los sistemas de enrutamiento tradicionales? Afortunadamente Palacin [23] realiza experimentos para comparar el rendimiento y latencia de un sistema OpenFlow con switches tradicionales y sus tecnologías de reenvío L2/L3 de manera tal se pueda obtener información relevante para responder esta pregunta.

7.2.1 Metodología

Los experimentos fueron diseñados tomando en cuenta el uso de un PC Ubuntu para simular distintas tecnologías (OF, switch L2, switch L3), una laptop con S.O. Ubuntu y un generador de paquetes sintético llamado Agilent N2X usado para enviar paquetes con distintos patrones. En el caso de switches L2 se configura la PC para proveer el sistema de reenvío en el kernel y se toman dos puertos para formar un puente de comunicación entre ambos. En el caso de enrutamiento L3 se habilita la funcionalidad ip_forwarding del kernel y se configuran dos puertos para la retransmisión en forma de enrutamiento. Finalmente, para el caso de OpenFlow se debe habilitar el módulo OpenFlow (OpenFlow versión v0.8.9r2) y configurar dos puertos que permitan actuar como switch OpenFlow usando la herramienta dpctl (DataPathControl).

Se realizaron seis experimentos:

- **1er Experimento:** Una prueba de carga en la cual las tres tecnologías se someten a comparación con diferentes tamaños de paquetes y cargas.
- **2do Experimento:** Una prueba de carga en la cual las tres tecnologías se someten una vez más a comparación, esta vez con diferentes tamaños de tablas de reenvío.
- **3er Experimento:** Una prueba para analizar el comportamiento de OpenFlow con la combinación de distintas tablas de configuraciones.
- **4to Experimento:** Una prueba para observar las diferencias al balancear el tráfico con las tablas de reenvío de OpenFlow.
- **5to Experimento:** Una prueba para evaluar la imparcialidad de las tres tecnologías en un contexto donde hay dos flujos de entrada y solo uno de salida.

- **6to Experimento:** Una prueba enfocada en comparar la latencia de paquetes acorde al tamaño de la tabla, tipo y longitud del paquete, bajo la utilización de OpenFlow.

Para la realización de estas pruebas se usaron distintos valores de parámetros acorde al experimento dentro del generador de paquetes:

- Tamaño de paquete.
- Tamaño de tablas de reenvío.
- Porcentaje de carga.
- Balanceo de carga.
- Velocidad del enlace.
- Tiempo de prueba.

7.2.2 Conclusiones

Palacin [23] resume sus experimentos en una frase "…podemos concluir que la tecnología de reenvío OpenFlow es una seria alternativa al software de reenvío Ethernet o enrutamiento IP porque realiza las mismas funciones L2 y L3 con mejor rendimiento y escalabilidad. OpenFlow no sólo realiza reenvío L2 y L3, sino que también puede hacer reenvío por puerto, así que podemos considerarlo más flexible y configurable".

Los resultados muestran en [23] que OpenFlow tiene un desempeño similar al enrutamiento IP en términos de rendimiento y procesamiento con grandes tablas de reenvío. Aunado a esto la tecnología L2 de reenvío Ethernet no mostró buenos resultados con paquetes pequeños y grandes tablas de reenvío comparado con las otras dos tecnologías. Ethernet muestra un mal desempeño cuando la red esta sobrecargada con flujos de altas tasas de tráfico. Palacin [23] afirma "el código OpenFlow ha sido bien implementado y optimizado para tareas de reenvío".

Los experimentos en [23] fueron implementados sin el uso de controladores e incluso fueron influenciados por el caching de OpenFlow. Puntos que no interesan en este trabajo de investigación, sin embargo, se considera de interés la capacidad demostrada que tiene OpenFlow de ser instalado en una PC o router comercial como una funcionalidad adicional en TCAM.

7.3 A Flexible OpenFlow-Controller Benchmark

El uso de SDN promete redes más flexibles capaces de adaptarse a los cambios. Esto implica mecanismos de administración más sencillos. En OpenFlow, estos mecanismos son representados a través de un controlador OpenFlow. Al ser el componente principal, el comportamiento y desempeño de este afecta directamente el desempeño de la red. Por tal razón es necesario entender los factores que influyen en él.

Debido a que el estándar de OpenFlow no indica detalles de la implementación de un controlador más allá de garantizar un canal seguro, se han desarrollado una variedad de controles con distintas características y comportamientos singulares.

Jarschel junto a los coautores de [1] logran recopilar a través de su investigación en otros trabajos relacionados condiciones relevantes que afectan directamente el desempeño de los controladores, estas son [1]:

- Redes OpenFlow basadas en switches tradicionales como unidades de reenvío recaen en cuellos de botella ya que no están diseñados para funcionar como switches de flujos.
- Las implementaciones de switches OpenFlow tienen problemas de desempeño debido a las cargas de CPU que conllevan.
- Configuraciones en tablas de flujos para mantener un comportamiento reactivo degrada terriblemente el desempeño de la red en centros de datos cuando sólo ocho switches son manejados por un único controlador.

Jarschel junto a sus coautores [1] se propusieron de desarrollar una herramienta de medición flexible que pueda evaluar los controladores un poco más de lo que se acostumbra con las herramientas de esa naturaleza usando la versión 1.0 de OpenFlow, en la cual el foco es el rendimiento y latencia, aportan la capacidad de emular escenarios y topologías que puedan evaluar el desempeño del controlador para obtener tal vez una noción de los mecanismos de control y asignación en un controlador. Esta herramienta crea un grupo de switches virtuales generadores de mensajes, y es capaz de emular distintos escenarios y capturar estadísticas para su posterior análisis.

7.3.1 Metodología

La herramienta desarrollada posee un diseño en arquitectura e implementación basada en la garantía de los siguientes objetivos:

- Escalabilidad: El software debe permitir múltiples instancias ejecutadas de manera coordinada en distintos núcleos del CPU, CPUs y hosts. Así ampliar el límite de las cargas generadas al probar controladores [1].
- Habilidad para proveer estadísticas detalladas de desempeño: El software debe proveer métricas de desempeño como RTT (Round Trip Time), número de paquetes recibidos/enviados por segundo o número de paquetes no respondidos, en diversas series de tiempo y en términos de switches individuales. Esto permitirá identificar si un controlador OpenFlow trata a los switches de manera diferente o cambia su comportamiento con el tiempo [1].
- Modularidad: El desarrollo de controladores se ha desenvuelto rápidamente, por lo tanto, la herramienta debe ser adaptable a nuevos escenarios. Esto es más alcanzable con un software modularizado. Permite acoplar y desacoplar fácilmente nuevos controles y métricas [1].

La arquitectura de la herramienta consiste en tres componentes primarios, el control central, el cliente ejecutado en los hosts y el switch virtual. Bajo un acercamiento distribuido, la herramienta de medición puede trabajar de manera coordinada en múltiples hosts.

El cliente siendo parte de la herramienta en cuestión, es un sistema de medición completo en sí mismo. Este ejecuta las pruebas de rendimiento usando objetos de

switches virtuales configurados. Sin embargo, el principal componente de la herramienta es el switch virtual, tiene una tabla de flujos simplificada capaz de responder ante el controlador, almacena las estadísticas y las actualiza. Más allá de sus funcionalidades, el switch virtual tiene dos sockets y tres hilos para establecer la comunicación y el procesamiento (generación de paquetes, comunicación con el controlador, comunicación con el control central). Conexión dirigida al control central y el controlador OpenFlow. Toda esta arquitectura del switch virtual permite tratarlo como una verdadera entidad individual [1].

El cliente y el switch virtual fueron desarrollados en C++ usando la librería Boost para el manejo de hilos. Los experimentos son configurados en el control central vía canales de comunicación o archivos de configuración. Entre las opciones de configuración se encuentra:

- Número de switches.
- Tiempo de transmisión inter-paquetes por switch.
- Tamaño de los paquetes.
- Archivos pcap que describan mensajes OpenFlow para producir.

El cliente permite la creación de topologías de switches, especificadas en un archivo de configuración aparte, permitiendo que los switches virtuales reenvíe los paquetes LLDP o OFLDP del controlador, retornando al controlador como un mensaje Packet-In dependiendo de la topología [1].

El proceso de medición sigue un flujo sencillo, el hilo generador de paquetes, crea y envía un mensaje Packet-In al controlador. El tiempo entre ambos paquetes depende de la configuración. Los mensajes contienen la cabecera del paquete del primer paquete en cada flujo IP que el controlador no ha recibido todavía. Cada mensaje es identificable con su *buffer-id*. El controlador luego responde a esos paquetes con mensajes Packet-Out y/o FlowMod usando el mismo *buffer-id.* Al recibir la respuesta, el hilo de comunicación con el controlador parsea el mensaje, extrayendo el ID, calculando el RTT y actualizando las estadísticas. Ya que ambos hilos están usando sockets se introdujo un semáforo para manejar la concurrencia y evitar corrupción en los datos.

El control central es una interfaz de usuario gráfica escrita en Delphi para que los experimentadores puedan ver la configuración actual y modificar las pruebas acordes a los requerimientos.

Para evaluar los resultados de la herramienta de medición, los autores hicieron pruebas comparativas (cada una repetida 5 veces) con Cbench con sistemas compuestos del mismo hardware y software. Sistemas donde un PC ejecuta el controlador y otro ejecuta las herramientas de benchmarking [1]. Estas pruebas consistieron en:

- **Experimento 1:** Una prueba de observación en la que se captura el RTT en milisegundos entre el envío de un mensaje Packet-In y la llegada de un mensaje Packet-Out o FlowMod por cada switch y por serie de tiempo.

- **Experimento 2:** Una prueba de carga en la que se mide la cantidad de paquetes aceptados. La discrepancia entre paquete enviados y los recibidos se llama “paquetes no respondidos” en inglés “outstanding packets”.

7.3.2 Conclusiones

Dada la importancia del controlador OpenFlow en Redes Definidas por Software, es clave entender el funcionamiento y desempeño del componente en un diverso espectro de contextos. Los autores de [1] introducen una herramienta de medición de controladores granular y flexible con resultados comparables con la actual herramienta referente en el área (Cbench). Los resultados, especialmente aquellos en términos de paquetes no respondidos, resaltan la importancia de cuellos de botella y problemas similares que no pueden ser percibidos desde una perspectiva global, problemas resueltos por algunos controladores por el uso de *buffers* internos y preferencias en el manejo de switches.

Al realizar experimentos que capturen métricas superficiales se pierde la capacidad de detectar una gama más amplia de características en controladores. Aun cuando los resultados obtenidos no fueron del todo precisos, el hecho de entregar resultados repetibles y diferentes entre controladores durante las mismas pruebas indica que hay diferentes comportamientos entre controladores. Por lo tanto, sus circunstancias deberían ser analizadas, con el uso de herramientas que permitan análisis más amplios.

7.4 Cbench

El desarrollo de herramientas para la medición de desempeño de controladores OpenFlow carece de fuerza, sin embargo, una de las herramientas pioneras es Cbench (Controller Benchmarker). Diseñada para monitorear controladores OpenFlow a través de la generación de eventos y el uso de Libpcap junto a sockets crudos para la construcción de paquetes. Fue desarrollada por Robert Sherwood como parte de un proyecto más grande llamado Oflops y se ha vuelto una herramienta estándar en la medición de desempeño en controladores OpenFlow. Emula un número configurable de switches OpenFlow los cuales se comunican con un controlador OpenFlow para medir diferentes aspectos de rendimiento y latencia. Su funcionamiento básico consiste en que cada switch emulado envía un número configurable de mensajes de nuevos flujos (mensajes Packet-In) al controlador OpenFlow, espera por las respuestas apropiadas de configuración de flujos (mensajes Packet-Out o mensajes de modificación de flujos FlowMod) y registra estadísticas de la diferencia de tiempo entre las solicitudes y las respuestas, así como otras métricas de desempeño [43]. La recopilación de estas estadísticas es grupal por lo que no se puede analizar en un si los controladores poseen preferencias por ciertos switches a la hora de procesar paquetes.

7.5 Hcprobe

Hcprobe, desarrollado por el instituto ARCCN (Applied Research Center for Computer Networks) en Moscú bajo el lenguaje de programación Haskell, permite

crear con facilidad escenarios para pruebas de control SDN. Es capaz de simular un gran número de switches y hosts conectados a un controlador. Empleando Hcprobe se pueden analizar varios índices de operación del controlador de forma flexible. Con esta herramienta se pueden especificar patrones para generar mensajes OpenFlow (incluidos los malformados) y establecer perfiles de tráfico, entre otros. Sus características principales son:

- Generación de paquetes OpenFlow y de tablas de flujos.
- Implementación en Haskell, un lenguaje de alto nivel, lo que hace que sea más fácil de extender.
- Existencia de una API para el diseño de pruebas personalizadas en temas de desempeño, funcionalidad y seguridad.
- Un lenguaje específico de dominio embebido (EDSL) para la creación de pruebas.

Hcprobe proporciona un framework para la creación de varios casos de uso para estudiar el comportamiento de los controladores OpenFlow a través del procesamiento de diferentes tipos de mensajes. Con esta herramienta se puede generar todo tipo de mensajes OpenFlow switch-controlador y reproducir diferentes escenarios de comunicación entre ellos.

8. Marco Metodológico

Para el desarrollo de la aplicación, se implementó la metodología Ad Hoc incremental [44], la cual combina elementos del modelo secuencial con la filosofía iterativa de construcción de prototipos. Cada iteración se divide en las etapas de análisis, diseño, codificación y pruebas, cuyo modelo se muestra en la Figura 8.1. Por cada requerimiento obtenido del análisis general de la solución se definieron módulos, que fueron desarrollados mediante iteraciones hasta lograr el objetivo.

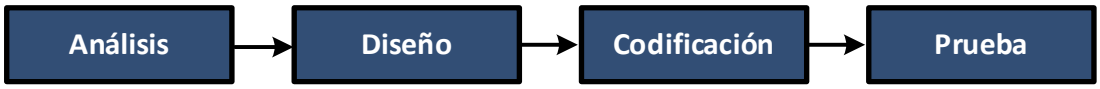

Figura 8.1: Modelo Lineal Secuencial

8.1 Adaptación de la Metodología de Desarrollo

Para satisfacer los objetivos planteados en el Capítulo 2 se dividieron los requerimientos de la aplicación en módulos principales y se determinaron puntos a modificar de la herramienta base, Cbench. Estos módulos y modificaciones son:

- Actualización de soporte OpenFlow de Cbench a la versión 1.3.
- Módulo de consulta SNMP.
- Módulo de reportes.
- Módulo de representación gráfica de resultados.
- Módulo distribuido (esclavo - maestro).
    - Módulo de pase de mensajes de sincronización.
    - Módulo de pase de reportes finales de los clientes.

En cada uno de los puntos anteriores se aplicó la metodología explicada, pasando por las etapas de análisis, diseño, codificación y pruebas, de forma iterativa hasta conseguir satisfacer los requerimientos. A continuación, se explicará con más detalle cada una de las fases pertenecientes a una iteración.

8.1.1 Análisis

Durante esta fase, se definen los requerimientos necesarios para el desarrollo del módulo específico durante esa iteración. De esta forma se genera una visión más granular de alcance de un módulo.

8.1.2 Diseño

En esta fase, se determina la estructura lógica del módulo a desarrollar, para esto se determinan los elementos que conforman el módulo, en conjunto con las funcionalidades pertinentes y estructuras de datos necesarias. Como resultado, se obtiene una descripción del módulo a desarrollar y la interacción entre sus componentes.

8.1.3 Codificación

Luego del análisis necesario, sobre los requerimientos y el diseño de la solución para la satisfacción de los objetivos, se procede a la codificación de lo propuesto.

Las estructuras y funciones planteadas se implementarán y se desarrollarán junto a la resolución de los retos programáticos que surjan.

Esta fase da como resultado un segmento de código desarrollado y almacenado en una rama del repositorio, dedicada a la funcionalidad.

8.1.4 Pruebas

La etapa final de la iteración consiste en la verificación de la solución desarrollada, donde se busca verificar el cumplimiento de los requerimientos planteados al inicio, y la determinación de nuevos requerimientos si es el caso. Con este fin se validan los datos arrojados por la funcionalidad desarrollada, se verifican que sean correctos y por consecuente, que el módulo se ejecute de la manera esperada.

De obtenerse resultados positivos en esta prueba, se procede a la inclusión del módulo en la aplicación mezclando la rama de la funcionalidad perteneciente al repositorio con la rama general del proyecto. Una vez el segmento de código se ha incluido en la rama general, se realiza una prueba de ejecución enfocada al módulo integrado, tomando en consideración los mismos aspectos que la prueba anterior.

8.2 Tecnologías a Utilizar

Como base de desarrollo será utilizado el lenguaje de programación C, por su robustez y amplia documentación. Una vez desarrollado el segmento de código usaremos como plataforma de manejo de código fuente, GitHub, ya que posee el protagonismo en el control de versiones a nivel mundial. Finalmente, el proyecto será ejecutado usando la plataforma de virtualización VMware, ya que con ella se realizará la emulación de sistemas operativos Unix como Ubuntu y OpenSUSE donde puedan ejecutarse las herramientas de benchmarking y de controladores SDN.

