ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN PROTOTIPO PARA OFRECER EL SERVICIO DE DHCP SOBRE UNA SDN

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

MANUEL ALEJANDRO VELASCO BERREZUETA
manuel.velasco.b@gmail.com

DIRECTOR: ING. RAÚL DAVID MEJÍA NAVARRETE, MSc.
david.mejia@epn.edu.ec

Quito, Febrero 2016

DECLARACIÓN

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

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

_____

Manuel Alejandro Velasco Berrezueta

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Manuel Alejandro Velasco Berrezueta, bajo mi supervisión.

Ing. David Mejía, MSc.
DIRECTOR DE PROYECTO

AGRADECIMIENTOS

Un infinito agradecimiento para mis padres, en especial para mi madre María Luisa, quienes con dedicación y esfuerzo siempre me han inculcado la disciplina y perseverancia que día a día me impulsan a la superación en cada uno de los ámbitos de la vida.

Igualmente deseo agradecer a mi familia, mis hermanos, primos y amistades que de cierto modo pasaron a ser parte de la familia por hacer este viaje más ameno y enriquecedor con cada una de sus enseñanzas que siempre llevo como bandera de mis acciones, y tengan presente que cada momento vivido sea positivo o negativo lo atesoro con gran afecto.

Un agradecimiento cordial a la Escuela Politécnica Nacional por sembrar en mí la pasión por la ciencia, a cada uno de los profesores que despertaron en mí la curiosidad por el aprendizaje a través de los conocimientos compartidos y de manera especial un agradecimiento a mi tutor el Master David Mejía por la orientación brindada en todo el desarrollo de este proyecto.

DEDICATORIA

Para mi padre Luis Antonio Velasco, que si bien te adelantaste en la fila hacia la gloria eterna, cada día te siento más presente y tus palabras cobran más vida convirtiéndose en mi impulso para seguir adelante.

CONTENIDO

ÍNDICE DE FIGURAS

CAPÍTULO I

CAPÍTULO II

CAPÍTULO III

ÍNDICE DE TABLAS

CAPÍTULO I

CAPÍTULO II

CAPÍTULO III

ÍNDICE DE CÓDIGOS

CAPÍTULO II

CAPÍTULO III

PRESENTACIÓN

El acelerado cambio de las redes actuales, tanto en la ubicación nómada de sus clientes como en la naturaleza de los servicios que solicitan en cuanto a diversidad de formatos y orígenes de contenidos, han logrado que la infraestructura tradicional de red no pueda abastecer satisfactoriamente a todos los solicitantes de un servicio, desembocando en usuarios insatisfechos y en potenciales pérdidas económicas en las empresas proveedoras de servicios.

Como respuesta a esta situación se ha optado por dejar de “parchar” a las redes tradicionales y más bien redefinir los conceptos del paradigma de red utilizado, es aquí donde, entre otras tecnologías, nacen las Redes Definidas por Software.

Si bien en sus inicios se trató de una solución solamente desplegada en ambientes educativos y con fines de investigación, ha llamado la atención de grandes fabricantes al responder eficientemente a los retos que presentan las redes actuales y mediante la compatibilidad de sus nuevos equipos *(hardware)* con esta tecnología de red se ha logrado difundir de manera el uso del paradigma de las SDN a nivel comercial proyectándola como la tecnología de próxima generación y la tendencia de las grandes marcas a futuro.

La solución desarrollada en el presente proyecto pretende atender un servicio básico dentro de una red que es la asignación de parámetros de red mediante DHCP bajo el paradigma de las SDN.

RESUMEN

El presente Proyecto de Titulación se encuentra divido en cuatro capítulos: en el primer capítulo se realiza una revisión del fundamento teórico de las SDN y del protocolo DHCP, el segundo capítulo corresponde al desarrollo de la aplicación prototipo de una red SDN que brinda el servicio de DHCP, el tercer capítulo trata sobre las pruebas de funcionamiento de la aplicación prototipo, y en el cuarto capítulo se presentan las conclusiones y recomendaciones obtenidas durante la realización de este Proyecto.

En el fundamento teórico se hace un breve análisis de las redes actuales y sus limitaciones, posteriormente se propone y desarrolla la utilización de una red SDN para superar las dificultades de las redes actuales. Se presenta una explicación de los controladores dentro de la arquitectura SDN, y adicionalmente se presentan las características del protocolo OpenFlow utilizado en el paradigma de redes SDN. Además en este capítulo se resume el protocolo DHCP a partir de la documentación del mismo y los aspectos de funcionamiento requeridos para el desarrollo de la aplicación prototipo.

Para el desarrollo de la aplicación se entra en mayor detalle con respecto al entorno donde se desarrollará la aplicación, desde los elementos que constituyen la red donde se realizan las pruebas iniciales hasta el software utilizado para la codificación de la aplicación prototipo. Posteriormente se detalla todo el proceso de desarrollo de la aplicación desde la etapa de modelado de las tareas a cumplirse hasta la obtención del *script* en Python que cumple con las tareas de asignación de parámetros de red a través del protocolo DHCP.

En la fase de pruebas de funcionamiento del prototipo se contemplan los dos escenarios donde se constata el funcionamiento de la aplicación, un primer escenario de tipo software con elementos virtuales y un segundo escenario con elementos de hardware. Se realiza adicionalmente la puesta en marcha de un servidor DHCP

tradicional para observar su comportamiento y capacidades de configuración en contraste con la aplicación prototipo desarrollada. Finalmente la comparación entre el servidor tradicional y la aplicación desarrollada se concentra en el parámetro del tiempo necesario para obtener exitosamente la configuración de red solicitada por parte de un cliente.

En el cuarto capítulo se presentan las conclusiones y las recomendaciones del presente Proyecto de Titulación.

Finalmente se incluyen los Anexos correspondientes a los tutoriales elaborados durante la instalación y configuración de todos los elementos de software y hardware empleados en el Proyecto.

Se adjunta el CD correspondiente, donde se encuentra el código de la aplicación desarrollada.

CAPÍTULO I

1. MARCO TEÓRICO

En este capítulo se dará una breve introducción a los conceptos básicos de las SDN (*Software Defined Networks*), se explicará la función de cada componente de su arquitectura y principalmente se enfocará al análisis de las características que ofrece para dar soluciones en cuanto a la funcionalidad requerida en las redes actuales.

Se explica la utilidad de Mininet como herramienta de simulación de una red SDN para el desarrollo de aplicaciones que operen bajo este paradigma, y se enfoca el estudio en las ventajas que brinda Mininet como ambiente de pruebas.

Asimismo se expone los componentes del protocolo DHCP, los tipos de mensajes que emplea y los campos del mensaje DHCP para un mejor entendimiento de la aplicación a desarrollarse.

1.1. INTRODUCCIÓN

1.1.1. CARACTERÍSTICAS DE LAS REDES ACTUALES

Hoy en día las redes informáticas son muy diferentes en todo sentido a las redes que se tenía cuando apenas emergía el concepto de Internet o se experimentaba con redes inalámbricas, desde el hecho que en la actualidad cada dispositivo electrónico tiene la posibilidad de conectarse a una red hasta la presencia de una infinidad de servicios y por lo tanto tipos de tráfico que circulan por las redes.