Las librerías de C, que se utilizarán para el desarrollo de las funcionalidades más importantes serán: Gnuplot_i, para la representación gráfica de resultados; la librería Libpcap, para la captura y manejo de paquetes de red; Net-SNMP para el envío de consultas SNMP hacia el controlador; Netinet/TCP para la construcción de las cabeceras TCP; time, para la manipulación de estructuras de tiempo durante la realización de la prueba, así como también para los timeouts necesarios; y por último, sockets, para la comunicación entre cliente, servidor y controlador. A continuación, se presenta una breve descripción de las herramientas y librerías mencionadas:

- **Lenguaje C:** Es un lenguaje de programación estructurado y de procedimientos que ha sido ampliamente usado tanto para sistemas operativos como para aplicaciones.
- **GitHub:** Es un controlador, basado en web, de repositorios de código fuente que permite el manejo de versiones. Provee características que dan soporte al trabajo colaborativo, manejo de bugs, asignación de tareas, entre otros.

- **VMware:** Es un hipervisor que soporta diversos sistemas operativos, entre estos, varias distribuciones de Linux, permitiendo así la existencia de diversas máquinas virtuales con diferentes configuraciones de hardware y software en una única máquina física.
- **Gnuplot_i:** Gnuplot es una herramienta de línea de comandos para la creación de gráficos de funciones matemáticas, la librería gniplit_i es un módulo que permite el envío de peticiones de despliegue de estos gráficos a través de una sesión Gnuplot en el lenguaje C.
- **Libpcap:** Permite el envío y captura de paquetes en el lenguaje C. Esto puede ser directamente de la red subyacente o a través de un archivo.
- **Net-SNMP:** Es una librería que contiene un conjunto de funciones que permiten el uso del protocolo SNMP en sus versiones 1, 2 y 3. La función más relevante es SNMPwalk, la cual permite la construcción de consultas a MIB remotas y la recepción de las correspondientes respuestas.
- **Netinet/TCP:** Librería de C que facilita estructuras de datos para el manejo de cabeceras TCP.
- **Time:** Permite la determinación del tiempo, con la mejor aproximación posible del tiempo actual, en un formato condensado.
- **Sockets**: Es una librería que permite la creación y manejo de canales generales de comunicación interprocesos, incluyendo procesos remotos, los cuales se comunican a través de una red.

9. Marco Aplicativo

Empleando la metodología descrita en el Capítulo 8, serán explicadas las actividades realizadas a lo largo del proceso de desarrollo de la solución.

9.1 Análisis General

Para cumplir con la solución de una herramienta de medición de desempeño, se realizó un estudio del funcionamiento de herramientas de evaluación de desempeño, para controladores OpenFlow ya existentes y se analizó sus oportunidades de mejora. La principal herramienta de código libre y buen reconocimiento es “Cbench”. Esta herramienta, descrita en la Sección 7.4, trabaja con la versión 1.0 del protocolo OpenFlow, proporcionando una amplia gama de parámetros para la medición de rendimiento y latencia. Su modelo de trabajo consiste en la simulación de switches virtuales que envían peticiones bajo el protocolo OpenFlow y miden el tiempo de respuesta. Aunque el método es certero, se encuentra en desventaja contra el vertiginoso avance que ha dado el desarrollo de controladores más recientes y de mejor rendimiento.

Tomando como base los fundamentos en la herramienta Cbench, es posible diseñar una herramienta más versátil. En principio, es necesario la actualización del protocolo OpenFlow, ya que la versión 1.0 usada por Cbench se considera actualmente un borrador. Con una nueva versión y controladores más modernos, se puede implementar modelos de trabajo distribuido maestro-esclavo basado en el pase de mensajes que permitan ejecutar de manera sincronizada pruebas simultaneas sobre el mismo controlador. Por último, con la intención de considerar características de hardware en la medición de desempeño se puede incluir el uso de consultas SNMP. En conjunto, estas características permiten la recolección de mucha información, información que podría ser difícil de entender y analizar sin el debido proceso, por lo que sería ideal facilitar el análisis de información incorporando la generación de gráficos que representen los resultados obtenidos.

En la Figura 9.1 se muestra un modelo de la aplicación propuesta, la cual lleva como nombre OFC-Benchmark (OpenFlow Controller Benckmark).

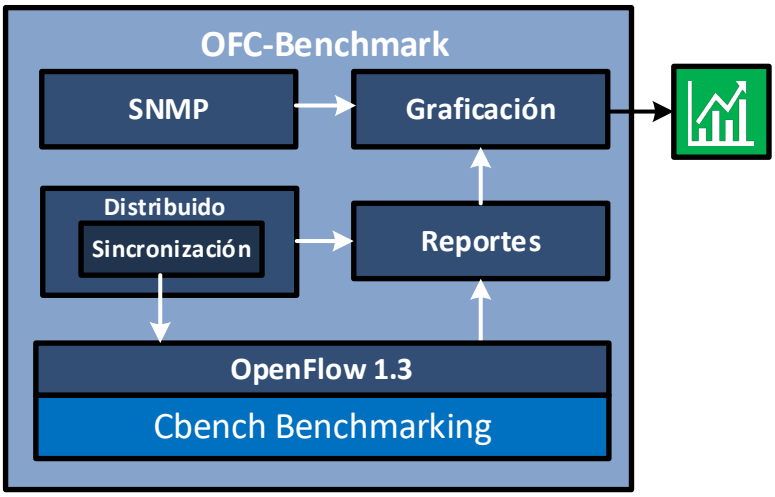

Figura 9.1: Propuesta de Medidor de Rendimiento

Para el desarrollo de la aplicación se planteó la capacidad de ejecución en dos modos, distribuido o aislado. Al momento de la ejecución de la herramienta, el modo predeterminado es el aislado, en caso contrario, debe configurarse el modo distribuido, indicando en principio la cantidad de nodos esclavos y posteriormente la dirección IP del maestro en cada uno de los nodos esclavos. En la Figura 9.2, se muestra un diagrama del modo aislado, en donde los módulos activos se presentan en verde y los inactivos en gris.

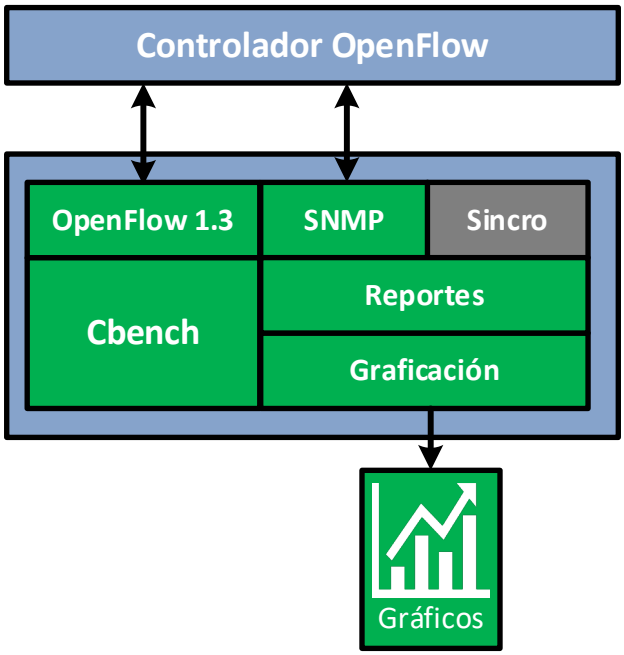

Figura 9.2: Modo Aislado

En el modo distribuido, el nodo maestro toma un papel intermediario que mantiene la sincronización entre los nodos esclavos, realiza las peticiones SNMP y maneja los reportes generados por esclavos. Para mantener a los esclavos sincronizados, se establece una conexión entre nodos con un intercambio de mensajes inicial. En

este proceso, el maestro espera por los esclavos, y cuando todos se han conectado se da la orden de inicio. Al finalizar las pruebas, los esclavos envían sus reportes al maestro. El maestro no solo recibe los reportes, este es el encargado de darle formato a los reportes en una manera que el módulo graficador pueda presentar los valores en forma de líneas, puntos y barras. Paralelo a las pruebas de estrés realizadas por los nodos esclavos, el maestro realiza las consultas SNMP. En la Figura 9.3, se define el diagrama de este modo de operación, con la interacción entre nodos, en el cual los módulos activos se muestran en verde y los inactivos en gris.

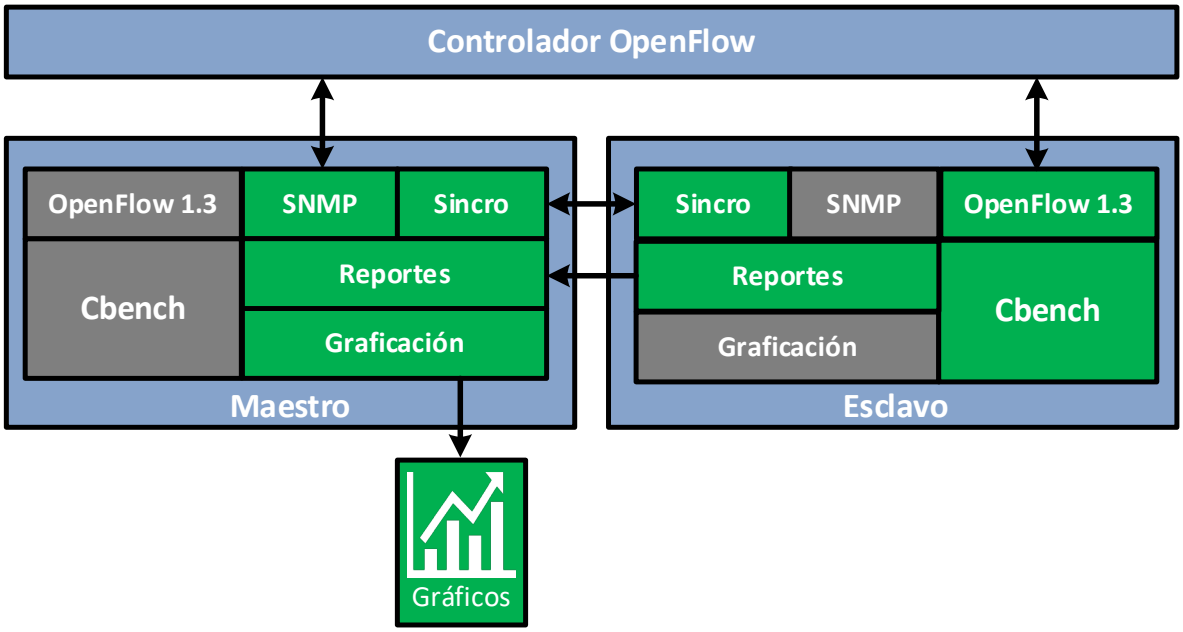

Figura 9.3: Modo Distribuido

9.2 Desarrollo de la Aplicación

Dado el esquema de trabajo planteado se estableció un grupo de funcionalidades representadas en seis módulos:

- Módulo Configuración.
- Módulo OpenFlow 1.3.
  - Manejo de mensajes OpenFlow 1.3.
- Módulo SNMP.
  - Establecimiento de sesión SNMPv2.
  - Consultas OIDs a MIB.
  - Manejo de respuestas SNMP.
- Módulo Reportes.
  - Construcción de resultados.
- Módulo Representación Gráfica.
  - Construcción de gráficos.
  - Manejo de Gnuplot.
- Módulo Distribuido.
  - Manejo de mensajes entre maestro y esclavos.

Cada módulo será desarrollado en una iteración siguiendo las fases mencionadas en el Capítulo 8. Durante dichas iteraciones se generará el código necesario para el cumplimiento de los requerimientos, y de ser necesario se realizarán más iteraciones para la corrección de errores presentados durante las pruebas o nuevos requerimientos no contemplados anteriormente.

9.2.1 Iteración 1: Módulo Configuración

Se define el proceso de creación de un módulo capaz de manejar parámetros y configuraciones sobre la aplicación, a fin de facilitar la configuración de las pruebas.

9.2.1.1 Fase de Análisis

Al realizar una prueba de desempeño, es necesario generar una configuración inicial para el establecimiento de parámetros como la dirección IP del controlador. Los valores de dichos parámetros son suministrados por el usuario durante el inicio de la prueba. El propósito del módulo desarrollado en esta iteración es encargarse del manejo de ellos durante la fase de configuración. De no colocarse algún valor, se usará el valor por defecto de la aplicación.

9.2.1.2 Fase de Diseño

Con el fin del cumplimiento de los requerimientos expuestos, se deben definir los parámetros que serán necesarios durante la ejecución de las pruebas, así como los tipos de datos en los que serán almacenados posteriormente. Estos parámetros son los que se observan en la Tabla 9.1.

Opción Corta	Opción Larga	Descripción
-C	--controller	Dirección IP del controlador OpenFlow, objeto de prueba.
-P	--port	Puerto OpenFlow al que se dirigirán las consultas.
-t	--throughput	Bandera para activar la prueba de rendimiento. Por defecto la prueba se realiza para medir latencia.
-n	--node-master	Dirección IP del nodo maestro en caso del modo distribuido.
-N	--nodes	Cantidad de nodos en el modo distribuido.
-s	--switches	Cantidad de switches virtuales a simular.
-D	--packet-delay	Milisegundos de espera entre envío de paquetes OpenFlow.
-d	--delay	Tiempo de retraso luego de la respuesta Features_Reply OpenFlow.
-O	--dpid-offset	Offset del campo datapath OpenFlow en caso del modo distribuido.

-r	--ranged-test	Bandera para realizar la prueba con un número incremental de switches.
-S	--size	Tamaño de paquetes OpenFlow.
-x	--debug	Bandera para activar la impresión de mensajes debug OpenFlow.
-L	--learn-dst-macs	Bandera para enviar ARP replies a fin de llenar la tabla con direcciones MAC de los switches virtuales.
-i	--connect-delay	Retraso en el establecimiento de conexión switches-controlador.
-I	--connect-group-size	Cantidad de switches con retraso en el establecimiento de conexión switch-controlador.
-R	--random	Bandera para realizar el envío de paquetes, aleatorio.
-m	--ms-per-test	Duración de la prueba, indicado en milisegundos.
-l	--loops	Cantidad de ciclos de prueba a ser realizados.
-w	--warmup	Ciclos a ser descartados durante de inicio de la prueba.
-c	--cooldown	Ciclos a ser descartados durante el final de la prueba.
-h	--help	Bandera para mostrar el listado posible de parámetros y la descripción.
-f	--fields	Bandera para activar el envío de paquetes modificados.
-p	--packets	Número de paquetes.
-M	--mac-addresses	Bandera para activar el uso de direcciones MAC únicas por switch.
-v	--verbose	Bandera para activar un modo que imprima por consola la información que se recibe, almacena y transfiere en modo distribuido.

Tabla 9.1: Opciones de OFC-Benchmark

9.2.1.3 Fase de Codificación

Para el manejo de parámetros, se desarrolló una librería llamada *myargs.h* con funciones internas capaces de definir valores por defecto para determinado grupo de opciones, capturar los valores introducidos a través de la terminal durante la ejecución de la aplicación y mostrar las opciones capturadas. La lista de parámetros se hizo para ser fácilmente configurable en futuras ocasiones al inicio de la aplicación, sólo hace falta establecer el tipo de parámetro, la descripción del parámetro en caso de que un usuario la necesite, y el comando del parámetro.

Los parámetros pueden ser omitidos por el usuario, sin embargo, la aplicación maneja valores por defecto para estos casos, a fin de evitar fallos por descuido.

Tampoco importa el orden con el que sean introducidas las opciones y parámetros siempre y cuando estén prelados por la bandera correspondiente.

9.2.1.4 Fase de Pruebas

Se revisa el funcionamiento de este módulo mediante pruebas ingresando múltiple combinación de parámetros. También se busca probar situaciones bordes donde ningún parámetro era introducido o los parámetros eran introducidos con valores inválidos, ej.: -N a. En estos casos, la aplicación señala el error de manera legible y finaliza su ejecución. Un ejemplo del uso de este módulo se muestra en la Figura 9.4.

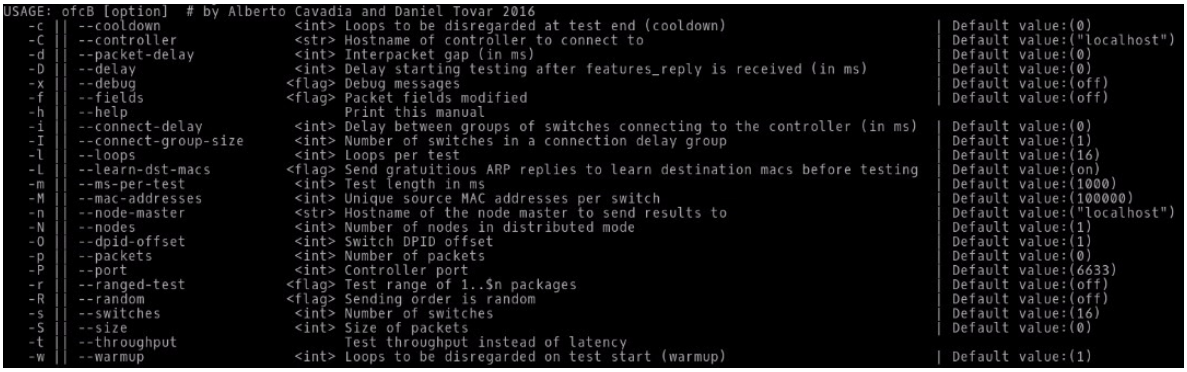

Figura 9.4: OFC-Benchmark Opción --help

9.2.2 Iteración 2: Módulo OpenFlow 1.3

Se representa el proceso de adaptación sobre Cbench para manejar los mensajes OpenFlow en su versión 1.3.