Tanto los protocolos como los equipos que atendían las necesidades de ese entonces ahora se ven limitados para abastecer la demanda actual de servicios y aplicaciones hasta el punto en el cual dichos equipos son sustituidos por nuevos dispositivos que para su implementación e instalación requieren readecuar los distintos nodos de una red para permitir su funcionamiento adecuado, incurriendo así en costos adicionales y tiempos en los cuales hay cortes de servicio. Asimismo los conceptos de virtualización de servidores e interconexión de centros de datos son ejemplo de cómo el volumen de información que circula por la red se ha

incrementado enormemente y dicho tráfico sigue siendo manejado por equipos *legacy*[1] dimensionados para distintos propósitos y características de tráfico.

1.1.2. ALTERNATIVAS PARA SOLUCIONAR LAS EXIGENCIAS DE LAS REDES ACTUALES

Como respuesta a estos inconvenientes y algunos más que han surgido con la evolución de las redes es evidente que es necesario un paradigma de redes distinto al tradicional que se adapte a las características de tráfico actuales.

La alternativa de las SDN nace como respuesta a dichas necesidades, ha surgido para tratar de manera más eficiente al tráfico presente en las redes actuales.

El objetivo de las SDN es conceder una administración de tipo abierta y controlada por el usuario (que tenga privilegios) para personalizar el direccionamiento del tráfico a nivel hardware. Esto lo logra a través de la centralización de la inteligencia del plano de control, separando al plano de datos. Esta característica permite mantener los equipos *legacy* para manejar tareas referentes al plano de datos y se le cede la inteligencia, es decir la capacidad de personalización y decisión, del plano de control a un ente centralizado denominado controlador, con lo cual se contribuye a la flexibilidad de toda la red desde este elemento centralizado [1].

Los campos de desarrollo para las Redes Definidas por Software pueden ser tanto académicos como tecnológicos a nivel de industria. A nivel académico brinda la libertad de programar y así modificar a un *switch* con libertad; mientras que en la industria tecnológica, SDN se acopla perfectamente a las tendencias de virtualización y computación en la nube al manejar el tráfico que se genera en estos ambientes de manera más eficiente.

Las SDN empiezan a captar la atención de fabricantes al presentar una solución a problemas como por ejemplo la escalabilidad, la virtualización de red y la automatización de tareas que son prioritarias en una red actual.

---

[1] Un sistema *legacy*, en el contexto de la informática, se refiere a un sistema obsoleto de computador o software que se utiliza en lugar de versiones mejoradas disponibles.

1.1.3. ARQUITECTURA DE UNA SDN

En la Figura 1.1 se muestra la arquitectura general de una SDN con sus respectivas interfaces y componentes:

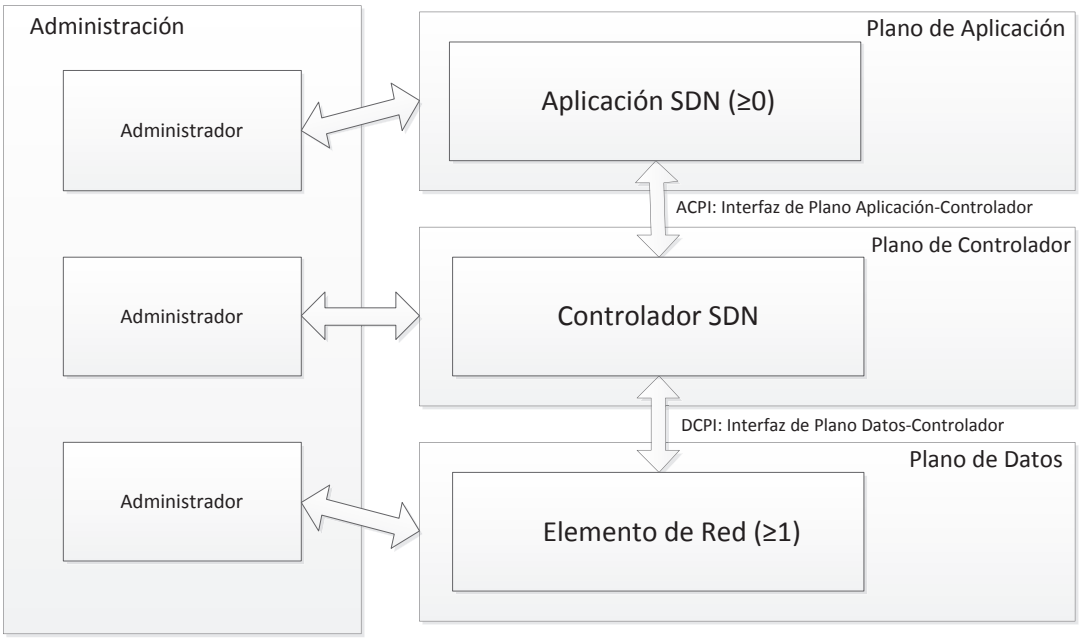

Figura 0.1. Arquitectura SDN en el plano de datos [2]

En el plano de datos se encuentran los NE[2] (*network element*) que contienen una serie de recursos para conmutación y procesamiento.

Para el plano de control SDN se tiene justamente el controlador o varios controladores, cada uno de ellos con control exclusivo de un conjunto de recursos asociados a uno o varios NE que residen en el plano de datos. Este tipo de recursos en el caso de múltiples controladores generalmente se basan en políticas del mejor esfuerzo o en colas de tipo FIFO[3].

Las funciones y características del controlador se detallan más adelante, pero como mínimo un controlador debe ser capaz de ejecutar las peticiones de las aplicaciones que residen en el de manera aislada con respecto a las demás.

Otra función que no es esencial pero sí es muy común para un controlador es la de actuar como un ente de control para posibles fallos o modificaciones en la red, una re

---

[2] NE (Elemento de Red) término común para el elemento que manipula, almacena o direcciona datos de usuario.

[3] FIFO (F*irst-In-First-Out*) para este caso se traduce como “llega primero, es atendido primero”.

optimización de recursos de red e incluso recuperación hacia estados previos de la SDN en la que se encuentra [2].

Finalmente en el plano de aplicación justamente se hallan las aplicaciones, pues pueden ser una o varias, a las cuales se les concede un control exclusivo de un conjunto de recursos indicados por uno o varios controladores SDN.

Si bien sus tareas son aisladas puede existir colaboración entre aplicaciones en caso de ser requerido y una aplicación puede actuar como controlador SDN por sus propias capacidades.

1.2. EL CONTROLADOR SDN

El servidor controlador es una entidad de software de la arquitectura SDN, separada de los dispositivos de red, que maneja las tareas correspondientes al plano de control, de ahí su nombre.

El concepto de controlador no se limita a un sistema monolítico encargado de dichas tareas, sino se extiende a la posibilidad de varios componentes funcionales acomodados para trabajar de manera colaborativa, que para fines de simplificar su estructura suelen ser considerados como un sistema de caja negra[4] al cual solamente se lo define a partir de su comportamiento funcional externo [3].

Un servidor controlador puede ser ejecutado en diversas plataformas computacionales, asimismo puede ser ejecutado sobre recursos listos para ser migrados como una máquina virtual.

El control lógico centralizado pretende que el controlador SDN tenga un alcance global y cada uno de sus módulos internos puedan compartir información de la red, de tal modo que no sea preocupación externa, interpretar comandos conflictivos o contradictorios que se generen en el controlador, pues éste asume todas las tareas de gestionar las modificaciones o fallos en la red. Previamente se mencionaba que el controlador trabaja como un todo, pues puede tratarse de un solo componente lógico

[4] En informática una caja negra indica que no se detalla la composición interna de un elemento, ya que solamente interesa el resultados que proporciona

o un arreglo de varios controladores trabajando conjuntamente y que componen una sola entidad que actúa como un solo controlador; dichos componentes del controlador podrán tener acceso compartido a los recursos de red, si cumplen estos parámetros:

- Se configuran para controlar conjuntos de recursos desvinculados.
- Se sincronizan entre sí para no causar inconvenientes de inconsistencia o comandos conflictivos.