9.2.2.1 Fase de Análisis

El código base de Cbench es bastante simple, puesto que OpenFlow en su versión 1.0 requiere de pocos mensajes para el establecimiento de conexiones. En la versión 1.3, la aparición de nuevos mensajes implicó el desarrollo de nuevas funciones capaces de manejar el envío, recepción y construcción de paquetes alineados a los nuevos formatos especificados.

9.2.2.2 Fase de Diseño

Utilizando la especificación OpenFlow 1.3.1 y la especificación OpenFlow 1.0, se puede realizar un mapeo de aquellos mensajes recientemente añadidos y las reformas sobre los anteriores. Inicialmente el paso más importante es el establecimiento de una conexión OpenFlow como la expuesta en la Figura 9.5. Posteriormente comienza el proceso de análisis para el mensaje OpenFlow más importante, Packet-In, pues es el mensaje que genera una respuesta medible del controlador.

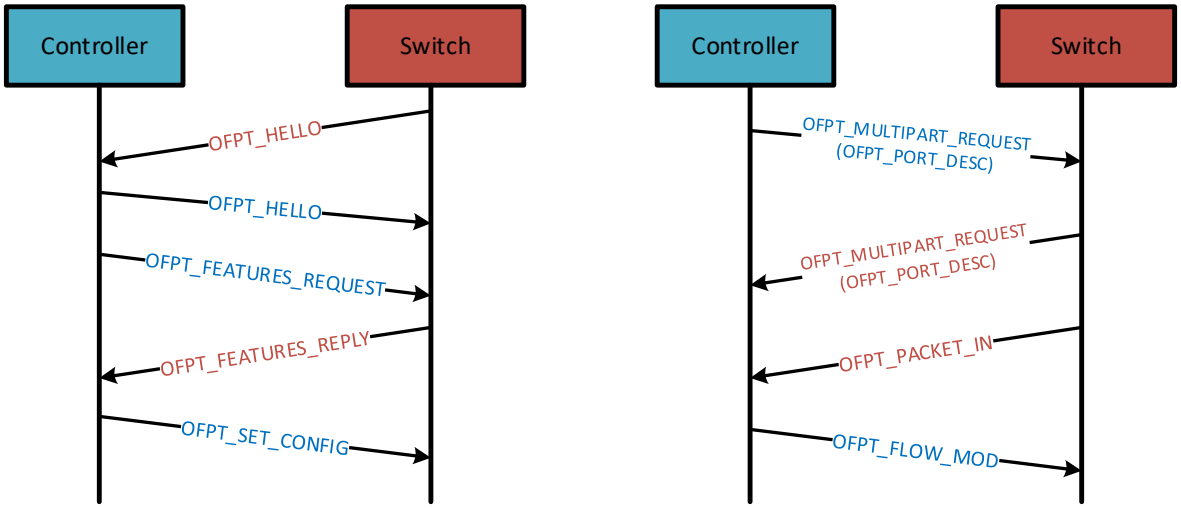

Figura 9.5: OpenFlow 1.3 Handshake

9.2.2.3 Fase de Codificación

El primer paso fue importar la interfaz en C de la especificación OpenFlow 1.3 a la librería *myswitch.h* creada por Cbench para el manejo de switches virtuales. La interfaz dio acceso a las estructuras de cabeceras y payloads necesarias para construir cualquier tipo de mensaje perteneciente al protocolo. Una vez identificado los mensajes, se actualizó el manejador de mensajes recibidos para adaptarse al nuevo handshake del protocolo y responder la nueva versión del mensaje Packet-In.

El desarrollo de las funciones asociadas a la construcción de mensajes involucró, analizar la cabecera de los mensajes y llenar la estructura adecuada, usando la especificación del protocolo y los parámetros recibidos. El reto de esta fase se encontró en la alineación de bytes necesaria para mantenerse dentro de los segmentos de memoria correspondiente.

Se decidió no implementar la lista completa de mensajes OpenFlow, sino en cambio, solo lo necesarios para poder enviar mensajes Packet-In y recibir Packet-Out. Sin embargo, se estableció un mecanismo de impresión para indicar el código de los mensajes recibido e ignorados.

9.2.2.4 Fase de Pruebas

En esta iteración, se realizan constantes pruebas sobre los paquetes recibidos y enviados, con distintos controladores OpenFlow (Floodlight, Ryu, OpenMUL, OpenDaylight). No todos los controladores responden y solicitan los mismos mensajes, fue necesario experimentar con el espectro más grande posible. La inspección se realizó a través de Wireshark, una herramienta muy conocida para la captura de paquetes. Al observar un paquete mal construido, se iniciaba un proceso de análisis sobre la información almacenada en los rangos de memoria correspondientes a los campos de las cabeceras OpenFlow. Un ejemplo exitoso resultado de la inspección que provee Wireshark se muestra en la Figura 9.6.

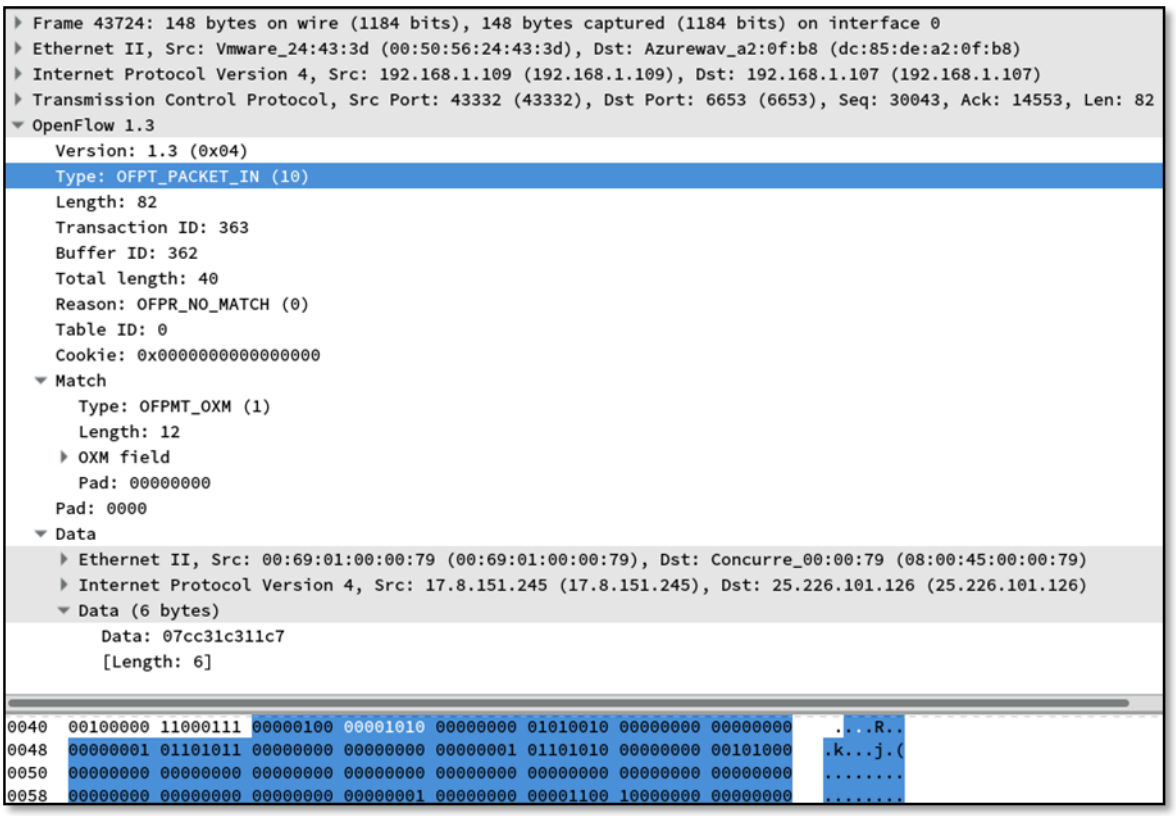

Figura 9.6: Captura de Mensaje Packet-In

9.2.3 Iteración 3: Módulo SNMP

Se desarrollan las funciones encargadas de realizar consultas y recepción de mensajes SNMP.

9.2.3.1 Fase de Análisis

En la creación de este módulo, se tomó como base el uso de la librería en C de NET-SNMP. A pesar de ser una librería muy extensa, se hizo uso principalmente de tres funcionalidades, el manejo de sesiones, envío de peticiones SNMP con ciertos OIDs y la recepción de su correspondiente respuesta. Las funcionalidades previstas por NET-SNMP permiten medir ciertos indicadores como memoria RAM, CPU y el desempeño de red sobre determinado equipo, en esta ocasión se toman indicadores sobre el controlador OpenFlow.

Con el propósito de explotar esta funcionalidad se establece como pre-requisito, la presencia del servicio SNMP en el puerto 161 con la comunidad “public” habilitada y las configuraciones de seguridad necesarias.

9.2.3.2 Fase de Diseño

El módulo de SNMP debe de trabajar en paralelo con el módulo distribuido, lo cual se soluciona implementando hilos para ambos procesos. Para los fines de la presente iteración, se explicará el funcionamiento del hilo correspondiente a SNMP.

Al inicio de sesión, se utiliza la dirección IP suministrada en el nodo maestro durante el inicio de la prueba junto con la comunidad “public” y el puerto 161. Una vez establecida la sesión, se retorna un descriptor de la misma para la realización de operaciones.

Al establecer la sesión SNMP, se realizan las consultas hacia el controlador, estas son respondidas según el OID suministrado en las mismas. Los OIDs se encuentran almacenados estáticamente en el código, y los mismos están orientados a consultas de máquina Linux de una distribución basada en Debian. Se corre una iteración en donde se consulta cada OID y se almacena el resultado en una estructura del tipo reporte antes de dormir durante 1 segundo, para repetir el proceso una vez más. Esto sucede indefinidamente hasta recibir la lista completa de reportes. Por último, se envía al módulo de representación gráfica el contenido de la estructura reporte que se obtuvo como resultado de las iteraciones del hilo SNMP. En la Figura 9.7 se muestra el proceso explicado con anterioridad.

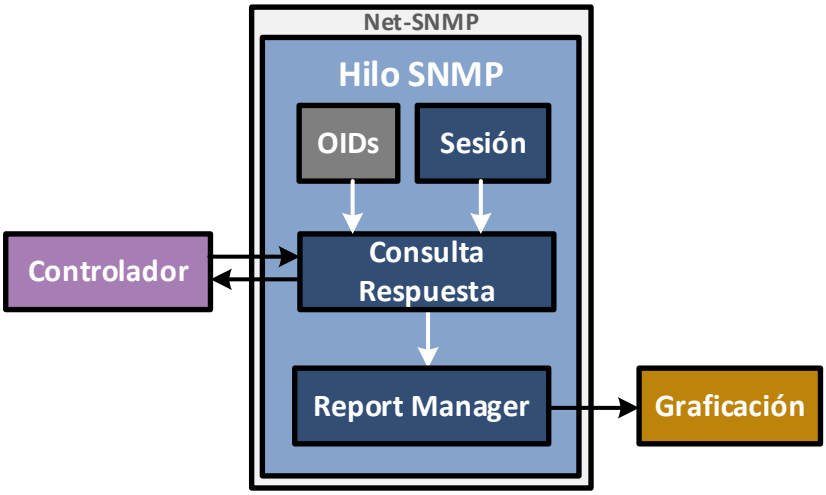

Figura 9.7: Módulo de Consulta SNMP

9.2.3.3 Fase de Codificación

Durante esta fase se concretó el desarrollo de lo definido en la fase de diseño, se crearon funciones para la realización de consultas SNMP y para el manejo de las respuestas, las cuales debieron ser formateadas para tomar valores porcentuales en el caso de memoria y CPU, en el caso de red se tomaron los valores en Bytes y tradujeron a Kilobytes. El cálculo de porcentajes involucró peticiones con OIDS adicionales, como fue el caso de memoria.

Una vez se tenían todos los valores, se almacenaba el valor aunado a un timestamp para ser utilizado para representar el instante de tiempo en el que se obtuvo ese valor, con él módulo de representación gráfica.

9.2.3.4 Fase de Pruebas

La realización de las pruebas necesarias para validar el cumplimiento de los requerimientos propuestos implicó la ejecución del módulo, aislado del resto. La prueba consistió en la impresión de lo que serían los resultados en cada segundo a través de la pantalla, como se puede ver en la Figura 9.8. Se comparó este resultado con el retornado usando comandos nativos de NET-SNMP en la terminal, como es el caso de *snmpget*. Posteriormente, se realizó la prueba del formateo de los resultados, igualmente realizando impresiones en consola de los resultados obtenidos.

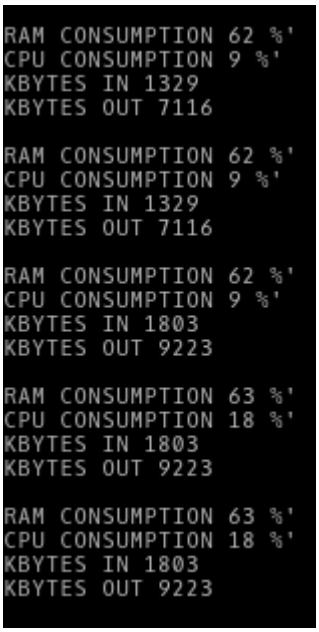

Figura 9.8: Resultados SNMP

9.2.4 Iteración 4: Módulo Reportes

Se desarrolló una estructura capaz de proveer una forma eficiente y reutilizable de recibir y emitir la información generada de pruebas.

9.2.4.1 Fase de Análisis

Tomando en cuenta que los valores de las pruebas pueden ser orientados a ser enviados a través de sockets o recibidos como parte de peticiones SNMP, se decidió crear una estructura que permita almacenar la posible información necesaria para manejar cómodamente el resultado de las pruebas al momento de graficar o enviar la información.

9.2.4.2 Fase de Diseño

Como se observa en la Figura 9.9, la estructura de un reporte posee acceso a la dirección IP del dispositivo que almacena el reporte, el socket si este fue enviado a través de uno, y un arreglo de colas doblemente enlazadas compuesta de apuntadores a estructuras de mensajes.

```
typedef struct message {
  char *buffer;
  struct message *next;
  struct message *back;
} message;

typedef struct queue {
  int length;
  struct message *first;
  struct message *last;
} queue;

typedef struct report{
  int sock;
  char *hostname;
  struct queue queues[MAX_QUEUE];
 } report;
```

Figura 9.9: Estructura de un Reporte, Cola y Mensaje

9.2.4.3 Fase de Codificación

Se diseñaron funciones que proveen interfaces para realizar acciones sobre el reporte, funciones capaces de almacenar segmentos de caracteres en forma de apuntadores a *char*, y funciones capaces también de mostrar por pantalla los elementos de un reporte. Se estableció que el reporte tendría un tamaño estático de 5 colas basado en la categorización de los resultados. Una prueba almacena en intervalos de 1 segundo, el instante de tiempo, los flujos que recibió cada switch en ese instante, el promedio de los flujos recibidos entre todos los switches, los resultados de las consultas SNMP y el resultado final con el máximo, mínimo, promedio y desviación de flujos recibidos. Cada categoría es manejada en una cola aislada.

Los mensajes, definidos cómo *messages* en la Figura 9.9, fueron diseñados y usados como buffer temporal de la información recabada en un instante de tiempo, por ello, el uso de un mensaje siempre involucra el uso posterior de los mensajes anteriores y posteriores a él. En el contexto de desarrollo, el mensaje facilita la concatenación de valores en un mensaje final con determinado formato. Un ejemplo de su uso puede observarse en la Figura 9.10, donde es usado para enviar todos los resultados de la prueba que recolectó un nodo a través de un socket.

```
//Sending Results
temp = myreport->queues[RESULTS].first;
while (temp != NULL) {
  bytes = writeSocket(serverFd, temp->buffer, strlen(temp->buffer), strlen(temp->buffer));
  temp = temp->back;
}
```

Figura 9.10: Envío de Mensajes

9.2.4.4 Fase de Prueba

Se provee una estructura con lineamientos, propósitos y flexibilidad de usos que ha exigido cambios constantes, hasta obtener el producto actual. Los cambios más importantes se realizaron durante el envío de mensajes por socket, y la concatenación de información para la representación gráfica. La estructura es usada para manejar la información de múltiples gráficos en paralelo, con reutilización de colas, manejo de semáforos y redimensionamiento de apuntadores a *char*.

9.2.5 Iteración 5: Módulo de Representación Gráfica

Provee funcionalidades que permiten recibir y formatear los reportes de resultados para la elaboración de los gráficos.

9.2.5.1 Fase de Análisis

El principal objetivo fue delegado al buen uso de la librería Gnuplot_i, que permite la interacción con la herramienta Gnuplot encargada de la realización de gráficos. Todo esto tomando como entrada la lista de cadenas de caracteres obtenida del módulo de reportes.

9.2.5.2 Fase de Diseño

La representación gráfica está compuesta por funciones encargadas de extraer los datos resultantes, de reportes y darles el formato adecuado para facilitar la conversión de valores en serie de puntos a representar en el gráfico con las coordenadas [x, y].

Gnuplot_i, es una librería que forma la interfaz con la herramienta del sistema Gnuplot para generar los gráficos construidos base a los reportes, en formato png. En la Figura 9.11 se muestra el proceso mencionado.

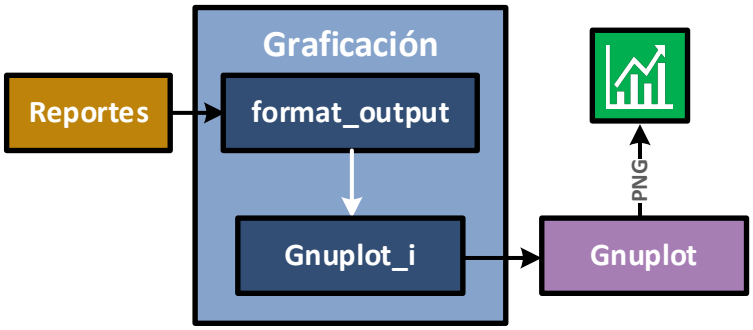

Figura 9.11: Módulo para la Representación Gráfica

9.2.5.3 Fase de Codificación

El desarrollo de las funciones pertinentes culmina con la construcción de una cadena de caracteres con un formato separado por comas. Dicha cadena tiene como finalidad facilitar la extracción de coordenadas [x, y] cómo se observa en la Figura 9.12. Los valores extraídos, son usados para alimentar la función *gnuplot_plot_xy* de la librería Gnuplot_i como lo muestra el código de la Figura 9.13.

```
while (input[offset] != ';'){
  //INITIALIZE text for number
  for(j = 0; j < 10; j++){
    numberText[j] = '\0';
  }

  j = 0;
  //READ number char by char
  while ((input[offset] != ',') && ( input[offset] != ';')){
    numberText[j] = input[offset];
    offset++;
    j++;
  }
  //TRANSLATE number from char* to float
  number = atof(numberText);
  flows->x[i] = number;
  flows = flows->next;

  if (input[offset] == ';') {
    i++;
    flows = checkpoint;
  }
  offset++;
}
```

Figura 9.12: Formateo de un Reporte

```
printf("graphication \n");
resultsIterator = checkpoint->next;
snprintf(command, 150, "set output \"../reports/charts/%s.png\"", name);

while (resultsIterator != NULL) {
  gnuplot_cmd(h1, command);
  gnuplot_plot_xy(h1, checkpoint->x, resultsIterator->x, i, resultsIterator->name);
  resultsIterator = resultsIterator->next;
}
```

Figura 9.13: Generación de un Gráfico

9.2.5.4 Fase de Pruebas

Para probar el correcto funcionamiento del módulo correspondiente a esta iteración, fue la generación de gráficos dada una entrada a la función, la cual se generó de manera manual, con el fin de tener un comportamiento controlado y un resultado conocido.

La prueba se llevó a cabo y los resultados obtenidos fueron los esperados para la generación de gráficos usando valores SNMP, resultados por nodo, y resultado final de la prueba realizada.

9.2.6 Iteración 6: Módulo Distribuido

Provee el proceso de comunicación necesario para la sincronización entre los nodos esclavos y el nodo maestro.

9.2.6.1 Fase de Análisis

En caso de querer generar una herramienta capaz de trabajar horizontalmente con instancias pares, es deseable poseer un mecanismo de sincronización. Con este, la herramienta puede ejecutarse en paralelo a través de múltiples instancias, pero ¿qué sucedería con todos los resultados? Deben unirse en algún punto de la prueba.

9.2.6.2 Fase de Diseño

Para mantener sincronizado múltiples instancias de OFC-Benchmark se estableció una serie de mensajes. Mensajes que una vez intercambiados, quedarán a un lado, hasta culminar la prueba donde cada nodo esclavo enviará la información obtenida en su prueba, el intercambio total de mensajes es ejemplificado en la Figura 9.14.

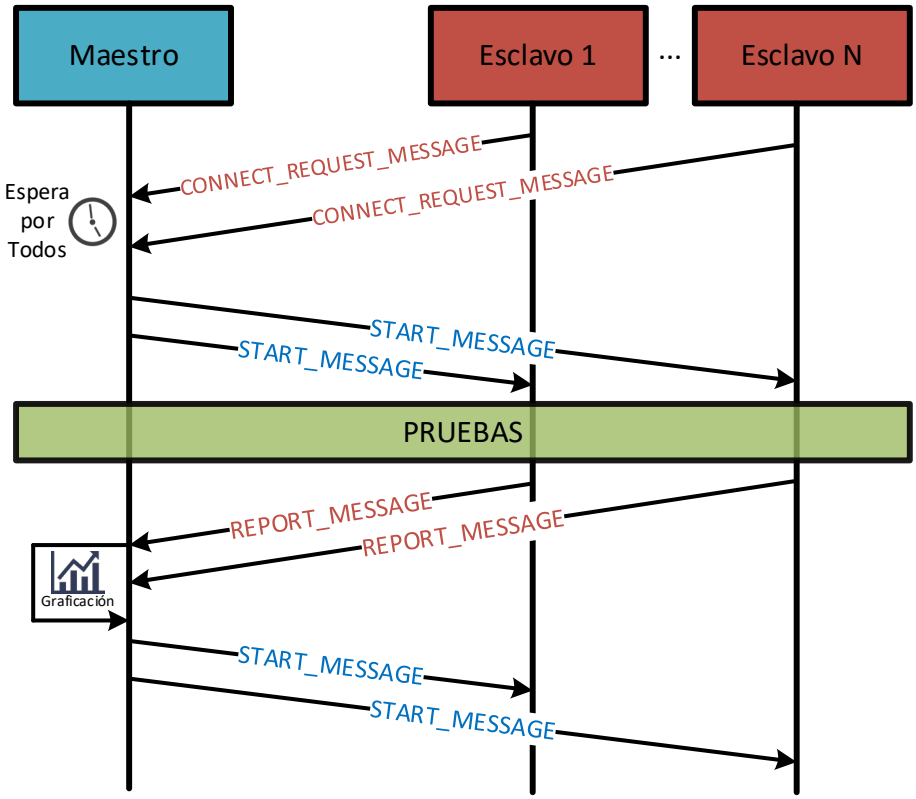

Figura 9.14: Intercambio de Mensajes Maestro - Esclavo

9.2.6.3 Fase de Codificación

Usando sockets se definieron mensajes representados en forma de identificadores numéricos. Cada identificador se almacena en una estructura de mensaje, y este es enviado esperando la respuesta correspondiente.

Para el manejo de mensajes, se crearon hilos dedicados a la recepción y envío de mensajes. El nodo maestro crea un hilo por cada nodo esclavo, que le permita manejar los mensajes de determinado nodo esclavo sin retrasar el manejo de otros nodos esclavos. El nodo esclavo en cambio necesita de una única función capaz de evaluar la recepción y envío de mensajes. El proceso de manejar la recepción de mensajes se puede observar en la Figura 9.15.

```
buffer = readSocket(serverFd, 1, 1, &bytes);

if (strcmp(buffer, START_MESSAGE) == 0) {
  end = 1;

  printf("received START_MESSAGE\n");
  controllerBenchmarking();
  displayMessages(myreport, VALUES);
  displayMessages(myreport, AVGS);
  displayMessages(myreport, RESULTS);

  snprintf(buffer, strlen(REPORT_MESSAGE) + 1, REPORT_MESSAGE);
  writeSocket(serverFd, buffer, 2 , 1);
  printf("sent REPORT_MESSAGE\n");

  //Sending Values
  temp = myreport->queues[VALUES].first;
  while (temp != NULL) {
    bytes = writeSocket(serverFd, temp->buffer, strlen(temp->buffer), strlen(temp->buffer));
    temp = temp->back;
  }
```

Figura 9.15: Manejo del Mensaje START por un Nodo Esclavo

La lectura y escritura del socket se manejó con funciones extendidas sobre las implementaciones nativas de la librería socket, definidas como funciones personalizadas para escribir y leer carácter por carácter hasta encontrar caracteres particulares que dictaminan el final de la línea a leer. Esto resulta ser útil para el módulo reporte ya descrito, puesto que permite realizar una lectura cómoda siempre y cuando el emisor haya cumplido con el formato deseado.

9.2.6.4 Fase de Pruebas

Este módulo fue probado mediante la impresión de mensajes escritos y leídos durante las fases de handshake e intercambio de reportes.

9.3 Pruebas Generales

En esta sección se describen las pruebas que se realizaron a la herramienta, con el fin de evaluar el cumplimiento de los objetivos, en las áreas de funcionalidades y compatibilidad OpenFlow 1.3. Así como también, se evaluó la integridad de estos resultados, realizando una comparación con herramientas afines que existen en el mercado.

Para estas pruebas se configuraron 16 switches simulados, por cada nodo esclavo. Realizando 50 ciclos donde 1 ciclo es equivalente a 1 segundo.

9.3.1 Pruebas de Funcionalidades

Para la verificación del correcto funcionamiento de la aplicación en términos de las funcionalidades definidas, se desarrollaron dos pruebas. La primera cumple con la verificación del modo aislado, y una segunda que permite la verificación del modo distribuido, tomando como configuración el controlador SDN (OpenMUL en este caso) y 4 nodos (1 maestro y 3 esclavos).

9.3.1.1 Prueba de Modo Aislado

Se efectuaron pruebas con la existencia de un único nodo, en la que se utilizaron las opciones mencionadas en la Sección 9.3, para corroborar las funcionalidades de la herramienta. Se obtuvieron los resultados mostrados en las Figuras desde la 9.16 hasta la 9.20.

La Figura 9.16 muestra el comportamiento del controlador SDN a nivel de respuestas a los Packets-In recibidos por el mismo, a lo largo de la prueba. Se puede apreciar la cantidad de flujos que fueron respondidos, para cada switch individualmente, en un segundo, estos flujos representan la cantidad de Packets-Out generados por el controlador. Al reflejarse cada switch individualmente, se obtiene información tal como la preferencia de respuestas para un switch en específico o la uniformidad de las mismas.

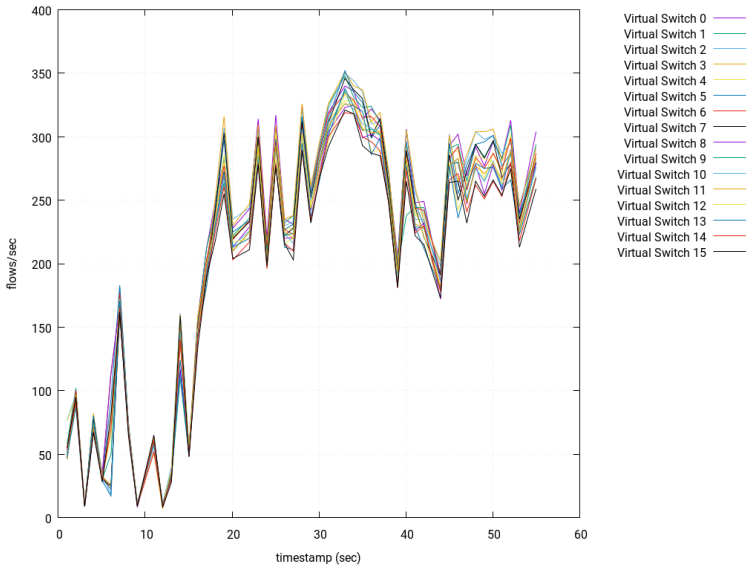

Figura 9.16: Flujos/seg por Switch – N1

La Figura 9.17 muestra el consumo de memoria y de CPU en el computador que corre el controlador SDN (OpenMUL) durante el transcurso de la prueba. Ambos parámetros son expresados como porcentajes del total de memoria y de CPU. Vale la pena recordar que la memoria total de dicho computador es de 4GB y que el modelo del CPU es un Intel i5 con 4 núcleos.

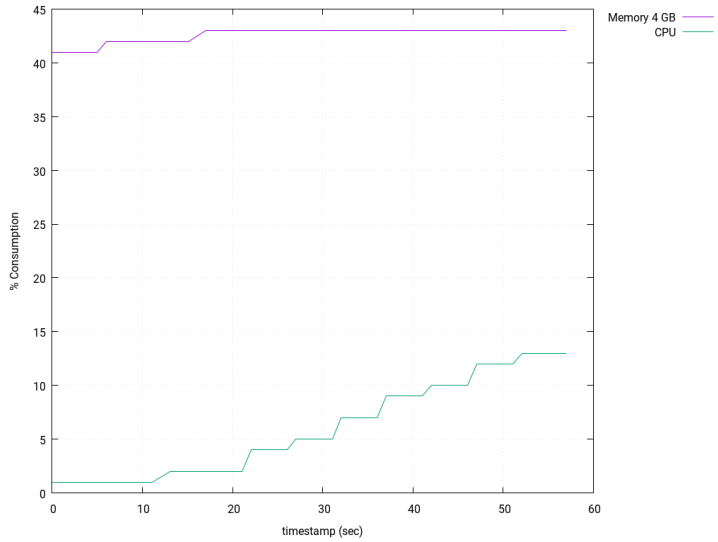

Figura 9.17: Consumo de Memoria y CPU – N1

La Figura 9.18 muestra el uso de la interfaz de red del controlador SDN, durante la prueba. Este uso es representado como la cantidad acumulada de paquetes enviados y recibidos por el controlador. En dicha figura, un aumento en la pendiente de la curva indicaría un aumento en la cantidad de paquetes recibidos o enviados, lo cual es directamente proporcional al uso de la interfaz de red. Mientras que una disminución en la pendiente representaría exactamente lo contrario.

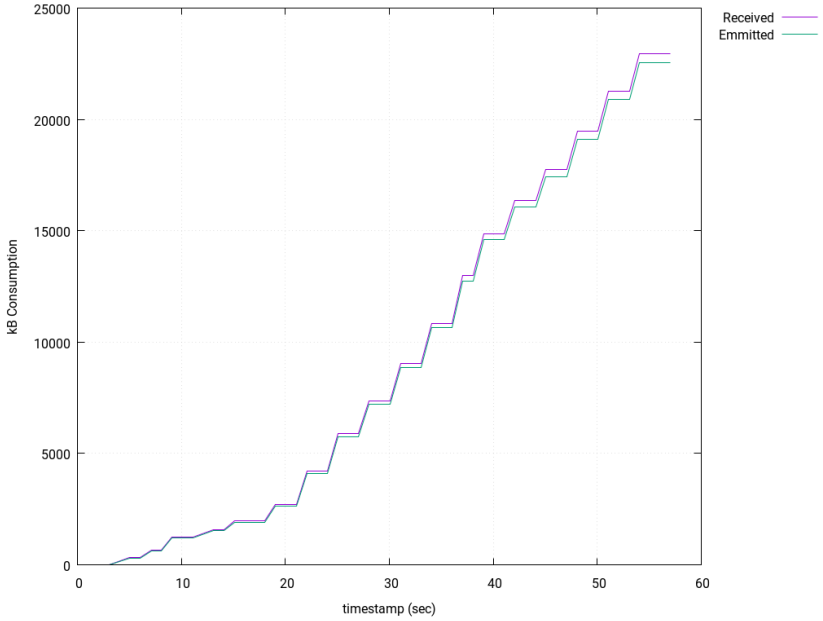

Figura 9.18: Consumo y Emisión de kB por Interfaces – N1

La Figura 9.19 muestra la cantidad de flujos por milisegundo. Lo que indica los flujos que el nodo esclavo recibió del controlador SDN, esto representa la cantidad de flujos que soporta el controlador en cada ciclo (1 segundo).

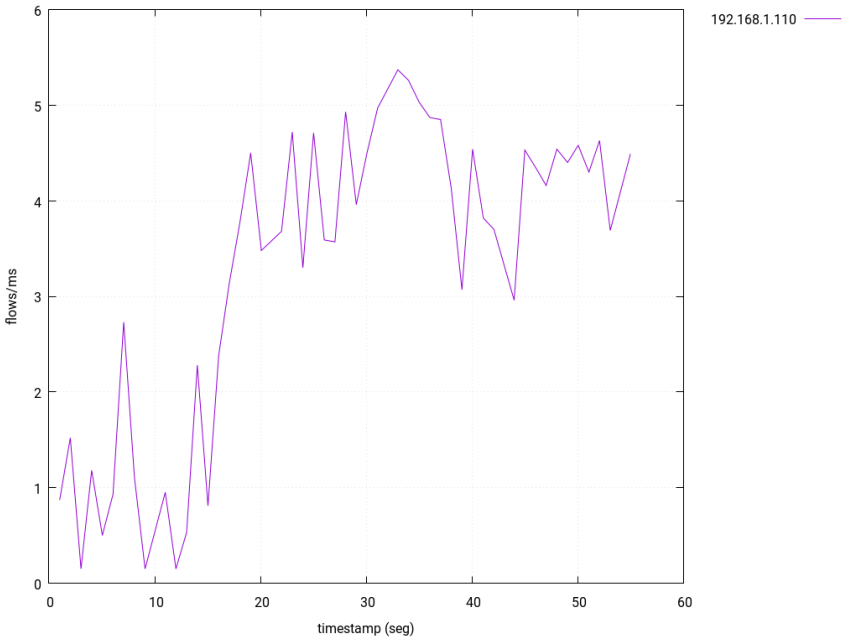

Figura 9.19: Promedio de Flujos/ms por Nodo – N1

La Figura 9.20 muestra la cantidad máxima, mínima, promedio y desviación estándar de milisegundos por flujo. Para la representación de estos valores, la herramienta consideró el acumulado total de flujos que el nodo esclavo recibió del controlador SDN (OpenMUL). El conjunto de estos valores representa el resultado final de la prueba de medición de latencia. En el caso contrario (prueba de medición de rendimiento) los valores son representados en unidad de flujos por milisegundo.