En resumen un controlador SDN contiene un conjunto de "módulos¨ que se pueden unir para ejecutar tareas de red, tales como la recopilación de estadísticas y análisis del estado de la red, entre otros.

1.2.1. HISTORIA DE LOS CONTROLADORES SDN

El primer controlador SDN fue NOX, desarrollado por Nicira Networks en paralelo con OpenFlow, protocolo que será explicado más adelante. Para el año 2008 NOX fue donado a la comunidad SDN como un proyecto de tipo *Open Source*[5], constituyendo la base para distintos controladores SDN que aparecieron posteriormente. Todos estos proyectos de controladores SDN dependen fuertemente en la comunidad de desarrolladores involucrados, pues la mayoría de ellos son de tipo *Open Source.*

Beacon aparece en el año 2010, es un controlador basado en Java que también tiene una licencia GPL v2 de tipo software libre en conjunto con una licencia de la Universidad de Stanford. Existen muchos más controladores disponibles, algunos de ellos son Trema, basado en Ruby y Ryu que está apoyado por NTT y está basado en Python, y asimismo Floodlight que se deriva de Beacon.

Algunos vendedores de equipos como HP, Cisco, IBM y Juniper tienen sus propias ofertas SDN lanzadas al mercado. Basados en Beacon se lanzaron los controladores originales para dichas marcas, pero ahora todas ellas se han inclinado por el uso de OpenDaylight. El controlador OpenDaylight, que es parte de Linux Foundation, fue anunciado en 2013. Está basado en Java, soporta OpenFlow e incluso protocolos

---

[5] *Open Source*: programas cuyo código fuente es de libre acceso.

propietarios como Cisco OpFlex[6] y soporta *clustering*[7] y alta disponibilidad, se lo considera un controlador muy robusto por estas características [2].

1.2.2. FLOODLIGHT

Es un controlador que opera con OpenFlow, completamente abierto y gratuito que esta licenciado bajo Apache y se deriva del proyecto Beacon. Al ser desarrollado en Java tiene las comodidades de compilación y ejecución propias de dicho lenguaje.

Floodlight ha sido probado en ambientes de producción operando con *switches* físicos que soportan OpenFlow con buenos resultados. Presenta ventajas por la existencia de Event Listeners (escucha de eventos) que permiten capturar y manejar eventos de OpenFlow de una manera sencilla, asimismo posee una API[8] extensible de tipo REST[9] como se indica en la Figura 1.2.

Floodlight está conformado por los siguientes elementos:

- Topología: hace un seguimiento de los enlaces entre *hosts*[10] y *switches*.
- Administrador de dispositivos: hace un seguimiento de los dispositivos en la red (por ejemplo direcciones MAC, direcciones IP).
- Almacenamiento: es una capa de abstracción para el almacenamiento propio del controlador mediante el uso de localidades de memoria.
- *Counter Store*: estadísticas de OpenFlow y Floodlight.
- Ruteo y conmutación: el motor central para almacenar calcular rutas e instalar flujos.
- Web: para el API tipo REST.
- *Switch* de aprendizaje: una aplicación ejemplo de un *switch*, con funciones de conmutación que pueden ser usadas en la práctica.

---

[6] Cisco OpFlex: protocolo de estándar abierto orientado a Infraestructura de Aplicación Centralizada.
[7] *Clustering* se refiere al enlace de varios sistemas que actúen como uno solo de mejores prestaciones.
[8] API (*Application Programmin Interface*) es una capa de abstracción, con métodos, funciones o subrutinas ofrecidos por una biblioteca, utilizada por un software
[9] REST (*REpresentational State Transfer*) interfaz para servicios web simple que disminuye la carga al no requerir excesivos intercambios de mensajes
[10] El término *host* se utiliza para describir equipos finales de usuario como por ejemplo una PC

- *Hub*: al igual que la anterior aplicación puede ser usada en la práctica, realiza la función de repetidor.

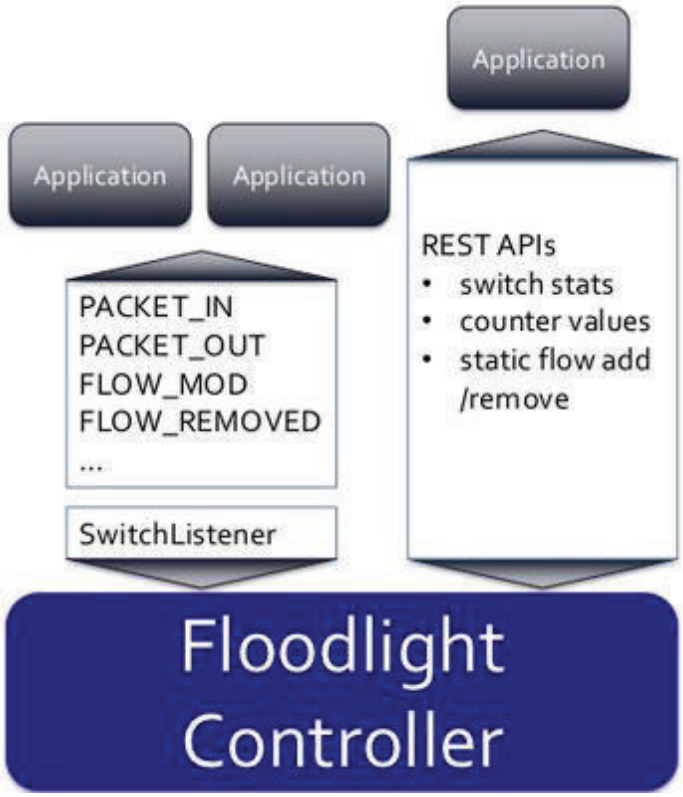

Figura 0.2. Escucha de eventos manejados por Floodlight [4]

1.2.3. RYU

Este controlador está desarrollado en el lenguaje de programación Python, por lo que posee las ventajas de dicho lenguaje. Python es un lenguaje de programación interpretado, o también llamado de *Scripting*[11], por lo que no requiere ser previamente compilado. Su sintaxis y semántica es relativamente sencilla a comparación de lenguajes similares y es totalmente compatible con el paradigma de la programación orientada a objetos.

Una característica heredada de Python en el controlador Ryu es la declaración de variables de tipo dinámico que evita errores de dicho tipo. El controlador Ryu

[11] *Scripting*: en este contexto es sinónimo de lenguaje interpretado, siendo Script un archivo de texto plano que se lee por parte de un intérprete línea por línea.

propone una alternativa de un *framework*[12] para aplicaciones de tipo SDN más ágil a cambio de un controlador monolítico multipropósito. Asimismo la flexibilidad que tiene para las distintas API de cada fabricante lo hace una elección conveniente al iniciar en el desarrollo de aplicaciones SDN [5]. La ubicación del *framework* se ilustra en la Figura 1.3.

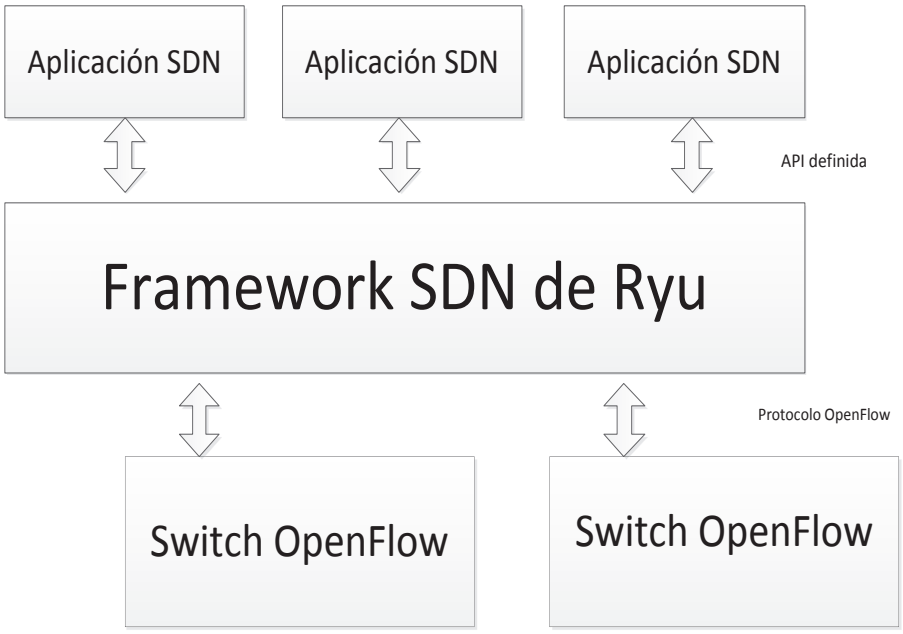

Figura 0.3. Ubicación del *Framework* de Ryu

Asimismo su arquitectura se indica en la Figura 1.4. Los elementos que la constituyen son:

- El *Switch* que soporta OpenFlow
- El operador
- OpenStack
- Aplicaciones de usuario

Dentro del *framework* SDN de Ryu se tiene:

- El serializador del Protocolo OpenFlow
- El serializador de protocolos que no son OpenFlow
- Las librerías
- Aplicaciones ya incluidas en el controlador

[12] *Framework* un entorno de desarrollo de software.

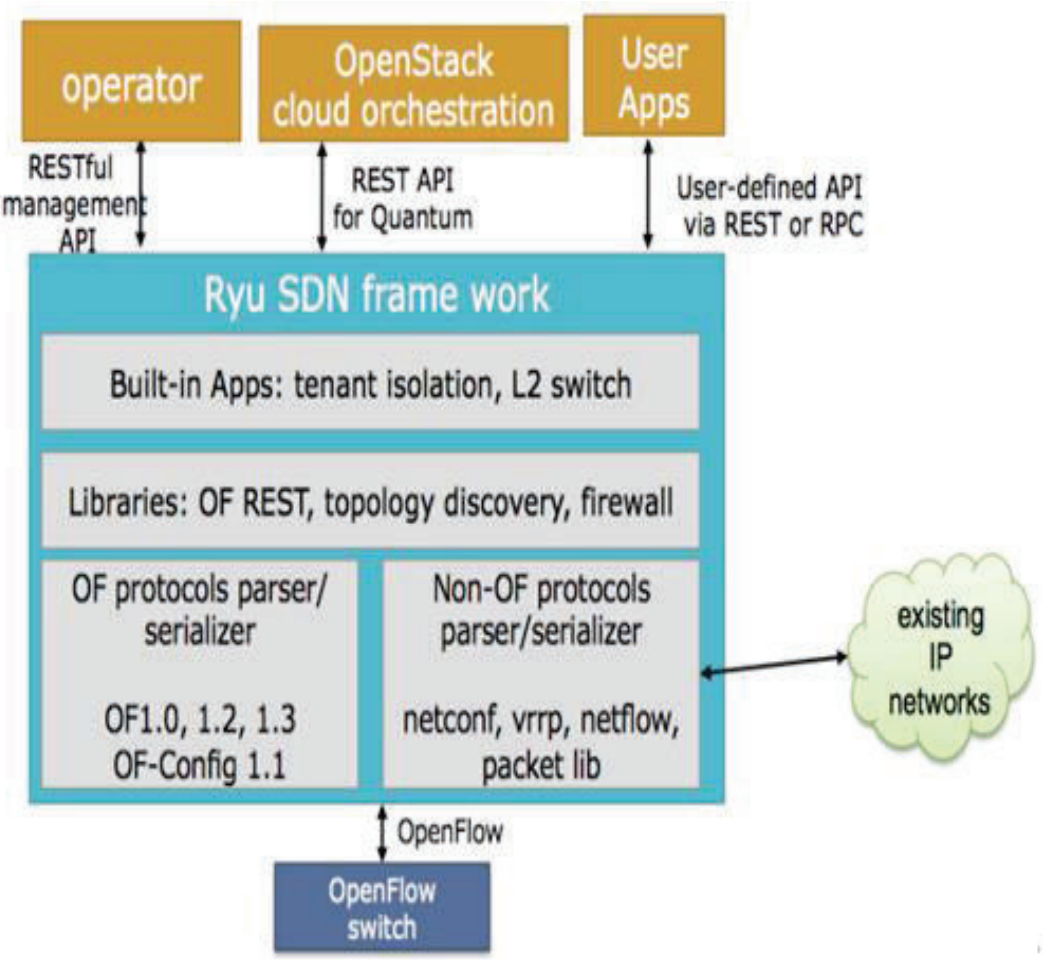

Figura 0.4. Arquitectura de Ryu [6]

1.2.3.1. Componentes de Ryu

Algunos de los componentes incluidos en Ryu son:

- OpenStack *Quantum*: Actualmente conocido como OpenStack Neutron, consiste en un controlador de red en la nube y un proyecto de tipo *networking-as-a-service* perteneciente a la familia de OpenStack, otro proyecto que opera bajo el paradigma de las SDN.
- Firewall: Consiste de un filtro que no es similar a un cortafuego tradicional, pero que es capaz de cumplir las tareas de cortafuegos con el tipo de tráfico presente en una SDN.

- OpenFlow REST: Brinda el manejo apropiado del protocolo OpenFlow, principalmente cumple la tarea de insertar o eliminar una regla de tipo OpenFlow.
- *Topology Viewer* (Visor de Topología): Contiene información dinámica de la topología de red y de los cambios que puedan haber sucedido mediante el uso del protocolo LLDP[13] (*Link Layer Discovery*) que se desarrolla en la siguiente sección, en la Figura 1.5 se muestra la interfaz de este componente.
- CLI: CLI[14] es la interfaz que permite dentro del entorno de Ryu la escritura de instrucciones.
- L2 Switch: Cumple las funciones de conmutación de un *switch* de capa 2 del modelo OSI.
- HA[15] con Zookeeper*:* Es un componente que previene fallos de tipo SPOF (*Single Point of Failure*), Zookeeper monitorea si ocurre un fallo para habilitar la réplica del controlador como lo indica la Figura 1.6 y de este modo no se produce una interrupción en el funcionamiento del controlador ya que entra en operación dicha réplica que existe de Ryu de modo automático.