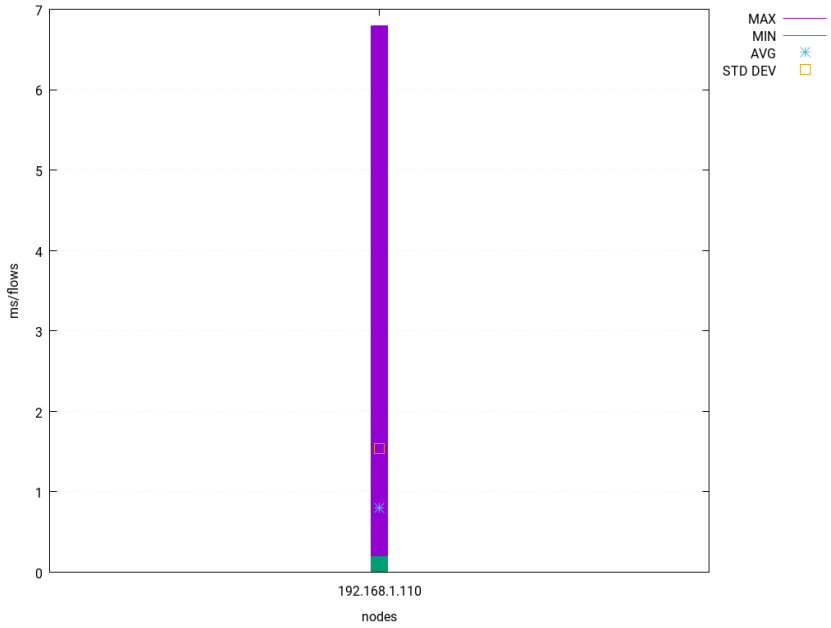

Figura 9.20: Máximo, Mínimo, Promedio y Desviación de Flujos/ms – N1

9.3.1.2 Prueba de Modo Distribuido

Se efectuaron pruebas con la existencia de 4 nodos (1 maestro y 3 esclavos). El maestro fue el encargado de la sincronización con el resto de los nodos, y la medición de indicadores SNMP. Los nodos esclavos se encargaron de la realización de las pruebas y el envío de resultados. Los resultados de estas pruebas se muestran en las Figuras desde la 9.21 hasta la 9.27.

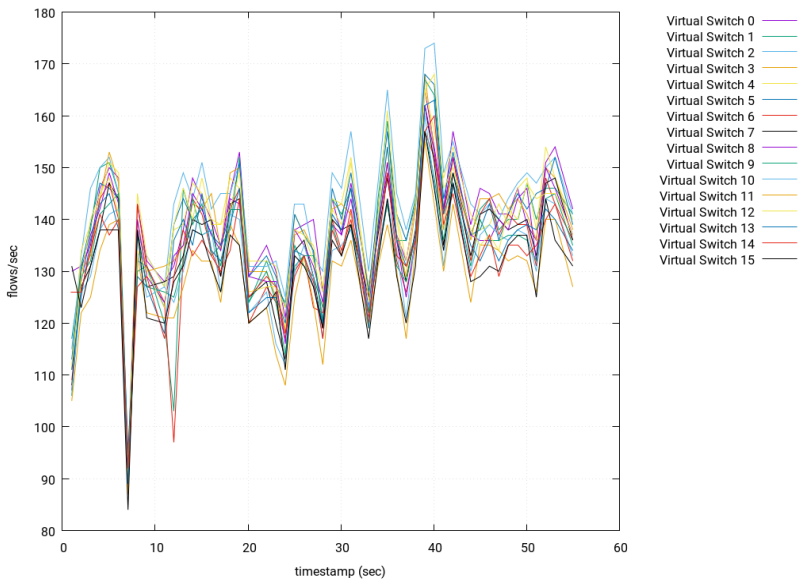

Figura 9.21: Flujos/seg por Switch - Nodo 1 – N4

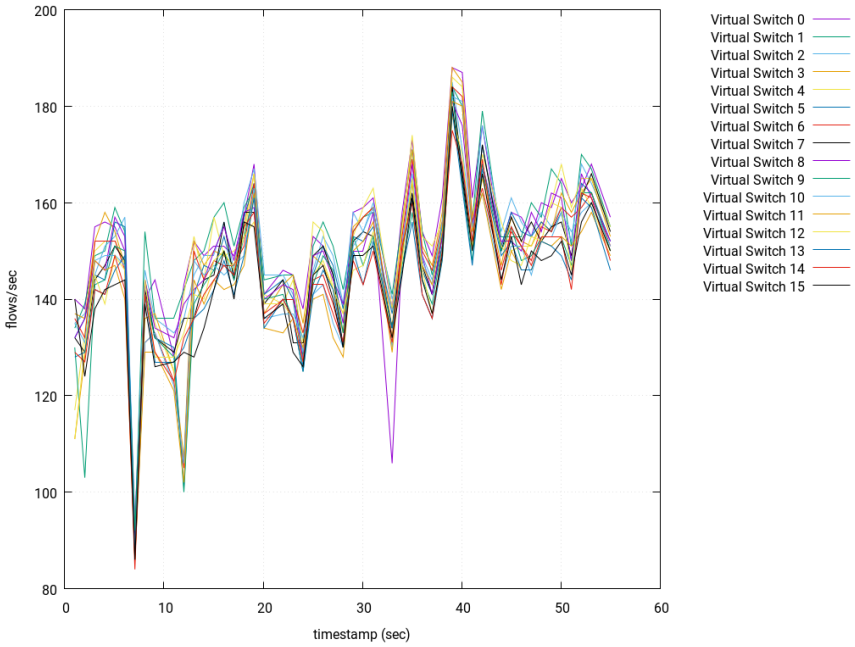

Figura 9.22: Flujos/seg por Switch - Nodo 2 – N4

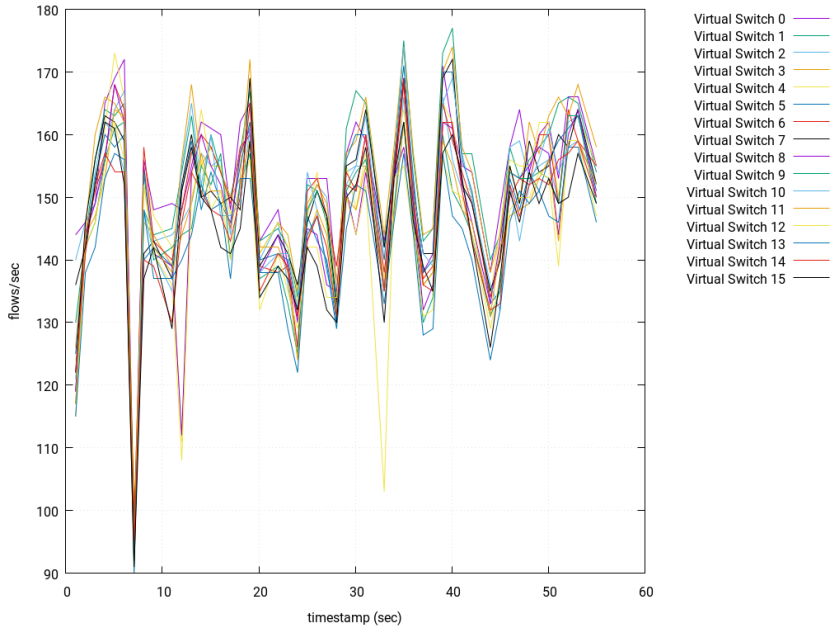

Figura 9.23: Flujos/seg por Switch - Nodo 3 – N4

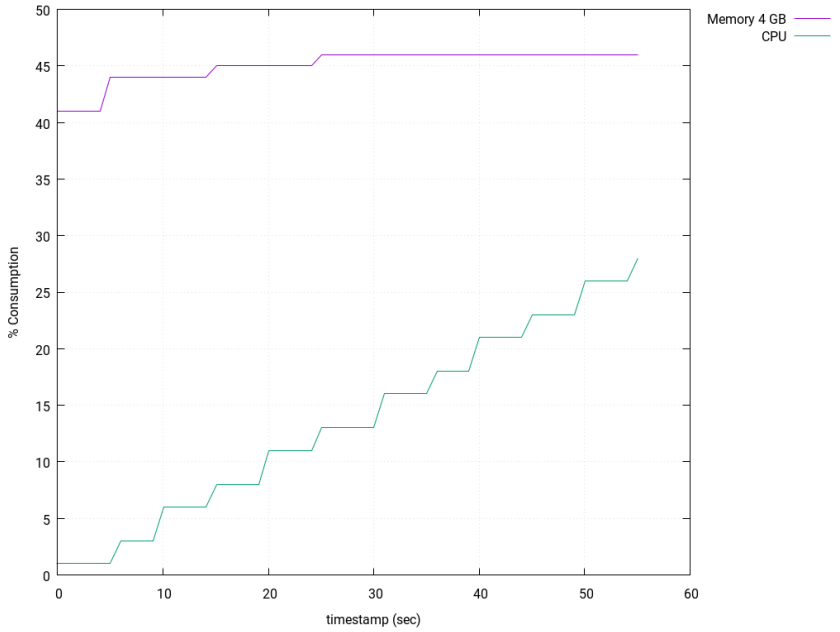

Figura 9.24: Consumo de Memoria y CPU – N4

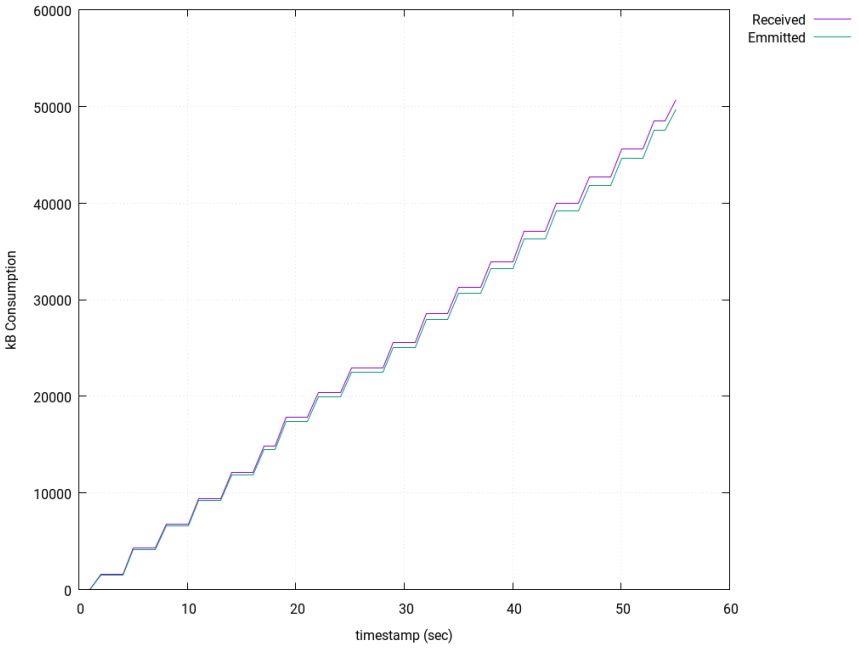

Figura 9.25: Consumo y Emisión de kB por Interfaces – N4

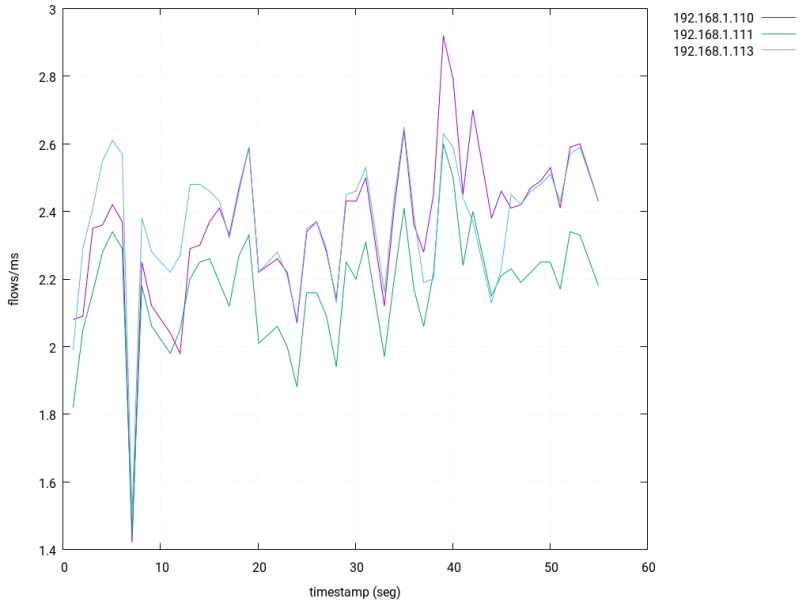

Figura 9.26: Promedio de Flujos/ms – N4

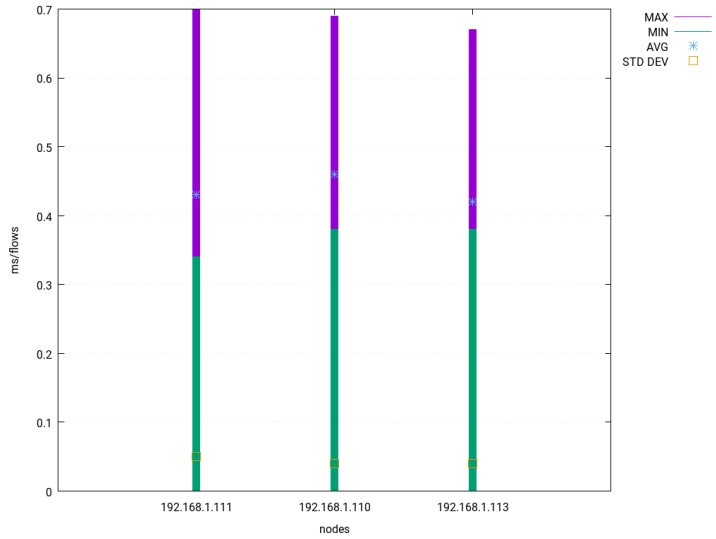

Figura 9.27: Máximo, Mínimo, Promedio y Desviación de Flujos/ms – N4

9.3.2 Prueba de Compatibilidad

Para evaluar la compatibilidad de la herramienta desarrollada se ideó la realización de una prueba con cada uno de los controladores elegidos durante la investigación: OpenDaylight, Floodlight, OpenMUL y Ryu, todos configurados para operar bajo el protocolo OpenFlow en su versión 1.3 en una prueba con 4 nodos (1 maestro y 3 esclavos).

Con el fin de tener una comparativa de estos controladores, se realizó la instalación de cada controlador en el mismo ambiente con las mismas características. Este, es una máquina de 4GB de memoria y un CPU de 4 núcleos,

en el que corre un sistema operativo Ubuntu 14.04 sobre el cual se instaló un cliente SNMP.

9.3.2.1 Prueba con Controlador OpenMUL

La prueba que se realizó sobre el controlador OpenMUL fue exitosa, se logró la conexión con el controlador en el puerto por defecto del mismo 6653, los mensajes OpenFlow fueron respondidos correctamente y a una tasa esperada. Los resultados de esta prueba se muestran en las Figuras desde la 9.21 hasta la 9.27 de la Sección 9.3.1.2. Para esta prueba se ejecutó el controlador OpenMUL con el comando mostrado en la Figura 9.28.

```
./mul.sh start l2switch
```

Figura 9.28: Comando para la Ejecución de OpenMUL

9.3.2.2 Prueba con Controlador Floodlight

El controlador Floodlight se ejecutó mediante la utilización del comando mostrado en la Figura 9.29.

```
java -jar target/Floodlight.jar
```

Figura 9.29: Comando para la Ejecución de Floodlight

El controlador Floodlight, fue utilizado durante el desarrollo y prueba de cada funcionalidad. Es el primer controlador en alcanzar el 100% de compatibilidad con las funcionalidades de OFC-Benchmark. Durante la prueba final, el controlador respondió correctamente a los paquetes Packet-In de OpenFlow 1.3, y los resultados se muestran en las Figuras desde la 9.30 hasta la 9.36.