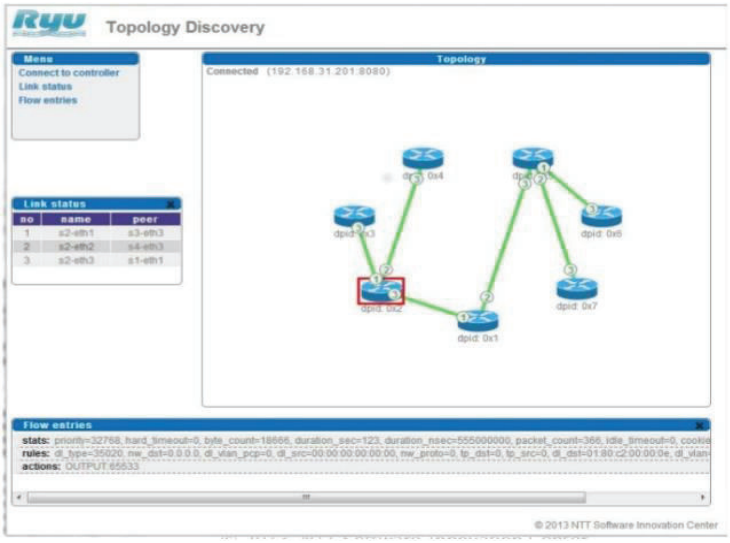

Figura 0.5. Componente Topology Viewer [4]

[13] LLDP (*Link Layer Discovery Protocol*) explicado a detalle en el presene capítulo.
[14] CLI (Interfaz de Línea de Comando) es una interfaz con capacidad de recibir e interpretar instrucciones..
[15] HA (*High Availability)* se traduce como Alta Disponibilidad.

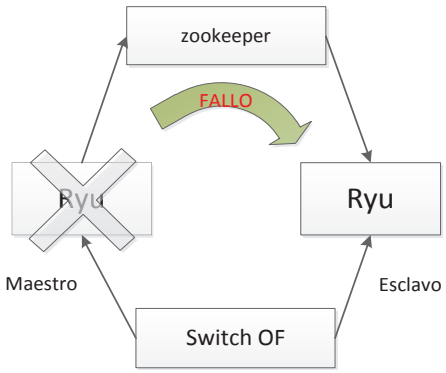

Figura 0.6. Funcionamiento de alta disponibilidad con Zookeeper [5]

- IDS:

IDS (*Intruder Detection System)* es un sistema de detección de intrusos, corresponde a un componente de seguridad que analiza paquetes de manera selectiva ya que hacer un análisis individual de cada paquete puede ocasionar gran carga computacional, pues se requiere un flujo de paquetes replicado del original adicional que vaya hacia el componente Snort indicado en la Figura 1.7.

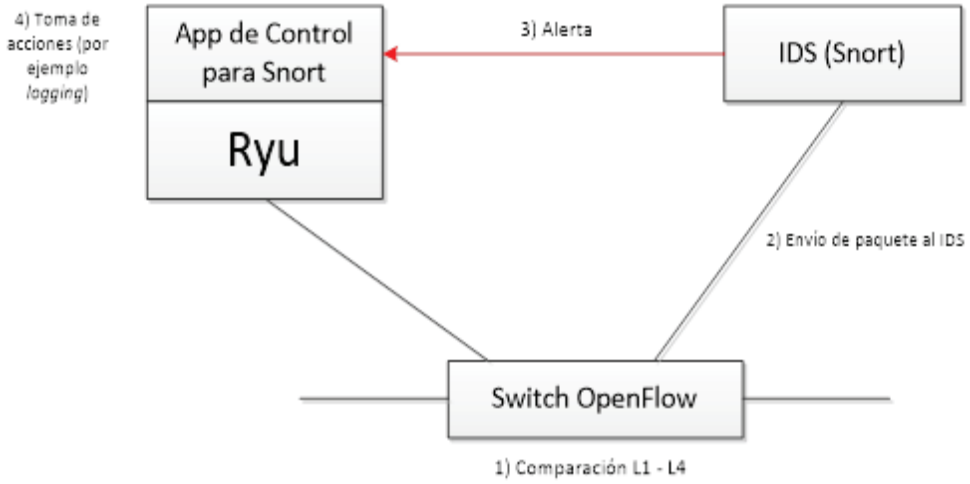

Figura 0.7. Integración de Snort para IDS [5]

Como se menciona previamente Snort requiere gran capacidad computacional para realizar el análisis de paquetes, en caso de requerir su implementación hay dos alternativas: La primera alternativa es ubicar tanto a Ryu como a Snort en la misma máquina, en este escenario observable en la Figura 1.8 las alertas se reciben vía `unixsock` (Unix Domain Socket) y se requiere crear un flujo que replica todos los paquetes a Snort [7].

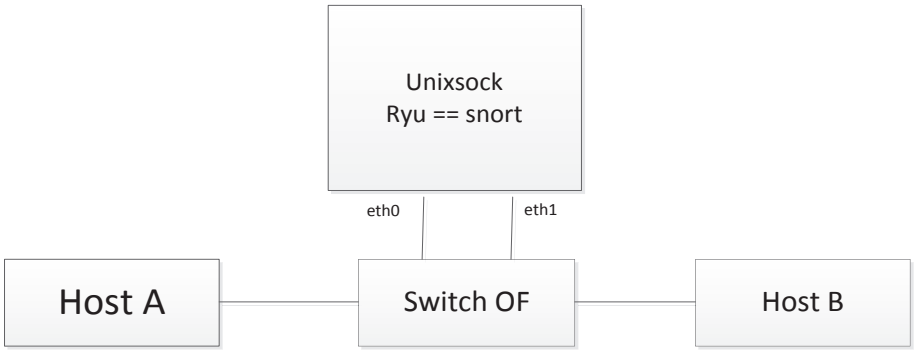

Figura 0.8. Snort y Ryu en la misma máquina

La segunda alternativa expuesta en la Figura 1.9, más recomendable para un escenario de despliegue, es tener a Snort y Ryu en máquinas distintas. En este caso Ryu recibe las alertas de Snort por medio de un Socket de Red. Para el monitoreo de paquetes es necesario crear un flujo que replique los paquetes hacia Snort [7].

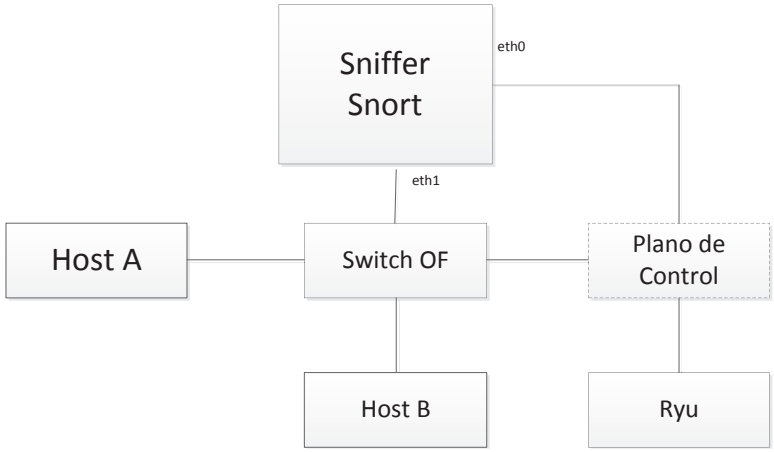

Figura 0.9. Snort y Ryu en máquinas separadas

Para conocer sobre otros componentes se recomienda revisar [5] y [8].

1.2.3.2. Link Layer Discovery Protocol [9]

LLDP es un protocolo desarrollado por Cisco, teniendo en mente que si bien existen diversos protocolos que cumplen las mismas tareas no poseen interoperabilidad. Algunas de las tareas que desempeña LLDP son el descubrimiento de la topología, la solución de problemas de red, proveer información sobre los equipos conectados y la automatización de la administración de la red. El alcance de este estándar es definir un protocolo y elementos de administración que sean capaces de portar y de

ser requerido brindar información a toda estación unida a una LAN (*Local Area Network)* con el propósito de alimentar una base de datos con información de la topología física y descubrimiento de dispositivos presentes [9].

El tipo de base de datos que contienen la información previamente mencionada es llamada MIB[16]. Una MIB tiene estructura jerárquica, en forma de árbol, con información sobre los dispositivos gestionados de una red.

Este tipo de organización fue pensado para ser compatible y legible por parte del protocolo SNMP *(Simple Network Management Protocol)*.

En las Figuras 1.10 y 1.11 se indican los respectivos tipos de dato y valores que contienen dos MIB de descubrimiento, cada una de ellas está asociada a distintos *switches*.

Puede darse el caso del ejemplo indicado en las Figuras 1.10 y 1.11 donde se observa como solamente una de ellas posee el dato `IP-PBX` y así se evidencia que la primera MIB no contiene la misma información que la segunda a pesar que se encuentre en un mismo segmento de red.

Para todos los casos y formatos de MIB la consolidación de la información disponible en cada una de ellas es tarea del protocolo que las manipula.

El formato de la trama LLDP se indica en la Figura 1.12.

port	device	info
A19	Switch	xxxx
C2	IP-Phone	xxxx
D2	IP-Phone	xxxx
F3	IP-PBX	xxxx

Figura 0.10. MIB de descubrimiento de un *switch*

[16] MIB (*Management Information Base)* un concepto usado tanto para las unidades que agrupan valores de distintas varibales asociadas al estado de una red como para la base de datos que se forma a partir del conjunto de dichas unidades.

port	device	info
A4	IP-phone	xxxx
B6	PC	xxxx
B21	Switch	xxxx

Figura 0.11. MIB de descubrimiento de un *switch* distinto

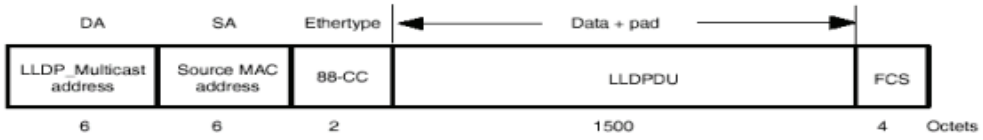

Figura 0.12. Formato de trama LLDP para IEEE 802.3

1.2.3.3. Manejo de Eventos en Ryu

Este controlador al estar basado en componentes debe tener una gestión de la tarea que realiza cada uno de ellos y como se consolida los posibles cambios que provoquen para evitar fallos, para lo cual existen orígenes (*sources*) y sifones (*sinks*) de **eventos** que se deben crear también como componentes. El paso de mensajes se realiza mediante eventos y no de manera directa. Todos estos eventos se manipulan con `eventlet` [10], una librería Python de tipo concurrente[17] para redes que permite el cambio de la ejecución de instrucciones sin alterar la escritura de las mismas. Se puede comparar para su entendimiento con el concepto de *threading* pero la entrada y salida no tiene bloqueo, siendo necesario tomar precauciones para evitar fallos. Para el *Event Dispatcher* (despachador de eventos) se utilizan las clases `AppManager` y `RyuApp`, que son elementos centrales del controlador Ryu. Su funcionamiento consiste en hacer una separación entre *sources* y *sinks* de eventos. Para un *source* se genera uno de los eventos mencionados previamente, mientras que para un *sink* se hace un registro de manejadores dinámicamente.

[17] Una librería concurrente puede ejecutar simultáneamente varias tareas.

El manejo de los eventos del *Dispatcher* identifica y actúa según la clase de cada evento, y hace que dicha información esté disponible para los *sinks* que requieran la información de la clase del evento. También el *Dispatcher* sabrá a cuál método le interesa cuál evento mediante ciertos atributos de métodos. En el interior del *Dispatcher* un evento tendrá temporalmente el atributo de sólo lectura ya que es compartido entre todas las aplicaciones de Ryu, y el *Dispatcher* determinará a cuál RyuApp se entrega el evento basado en su clase, cabe mencionar que existe una cola previa en cada RyuApp hacia donde llegan los eventos y esperan en esta cola en caso que sea necesario.

Desarrollando un poco más los conceptos de programación de *source* y *sink,* un *source* se encarga del llamado a métodos del despachador de eventos para así poder generar eventos; por otra parte un *sink* corresponde a una subclase de la clase RyuApp, y como se menciona previamente el despachador de eventos tiene conocimiento de los *sinks* presentes, así como también conoce cual evento corresponde o más bien interesa a cada método. Los manejadores de eventos (*handlers)* se invocan en su propio hilo y además en su respectiva clase RyuApp, para evitar condiciones de carrera entre eventos consecutivos. Es posible también tener un *direct queueing* (encolamiento directo) en el cual existe un hilo permanente que está constantemente consumiendo eventos, asimismo sobre una RyuApp. En la Figura 1.13 se muestra el orden de los respectivos pasos y procedimientos que ocurren entre RyuApps para la petición y respuesta de eventos.

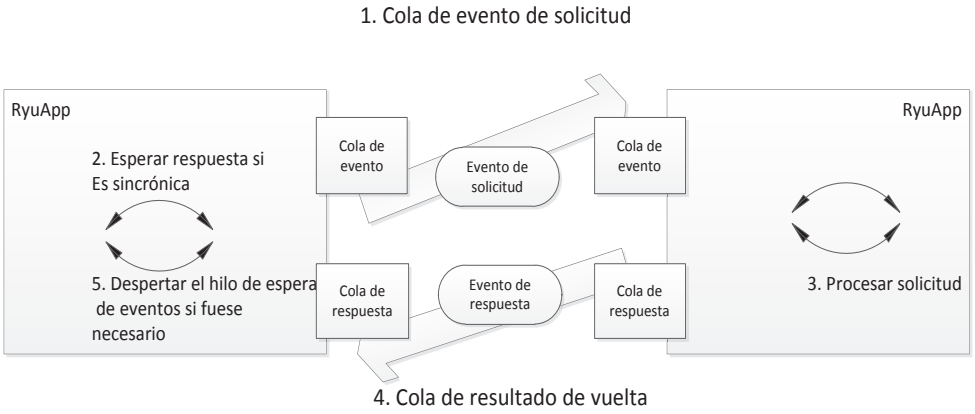

Figura 0.13. Solicitud y Respuesta de Eventos

1.2.3.4. Conexiones en Ryu

Para la conexión hacia un *switch* OF, se hace uso de las clases `OpenFlowController` y la clase `Datapath` con la presencia de un bucle de recepción y un bucle de envío, algunas de sus tareas se indican en la Figura 1.14.