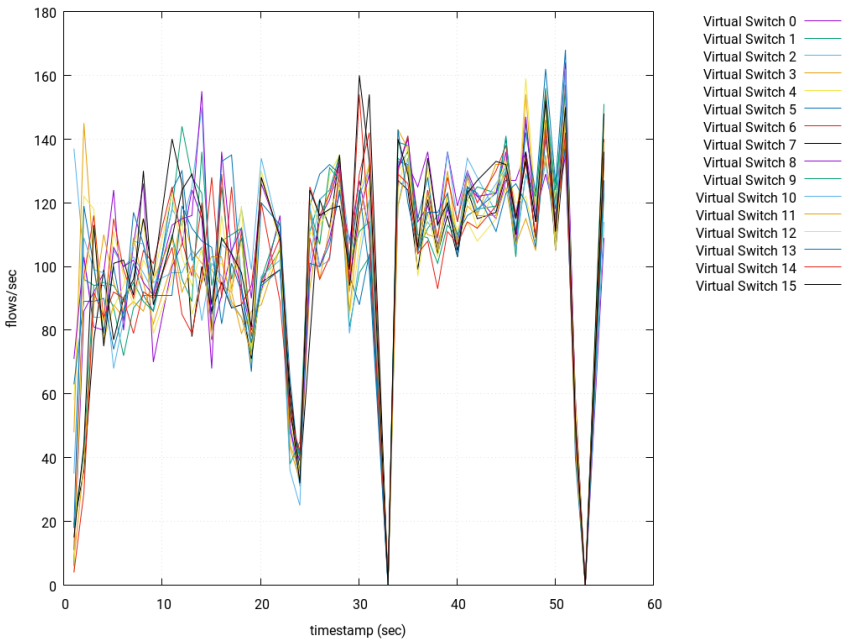

Figura 9.30: Flujos/seg por Switch – Nodo 1 – Floodlight – N4

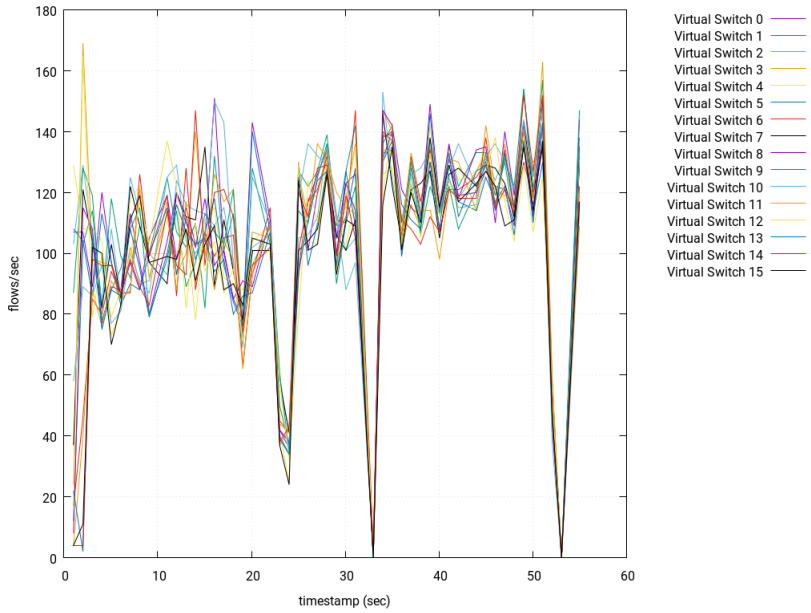

Figura 9.31: Flujos/seg por Switch – Nodo 2 – Floodlight – N4

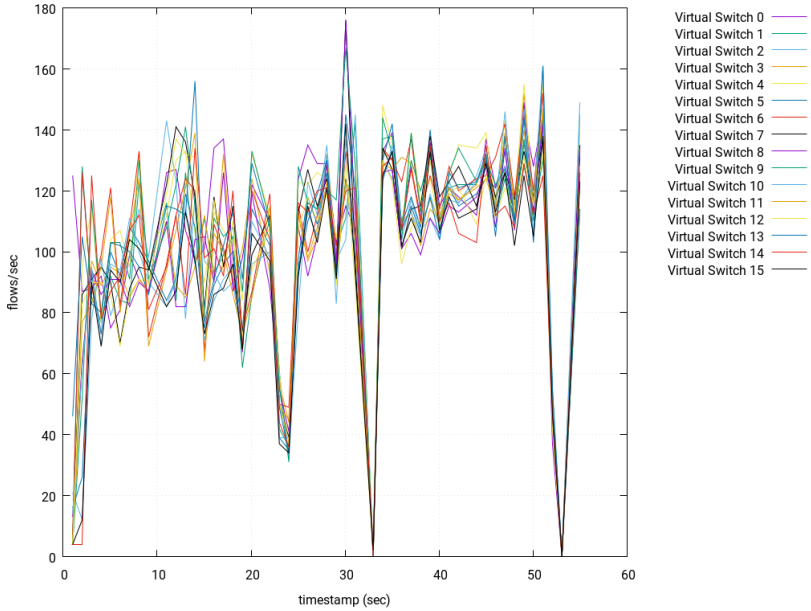

Figura 9.32: Flujos/seg por Switch – Nodo 3 – Floodlight – N4

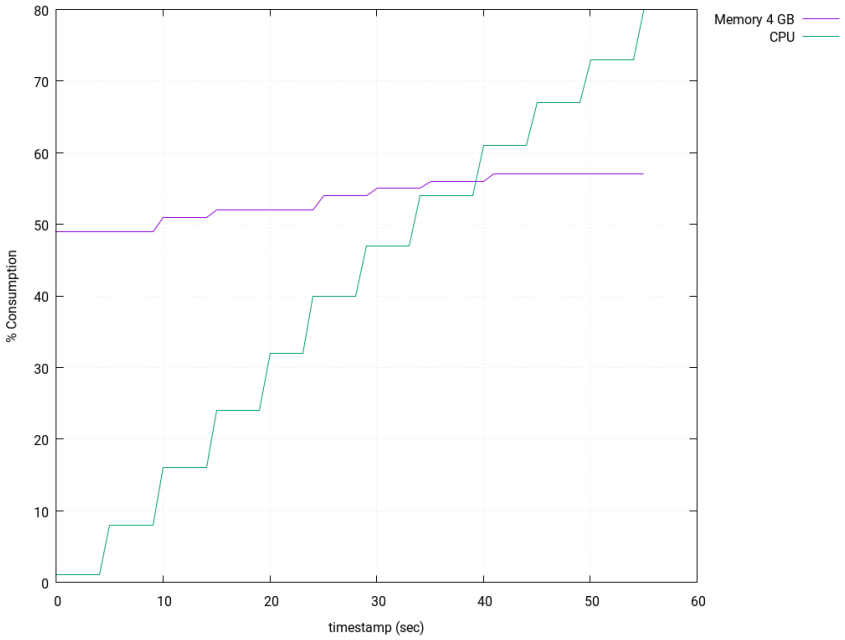

Figura 9.33: Consumo de Memoria y CPU – Floodlight – N4

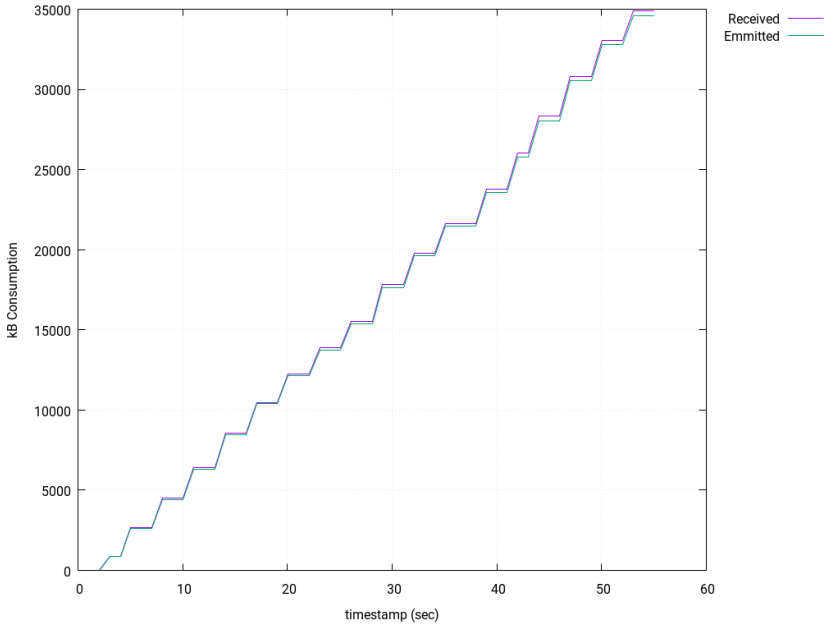

Figura 9.34: Consumo y Emisión de kB por Interfaces – Floodlight – N4

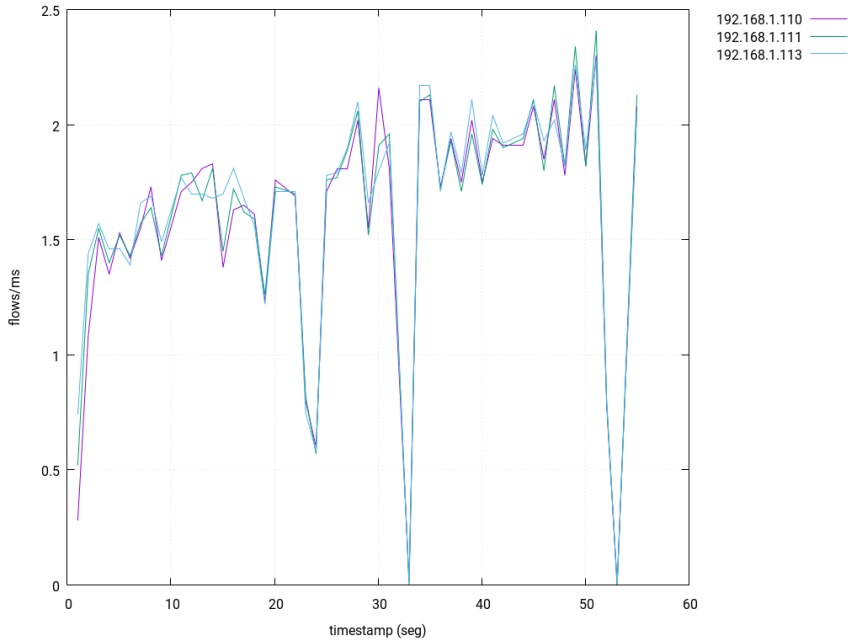

Figura 9.35 : Promedio de Flujos/ms por Nodo – Floodlight – N4

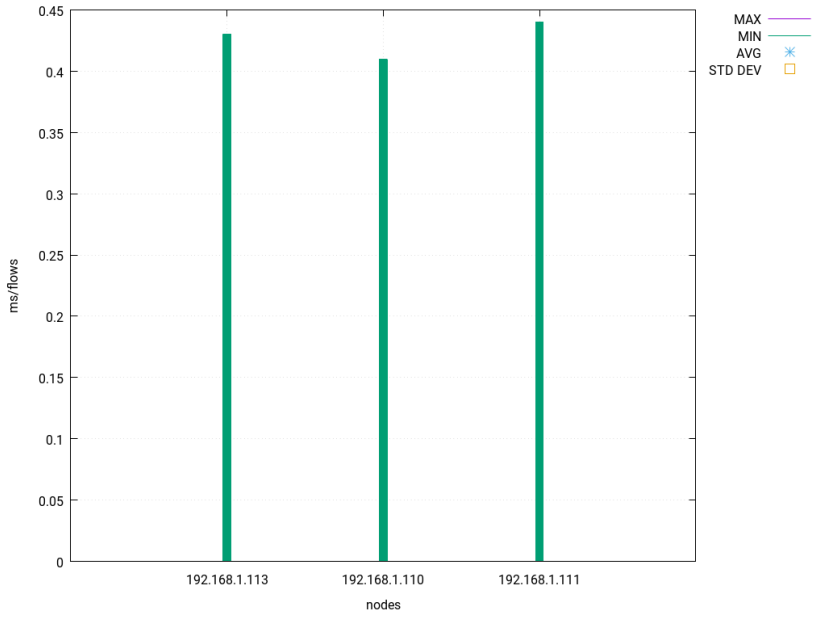

Figura 9.36: Máximo, Mínimo, Promedio y Desviación Flujos/ms – Floodlight – N4

Vale destacar, para la Figura 9.36, no fue graficado el máximo debido a que el mismo no fue definido durante la prueba, el máximo representaría el mayor tiempo de espera para respuesta a un flujo. Pero existen flujos sin respuesta durante la prueba, como se muestra en la Figura 9.35 , donde la cantidad de flojos por milisegundo llega cero en diferentes oportunidades, esto representa que el máximo tiempo de respuesta a un flujo fue indefinido, por lo que no se representa gráficamente.

9.3.2.3 Prueba con Controlador Ryu

El controlador Ryu se ejecutó con la aplicación de switch simple que incluye la instalación. Para eso se utilizó el comando mostrado en la Figura 9.37.

```
./bin/ryu-manager --verbose ryu/app/simple_switch_13.py
```

Figura 9.37: Comando para la Ejecución de Ryu

Para el controlador Ryu se realizaron pruebas análogas a las realizadas con los demás controladores, pero en este caso se recibieron valores de rendimiento bajos, esto posiblemente debido a la simplicidad de la aplicación de switch incluida por defecto. Los resultados se muestran en las Figuras desde la 9.38 hasta la 9.44.