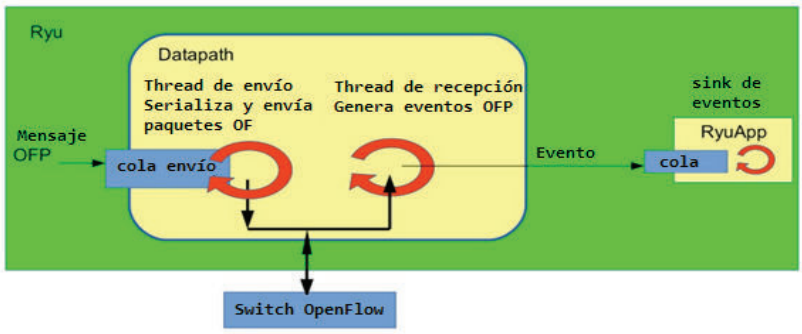

Figura 0.14. Diagrama de conexión entre Ryu y *Switch* OpenFlow [4]

1.2.4. VENTAJAS DE DESARROLLO EN RYU

Al trabajar con aplicaciones SDN es una gran ventaja optar por una alternativa de desarrollo abierto como lo ofrece Ryu, el ambiente de desarrollo es de estilo Linux, lo cual resulta familiar y amigable a un gran número de desarrolladores en distintas plataformas. Una ventaja que posee esta selección de ambiente de trabajo es el soporte de la comunidad, se puede iniciar discusiones abiertas en la lista de correos que siempre es sugerida al momento de instalar el controlador y definitivamente es de gran ayuda en caso de querer aclarar dudas con una respuesta pronta y eficaz.

Ryu tiene soporte a través de GitHub, un servicio de compartición y publicación de código con facilidades de tipo red social para que los desarrolladores tengan realimentación de los demás miembros de la comunidad.

GitHub, al ser un repositorio, principalmente ofrece un control de versiones del código manejado a través de él, y las estadísticas mostradas sobre el desarrollo y aportes de Ryu indican que se trata de un proyecto activo y que ha crecido significativamente desde sus inicios [11].

Las ventajas adicionales de Ryu derivan del lenguaje utilizado. Es fácil y breve de aprender y aún más si se tiene experiencia previa en lenguajes de programación orientados a objetos.

Las características que serán útiles para el desarrollo de aplicaciones bajo Ryu presentes en este lenguaje son:

- Lenguaje dinámico
- Decoradores[18]
- Introspección[19]

Además de disponer de un amplio número de librerías de utilidad para los distintos propósitos de una aplicación a desarrollarse.

1.3. PROTOCOLO OPENFLOW

1.3.1. CARACTERÍSTICAS

OpenFlow es un estándar de tipo abierto con la capacidad de soportar y correr protocolos experimentales. Se puede encontrar como una característica incluida en *switches* Ethernet, *routers* o puntos de acceso inalámbricos comerciales [1].

OpenFlow constituye la primera interfaz de comunicación estandarizada entre las capas de control y *forwarding*[20] de una arquitectura SDN. Permite un acceso directo y la posibilidad de manipular el plano de conmutación de los dispositivos de red previamente mencionados, tanto para dispositivos físicos como para dispositivos virtuales.

Entre las características de OpenFlow se puede mencionar:

- Programabilidad: Permite innovar y diferenciar las soluciones a ser implementadas, y también acelerar la creación de nuevas características y la introducción de nuevos servicios.
- Inteligencia Centralizada: Simplifica la provisión de recursos de parte de los dispositivos de red para realizar tareas, resultando en una optimización del

---

[18] Un decorador consiste en pasar como argumento de una función 'a' otra función 'b' retornando una nueva función 'c', esta nueva función 'c' resulta ser la función 'b' decorada con 'a'

[19] La introspección es código que examina como objetos otros módulos y funciones en memoria, obtiene información sobre ellos y los maneja. La introspección es propia de todo lenguaje orientado a objetos, donde "todo es un objeto".

[20] *Forwarding*: capacidad de conmutación de un dispositivo de capa 2 del modelo ISO/OSI.

rendimiento del sistema. Además esta característica facilita el manejo de políticas granular.

- Abstracción: Desacopla hardware y software, los planos de control y de *forwarding* (también llamado plano de datos) y también desacopla las configuraciones de tipo físico y las de tipo lógico.

Asociando dichas características con las capas presentes en la arquitectura SDN para la capa de Aplicación se tiene la ventaja de Programabilidad, para la capa de control se tiene la Inteligencia Centralizada y para la capa de infraestructura se tiene la Abstracción. La Figura 1.15 muestra la ubicación del protocolo OpenFlow en la arquitectura SDN.

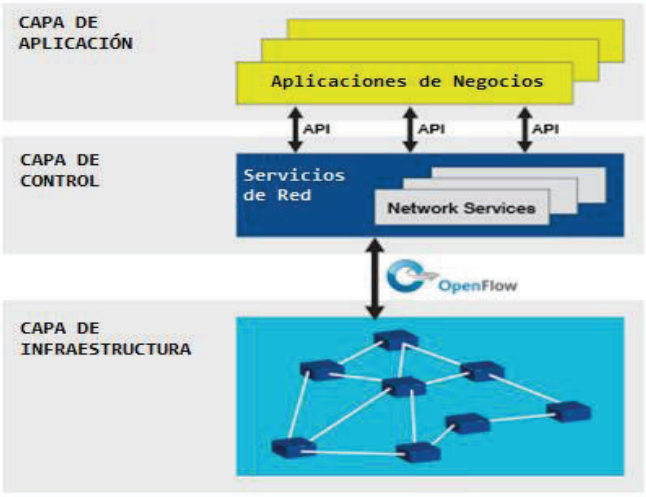

Figura 0.15. Ubicación de OpenFlow en la Arquitectura SDN [12]

La ausencia de una interfaz abierta es responsable de que los dispositivos de red actuales y sus fabricantes constituyan sistemas monolíticos, de arquitectura cerrada e incluso de características funcionales de tipo caja negra. OpenFlow provee una solución a la necesidad de mover el control en *networking* desde los *switches* hacia un software de control centralizado [1] [13]. OpenFlow se puede comparar con el conjunto de instrucciones de una CPU. Como se muestra en la Figura 1.16, el protocolo va a especificar las primitivas básicas que pueden ser utilizadas para programar el plano de control de dispositivos de red, al igual que el conjunto de instrucciones de una CPU permite programar un sistema informático.

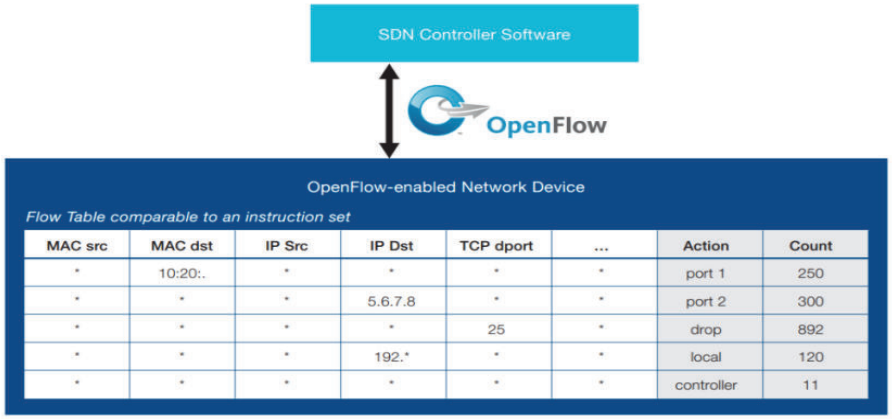

Figura 0.16. Tabla OpenFlow de flujos y su *set* de instrucciones [1]

OpenFlow es implementado en los dispositivos de red y también en el controlador SDN. Utiliza el concepto de flujos para identificar el tráfico de red y se basa en reglas predefinidas, las cuales pueden ser estática o dinámicamente programadas directamente en el controlador y además permite definir cómo el tráfico debe circular a través de los dispositivos de red, basados en ciertos parámetros, como por ejemplo los patrones de uso, aplicaciones fuente / destino y recursos.