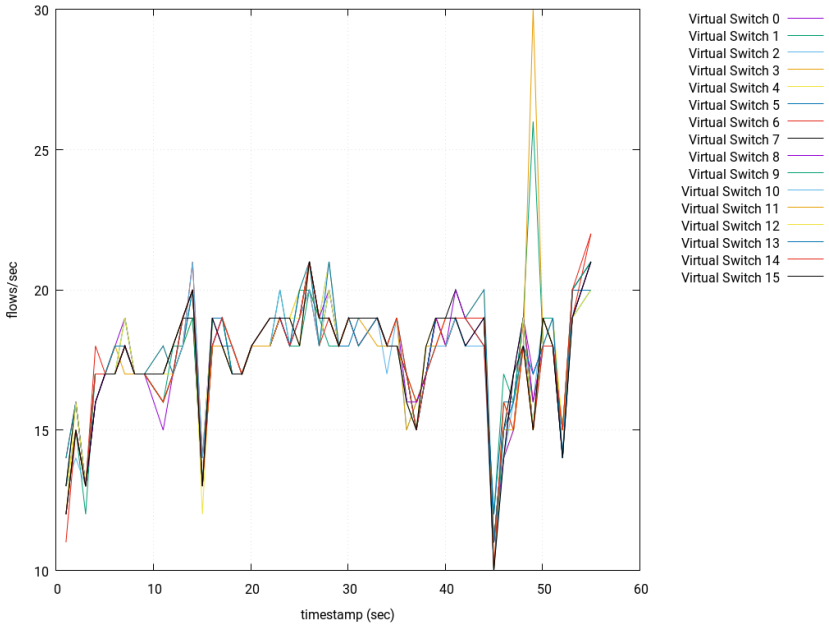

Figura 9.38: Flujos/seg por Switch Nodo 1 – Ryu – N4

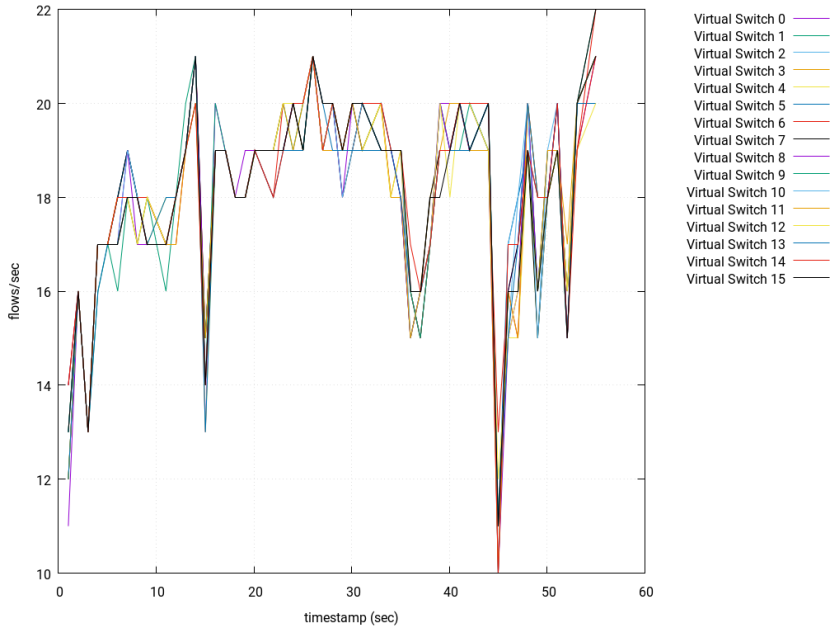

Figura 9.39: Flujos/seg por Switch Nodo 2 – Ryu – N4

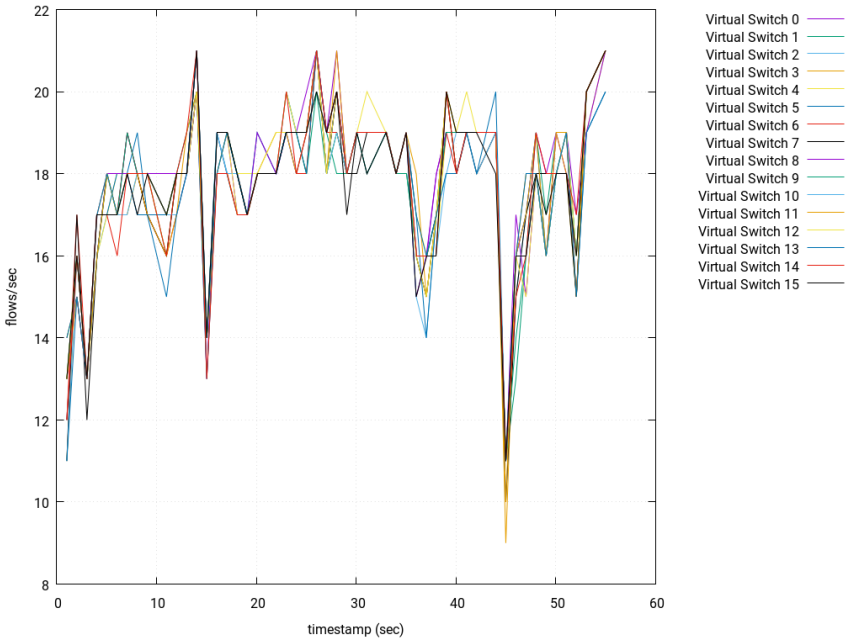

Figura 9.40: Flujos/seg por Switch Nodo 3 – Ryu – N4

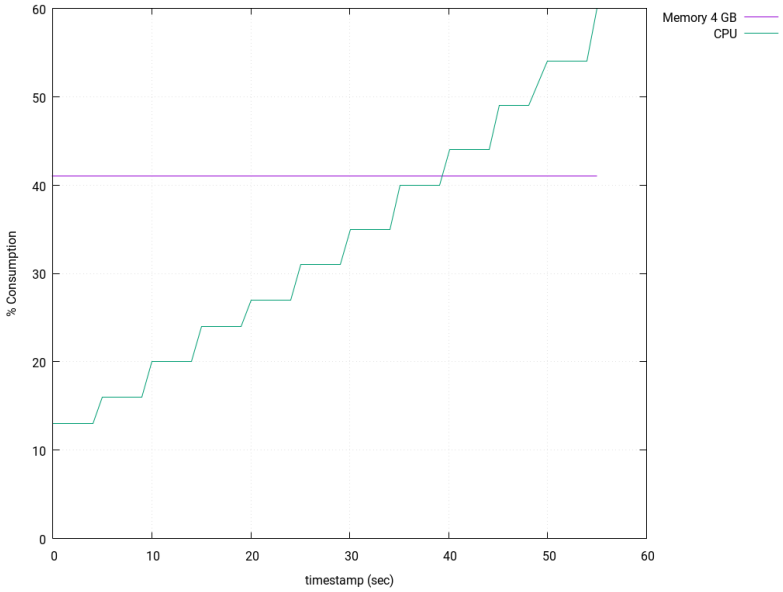

Figura 9.41: Consumo de Memoria y CPU – Ryu – N4

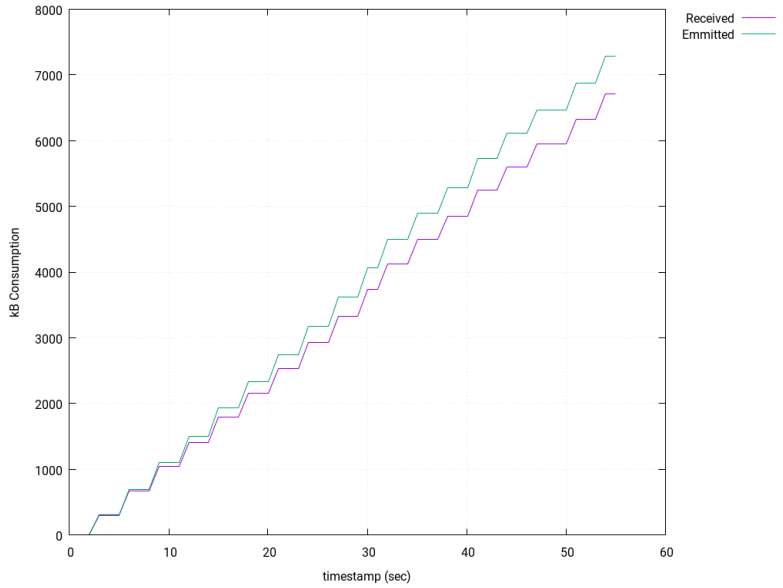

Figura 9.42: Consumo y Emisión de kB por Interfaces – Ryu – N4

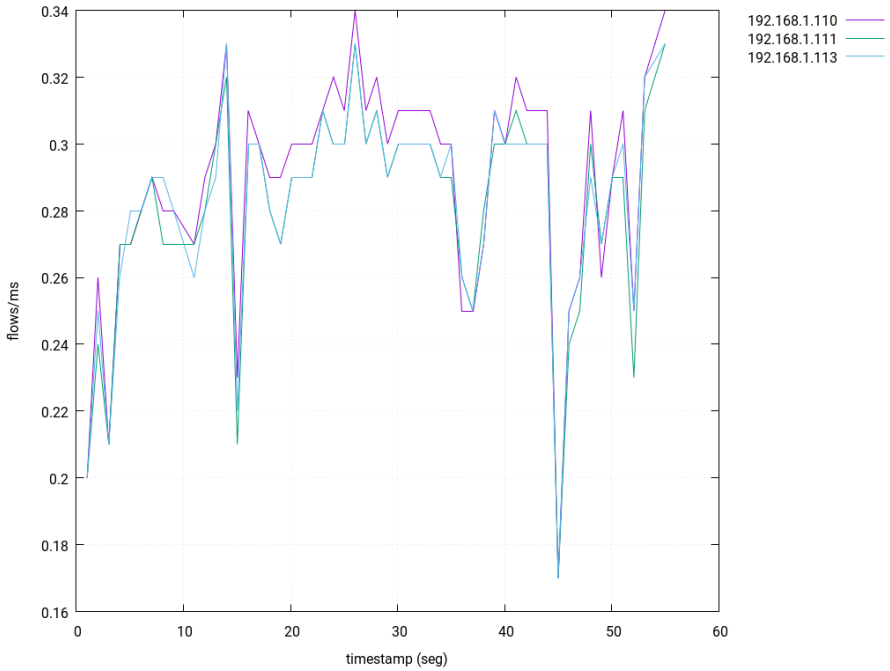

Figura 9.43: Promedio de Flujos/ms por Nodo – Ryu – N4

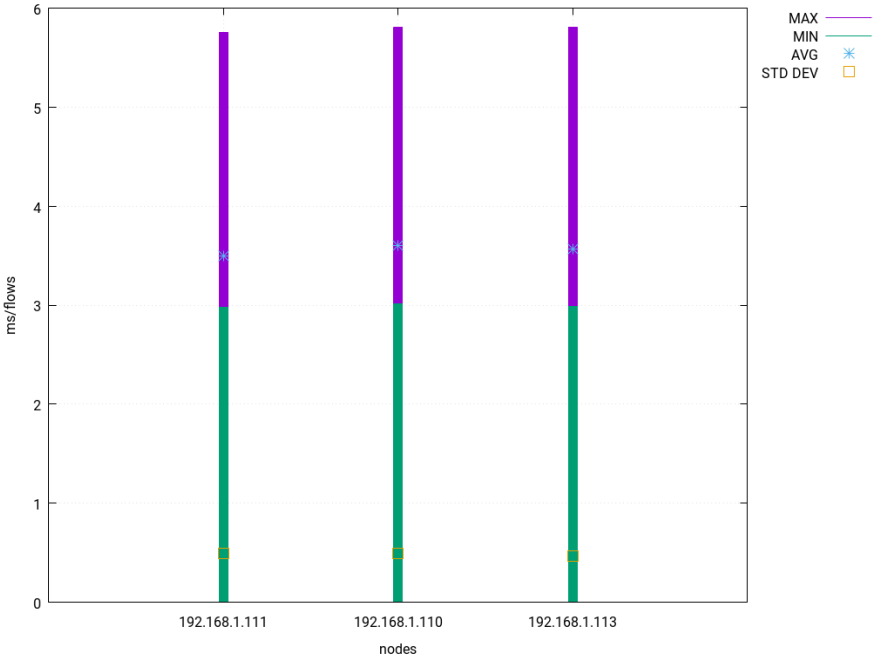

Figura 9.44: Máximo, Mínimo, Promedio y Desviación de Flujos/ms – Ryu – N4

9.3.2.4 Prueba con Controlador OpenDaylight

Para la prueba del controlador OpenDaylight con la versión 1.3 de OpenFlow, se usó se usó el comando mostrado en la Figura 9.45.

```
./run.sh -virt ovsdb -of13
```

Figura 9.45: Comando para la Ejecución de OpenDaylight

El controlador OpenDaylight tuvo problemas durante la prueba planteada, el mismo funcionaba bien durante las primeras iteraciones, pero luego, el controlador reducía la tasa de respuesta hasta quedar inhibido. Se repitió la prueba con una cantidad menor de ciclos con el fin de lograr un resultado positivo. Durante la siguiente prueba se utilizaron 16 ciclos y los resultados obtenidos en las misma se muestran en las Figuras desde la 9.46 hasta la 9.52.

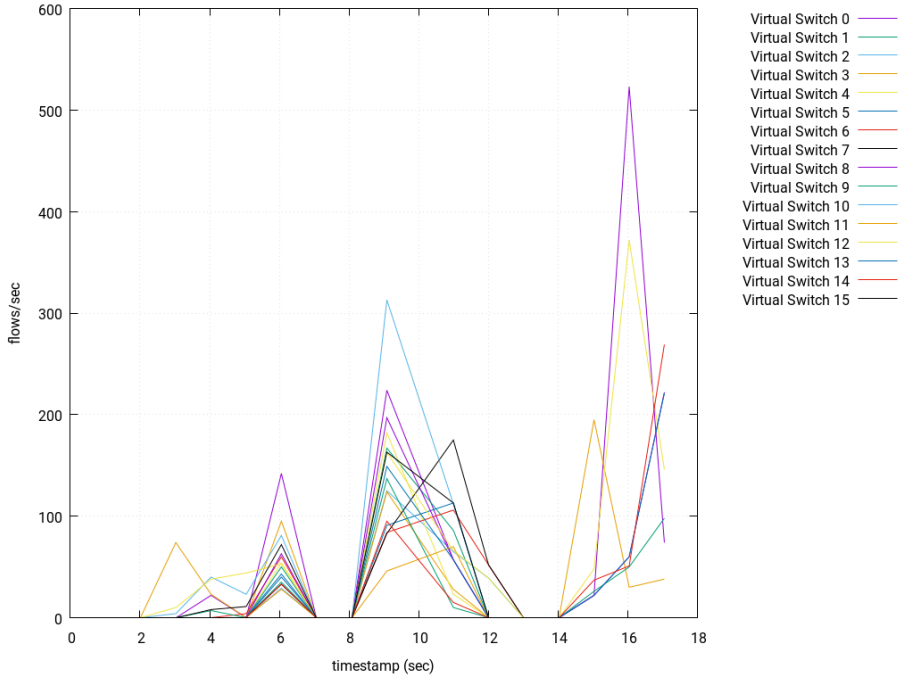

Figura 9.46: Flujos/seg por Switch – Nodo 1 – ODL – N4

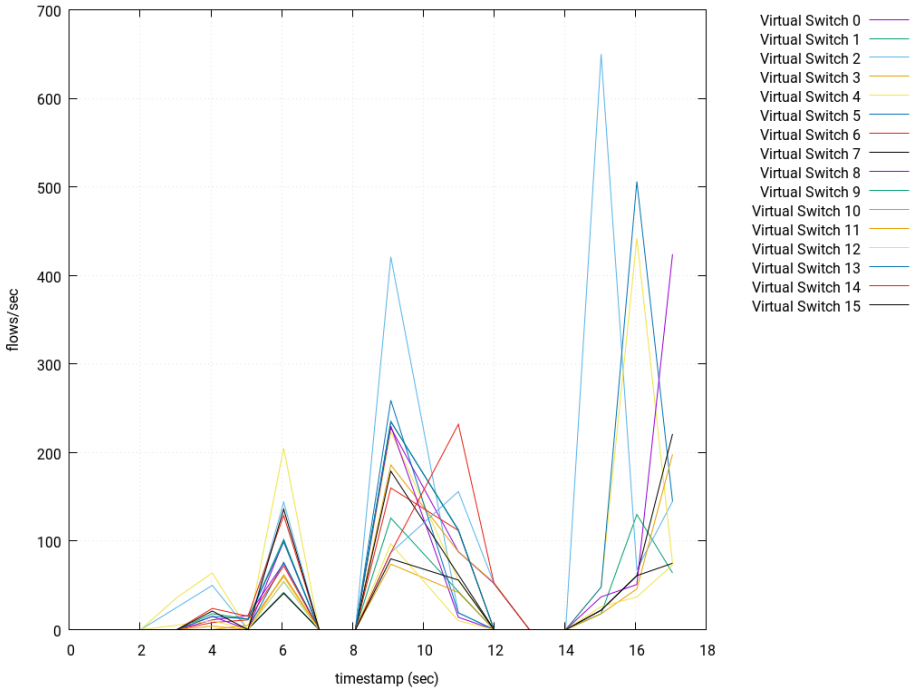

Figura 9.47: Flujos/seg por Switch – Nodo 2 – ODL – N4

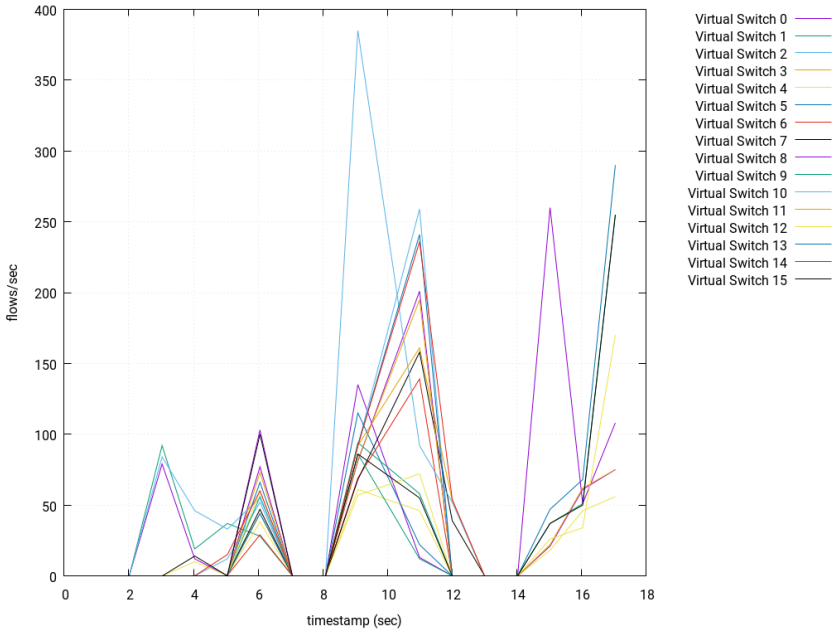

Figura 9.48: Flujos/seg por Switch – Nodo 3 – ODL – N4

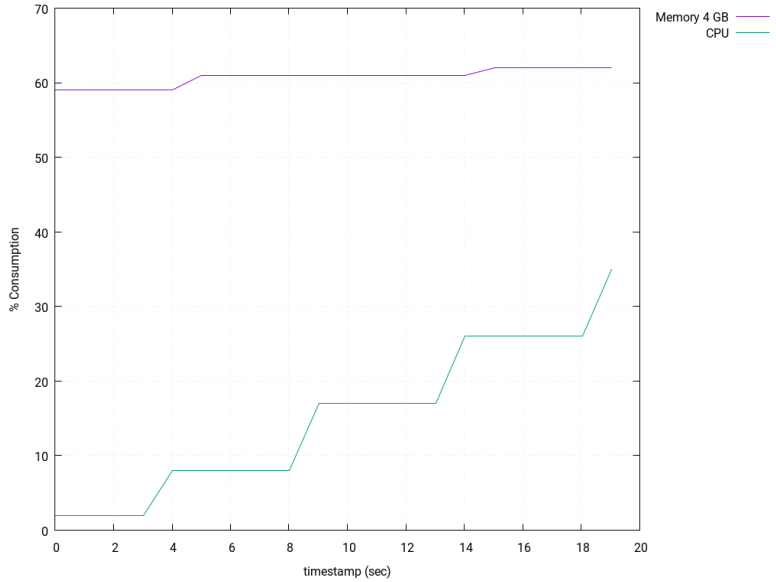

Figura 9.49: Consumo de Memoria y CPU – ODL – N4

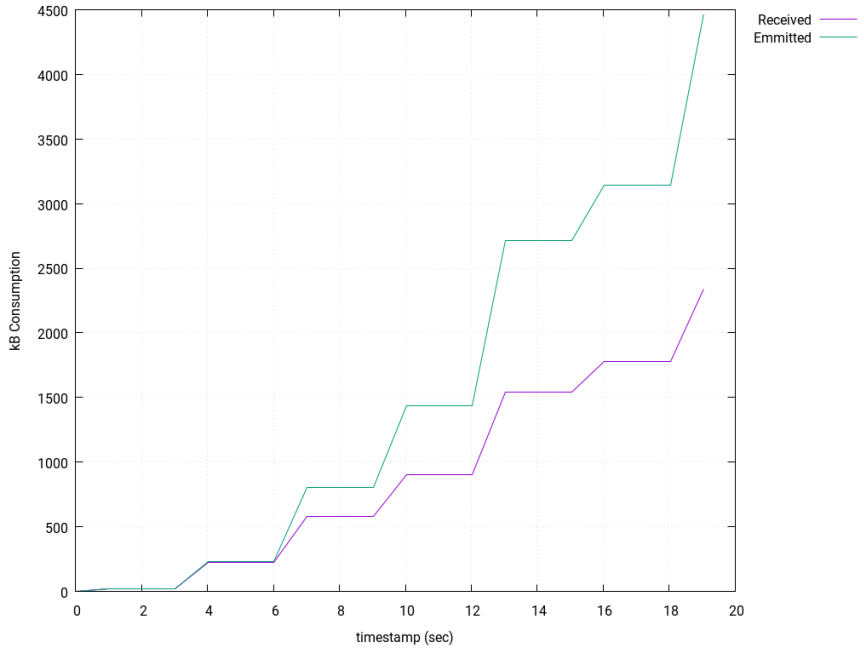

Figura 9.50: Consumo y Emisión de kB por Interfaces – ODL – N4

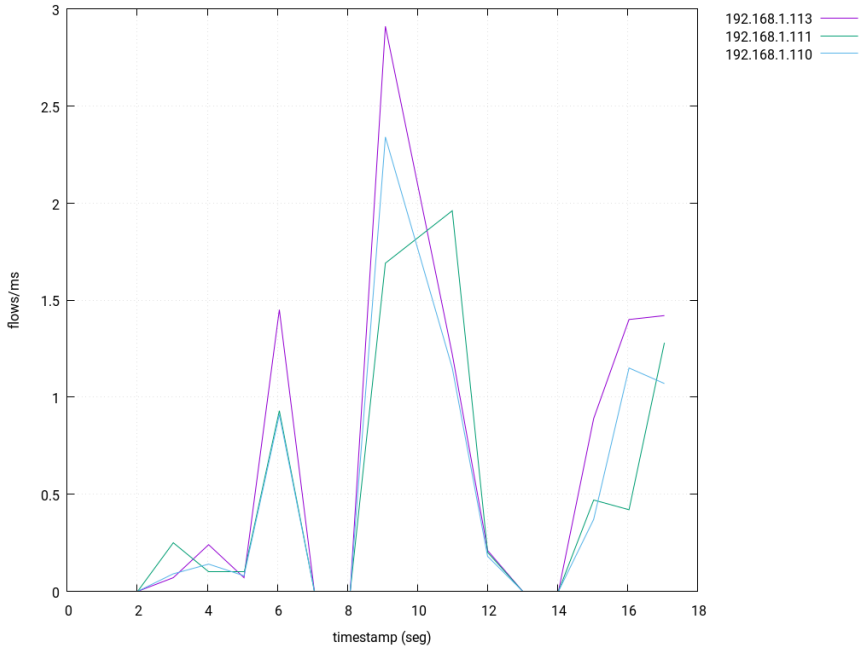

Figura 9.51: Promedio de Flujos/ms por Nodo – ODL – N4

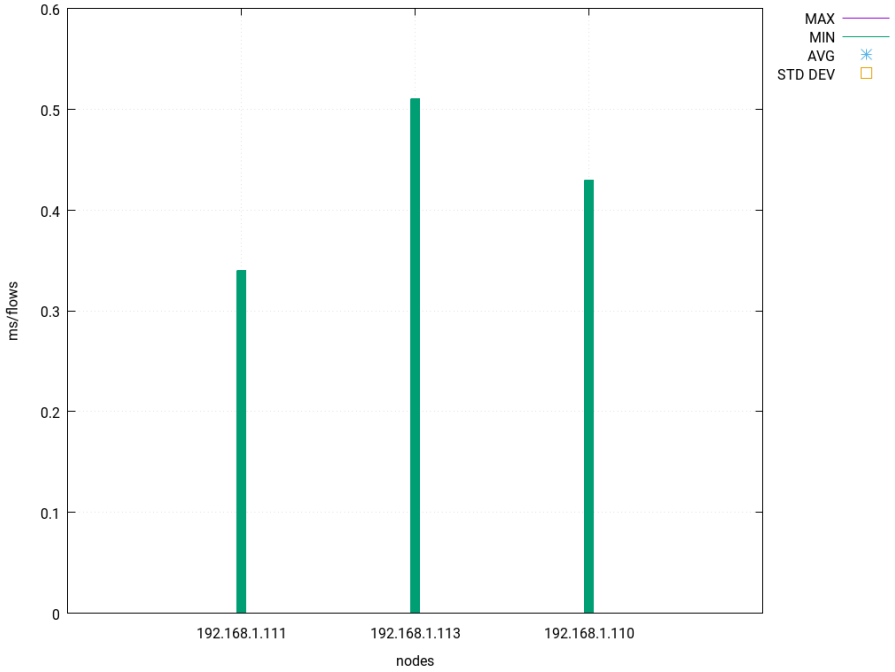

Figura 9.52: Máximo, Mínimo, Promedio y Desviación de Flujos/ms – ODL – N4

9.3.3 Prueba Comparativa

La realización de pruebas con herramientas similares fue realizada con dificultad, esto debido a la existencia de pocas herramientas afines que sean Open Source o al menos permitan la ejecución de pruebas con licencias gratuitas. La principal opción Open Source es Cbench, cuyo comando utilizado se muestra en la Figura 9.53. La prueba fue realizada sobre un controlador OpenMUL análogo al utilizado en la prueba mencionada en la Sección 9.3.1, el cual se usaría como objeto de comparación a los resultados obtenidos en la prueba.

```
mininet@mininet-vm:~$ cbench -c 192.168.1.6 -l 50 -p 6653
```

Figura 9.53: Comando Cbench para Prueba Comparativa

Es de suma importancia señalar que existe una diferencia significativa en ambas pruebas, que es la versión de OpenFlow utilizada. En la prueba a realizarse con Cbench se utiliza la versión 1.0, debido a que es la única versión soportada. Mientras que en la prueba realizada a diferentes controladores SDN por la herramienta desarrollada, se utiliza el protocolo en la versión 1.3, el cual tiene cambios significativos. Los resultados de la prueba realizada con la versión 1.0 se muestra en la Figura 9.54.

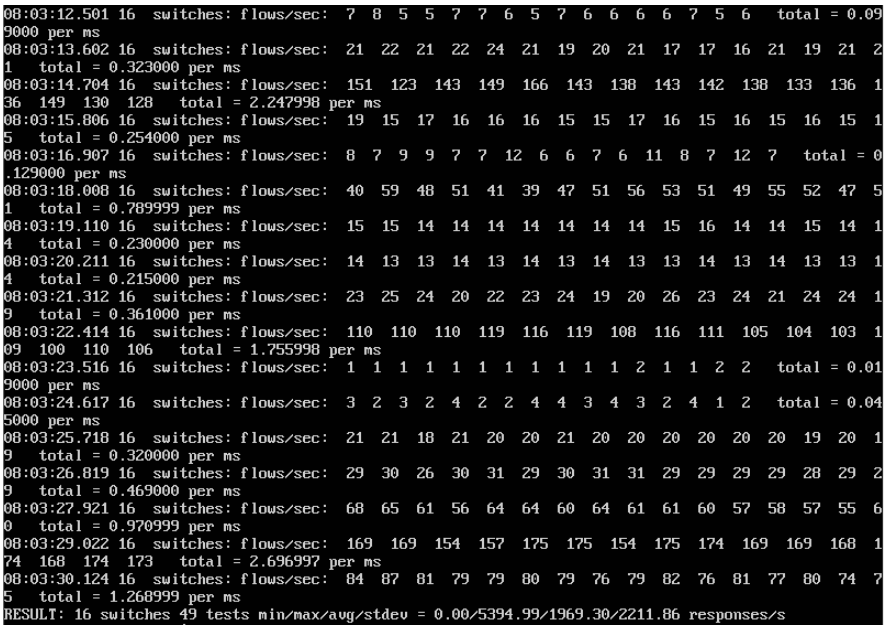

Figura 9.54: Resultados de Cbench Evaluando a OpenMUL

Se puede observar en los resultados, un comportamiento del controlador parecido al obtenido en la prueba de modo aislado, lo que muestra la correcta operación de la herramienta para evaluar controladores OpenFlow 1.3.

Para la comparativa directa con la misma versión del protocolo OpenFlow, se identificó una herramienta de evaluación con esta compatibilidad, llamada PKT-Blaster [45] de la empresa Veryx. Desafortunadamente no se logró obtener una licencia para la realización de pruebas.

10. Conclusiones

SDN y OpenFlow proponen un nuevo camino en la creación, manejo y automatización de redes.

El cambio de paradigma de SDN abrió nuevas puertas a los desarrolladores en el ámbito de redes de computadoras, lo que causó una alta aceleración en el desarrollo de versiones OpenFlow, y a su vez, una variedad de controladores.

La diversidad entre controladores dio pie a la exigencia de herramientas de evaluación de desempeño como Cbench. Sin embargo, pocas son de código libre, y aún menos, aquellas con el mantenimiento adecuado. Si se suma esto a la existencia de controladores SDN de alto rendimiento, el hardware limitado que puede soportar una sola herramienta de benchmarking, así como la falta de pruebas que consideren indicadores de uso del hardware del controlador (memoria y CPU), surge una oportunidad para el desarrollo de OFC-Benchmark, una herramienta de evaluación de desempeño con OpenFlow 1.3.

OFC-Benchmark se desarrolló con el lenguaje de programación C, y fue complementada con librerías que facilitaron el manejo y construcción de paquetes para los de protocolos de comunicación SNMP y OpenFlow 1.3. Otras librerías facilitaron la abstracción en el desarrollo, cómo lo permitió *time.h* en la manipulación del tiempo.

OFC-Benchmark terminó siendo una propuesta poderosa e interesante para realizar pruebas de fácil comprensión, adecuada a controladores más recientes. Es ídeal para apoyar la toma de decisión sobre el análisis de desempeño en controladores.

El presente trabajo logró cumplir con los objetivos generales planteados y reconoce e impulsa el desarrollo de nuevas funcionalidades y áreas que tomen en cuenta:

- Inclusión de nuevas versiones de los protocolos OpenFlow y SNMP.
- Ofrecimiento de compatibilidad hacia atrás con OpenFlow 1.3.
- Incremento del rango de opciones de configuración ofrecidas.
- Implementación de temporizadores para la sincronización de mensajes.
- Implementación del manejo de errores basado en logs.
- Control de seguridad sobre la recepción de información no deseada o suplantación de identidad.
- Implementación de buffers dinámicos e incrementales.
- Integración de interfaces gráficas que mejoren la usabilidad y experiencia de usuario.
- Posibilidad de realizar pruebas a redes más complejas con controladores bajo arquitecturas de balanceo de cargas.

Referencias

[1] M. Jarschel, F. Lehrieder, Z. Magyari y R. Pries, “A Flexible OpenFlow-Controller Benchmark”. Octubre 2012.

[2] N. Feamster, J. Rexford y E. Zegura, “The Road to SDN: An Intellectual History”. *ACM SIGCOMM Computer Communication Review,* vol. 44, no. 2, pp. 87-98, Abril 2014.

[3] N. Figuerola, “SDN – Redes Definidas por Software”. Octubre 2013. https:// articulosit.files.wordpress.com/2013/10/sdn.pdf.

[4] I. Gavilán, “Fundamentos de SDN (Software Defined Networking)”. Agosto 2013. http://es.slideshare.net/igrgavilan/20130805-introduccion-sdn.

[5] W. Stallings, Software Defined Networks and OpenFlow, vol. 16, Internet and Intranet Professionals, Marzo 2013.

[6] A. Shalimov y R. Smeliansky, “On Bringing Software Engineering to Computer Networks with Software Defined Networking”. *Proceesdings of the Spring/Summer Young Researchers´ Colloquium on Software Engineering,* no. 7, 2013.

[7] P. Ivashchenko, A. Shalimov y R. Smeliansky, “High Performance Inkernel SDN/OpenFlow Controller”. Enero 2014. https://www. usenix.org/ sites/default/files/ons2014-poster-ivashchenko.pdf.

[8] S. Yeganeh, A. Tootonchian y Y. Ganjali, “On Scalability of Software Defined Networking”. *IEEE Communications Magazine,* vol. 51, no. 2, pp. 136-141, Febrero 2013.

[9] Open Networking Foundation, “Software-Defined Networking: The New Norm of Networks”. Abril 2012. https://www.opennetworking.org/images /stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[10] T. D. Nadeau y K. Gray, “SDN: Software Defined Networks”. Sebastopol: O'Reilly, Agosto 2013.

[11] Open Networking Foundation, “OpenFlow Switch Specification Version 1.5.1”. Marzo 2015. https://www.opennetworking.org/images/stories/ downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf.

[12] N. McKeown, T. Anderson, L. Peterson, J. Rexford, J. Turner, S. Shenker, G. Parulkar y H. Balakrishnan, “OpenFlow: Enabling Innovation in Campus Networks”. Marzo 2008.

[13] S. Azodolmolky, "Software Defined Networking with OpenFlow", Packt Publishing Ltd., Octubre 2013.

[14] Open Networking Foundation, “OpenFlow Switch Specification Version 1.3.2”. Abril 2013. https://www.opennetworking.org/images/stories/ downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.2.pdf.

[15] Flowgrammable, “Message Layer”. 2016. http://flowgrammable.org/sdn/ openflow/message-layer.

[16] T. Dierks y E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2”. RFC 5246. Agosto 2008.

[17] P. Göransson y C. Black, “Software Defined Networks: A Comprehensive Approach”, Waltham: Morgan Kaufmann, Mayo 2014.

[18] Flowgrammable, “FeatureReq - FeatureRes”. 2016. http://flowgrammable .org/sdn/openflow/message-layer/feature/#FeatureRes_1.3.

[19] Flowgrammable, “GetConfigReq - GetConfigRes - SetConfig”. 2016. http:// flowgrammable.org/sdn/openflow/message-layer/config/#Config_1.3.

[20] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown y S. Shenker, “Ethane: Taking Control of the Enterprise”. Octubre 2007.

[21] P. Mockapetris, “Domain Names - Concepts and Facilities”. RFC 1034. Noviembre 1987.

[22] R. Droms, “Dynamic Host Configuration Protocol”. RFC 2131. Marzo 1997.

[23] M. Palacin Mateo, “OpenFlow Switching Performance”. Julio 2009.

[24] G. Watson, N. McKeown y M. Casado, “NetFPGA: A Tool for Network Research and Education”. Febrero 2006.

[25] M. Bjorklund, “YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)”. RFC 6020. Septiembre 2010.

[26] A. García Centeno, C. M. Rodríguez Vergel, C. Anías Calderón y F. C. Casmartiño Bondarenko, “Controladores SDN, Elementos para su Selección y Evaluación”. *Revista Telem@tica,* vol. 13, no. 3, pp. 10-20, 2014.

[27] OpenDaylight, “Wiki OpenDaylight”. OpenDaylight, https://wiki.OpenDaylight .org/view/OpenDaylight_Controller:Architectural_Framework.

[28] B. Guo, “Lightness”. Junio 2014. http://www.ict-lightness.eu/wp-content/ uploads/2014/06/lightness-D4.3_final.pdf.

[29] K. Ohmura, “OpenStack/Quantum SDN-based Network Virtulization”. Mayo 2013. http://osrg.github.io/ryu/slides/LinuxConJapan2013.pdf.

[30] T. Fujita, “Introduction to Ryu SDN Framework”. Abril 2013. http://osrg .github. io/ryu/slides/ONS2013-april-ryu-intro.pdf.

[31] M. Bredel, “Admin Magazine: Network & Security”. Marzo 2015. http://www. admin-magazine.com/Articles/Floodlight-Welcome-to-the-World-of-Software-Defined-Networking.

[32] N. Malik y D. Saikia, “An Introduction to OpenMUL SDN Suite”. Septiembre 2014.

[33] L. Bell, A. Smith, P. Langille, A. Rijhsinghani y K. McCloghrie, “Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering and Virtual LAN Extensions”. Agosto 1999.

[34] E. Kjeld Borch y F. Paul, “The IP Network Address Translator”. RFC 1631. Mayo 1994.

[35] P. Sim, “Virtualized Network with OpenvSwitch”. Noviembre 2013. http://www. slideshare. net/janghoonsim/virtualized-network-with-openv-switch.

[36] B. Lantz, N. Handigol, H. Brandon y V. Jeyakumar , “Introduction to Mininet”. Diciembre 2015. https://github.com/mininet/mininet/wiki/Introduction-to-Mininet.

[37] L. M. Garcia, “Programming with Libpcap”. *Hacking,* vol. 3, no. 2, 2008.

[38] A. López Monge, “Aprendiendo a Programar con Libpcap”. Febrero 2005.

[39] J. Case, M. Fedor, M. Schoffstall y J. Davin, "IETF", A Simple Network Management Protocol (SNMP), Mayo 1990: https://tools.ietf.org/html/rfc1157.

[40] P. K. Janert, Gnuplot in Action, Second Edition, Marzo: MNNING, 2016 .

[39] A. Hassan y S. Ahmed, “Performance Comparison of the State of the Art Openflow Controllers”. Halmstad, Septiembre 2014.

[40] L. Anderson y T. Madsen, “Provider Provisioned Virtual Private Network (VPN) Terminology”. RFC 4026. Marzo 2005

[41] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado y R. Sherwood, “On Controller Performance in Software Defined Networks”. USENIX, 2012

[44] INTECO, "METODOLOGÍAS Y CICLOS", INTECO, España, 2009.

[45] "Veryx": http://www.veryxtech.com/products/pktblaster-sdn-software-defined-network-test/.