Si se compara con el enrutamiento actual basado en el protocolo IP[21] (el cual no proporciona un nivel de control) donde los flujos entre dos puntos finales deben seguir caminos a través de la red independientemente de sus diferentes requerimientos; si se utiliza OpenFlow se puede lograr que la red sea programada en base a flujos, proporcionando un control extremadamente granular, permitiendo a la red responder en caso que existan cambios en tiempo real a nivel de aplicación, usuario y sesión. OpenFlow como protocolo es una parte clave en las SDN por permitir la directa manipulación del plano de datos en dispositivos de red. Si bien inicialmente solo se aplicó a redes basadas en Ethernet, OpenFlow es extensible a varios casos de uso. SDN basado en OpenFlow puede ser aplicado a las redes existentes, tanto físicas como virtuales.

Este protocolo puede ser usado como apoyo en conjunto con mecanismos de *forwarding* tradicionales, lo que facilita la transición a empresas que deseen introducir

---

[21] IP (*Internet Protocol*) es un protocolo que se encarga de la transmisión de paquetes de datos a través de Internet.

tecnologías SDN basadas en OpenFlow, incluso en caso de ambientes de red de múltiples fabricantes.

1.3.1.1. Tablas de Flujos

Un componente fundamental de OpenFlow mencionado anteriormente son las tablas de flujos (*flow tables*) que contiene un conjunto de registros instalados en cada uno de los *switches* que indican a cada dispositivo que hacer con el tráfico.

Cada *switch* previamente realizará una consulta en su tabla de flujos antes de procesar su tráfico conforme al contenido de las mismas. En caso de no saber qué hacer con ese tráfico se hace una consulta al controlador quien indicará como proceder, esto refuerza el concepto de que la red se gestiona de manera centralizada. Algunas ventajas de las tablas de flujos es que permiten, debido a la toma de decisiones centralizada, aplicar QoS[22], tener rutas de menor latencia, mejorar la velocidad, entre otras.

1.3.2. MENSAJES EN OPENFLOW

OpenFlow maneja mensajes para poder indicar acciones o transmitir información tanto desde el controlador hacia los dispositivos como viceversa, es necesario diferenciar cada uno de los tipos de mensaje que maneja el protocolo. Los tipos de mensajes de OpenFlow [13] son:

- Mensajes del controlador al *switch*
- Mensajes asincrónicos
- Mensajes simétricos

1.3.2.1. Mensajes del Controlador al *Switch*

Estos mensajes se inician desde el controlador y pueden o no requerir una respuesta por parte del *switch*. El controlador puede solicitar las características de un *switch*

[22] QoS (*Quality of Service*) Calidad de Servicio es el concepto de garantizar la transmisión de cierta cantidad de información en un tiempo determinado.

enviando una solicitud de tipo `Features`, el *switch* replicará con una respuesta que especifica las capacidades del *switch*, es una función utilizada antes de crear un flujo. Un controlador es capaz de establecer y preguntar parámetros de configuración en el *switch*. El *switch* solamente responderá a consultas del controlador. Los mensajes de tipo `Modify-State` (estado de modificación) se envían desde el controlador para administrar el estado de los *switches*.

Su propósito principal es añadir, eliminar o modificar flujos y grupos en las tablas de OpenFlow y para establecer propiedades en los puertos del *switch*.

Los mensajes `Read-State` (estado de lectura) se usan por parte del controlador para recoger estadísticas brindadas por el *switch*.

Los mensajes `Packet-Out` los usa el controlador para enviar paquetes a un puerto específico del *switch* y para conmutar paquetes recibidos en respuesta a mensajes de tipo `Packet-In`.

Estos mensajes deben contener una referencia o ID de cierto paquete almacenado en el *switch*.

El mensaje debe también contener una lista de acciones a ser aplicadas; en caso de encontrarse con una lista de acciones vacía se descarta dicho paquete.

Finalmente los mensajes de solicitud `Barrier` (barrera) se usan por parte del controlador para asegurar que las dependencias de mensajes se cumplen o para recibir una notificación de operaciones completadas.

1.3.2.2. Mensajes Asíncronos

Estos mensajes se envían sin previa solicitud del controlador a un *switch*. El *switch* envía mensajes asíncronos al controlador para indicar la llegada de paquetes, cambios del estado del *switch* o errores. Los principales mensajes asíncronos se indican a continuación:

Un mensaje `Packet-In` es un mecanismo del *switch* para enviar un paquete capturado al controlador.

Hay tres motivos para originar este mensaje; que ocurra una acción explícita como resultado de un emparejamiento pidiendo este comportamiento, que exista un fallo

en las tablas, o en el caso de existir un error de TTL[23]. Un mensaje `Packet-In` se inicia en un *switch.* Su estructura se indica en la Figura 1.17.

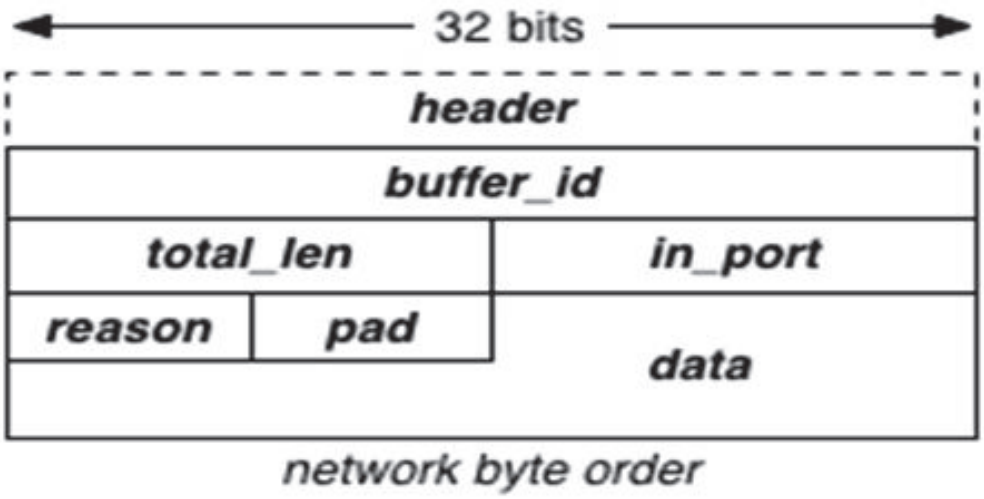

Figura 0.17. Estructura de un mensaje `Packet-In` [14]

El mensaje `Packet-In` consiste de una cabecera (*header)* seguida de un identificador de *buffer*[24] (*buffer_id)*. El *buffer_id* es un valor único usado para dar seguimiento al paquete en *buffer.*

La longitud del paquete capturado se indica en el campo *total_len.* El puerto en el que el paquete fue recibido se indica en el campo *in_port.* El campo *reason* indica por qué el paquete fue capturado y conmutado. Finalmente la carga útil del paquete empieza a continuación del campo *pad*[25].

El mensaje `Flow-Removed` funciona de este modo: cuando una entrada de flujo se añade al *switch*.

Un valor de tiempo de inactividad indica en qué momento debe ser eliminada la entrada por falta de actividad, o si se cumple un *hard timeout*[26]*;* entonces sin importar el estado de actividad se elimina dicha entrada de flujo. El mensaje `Port Status` (estado de puerto) se espera que sea enviado desde el *switch* al controlador cuando

---

[23] TTL (*Time To Live)* un contador para indicar el tiempo de vida de un paquete.
[24] Un *buffer* es un área de contención temporal mientras se traslada cierta información.
[25] *Pad* en este contexto significa relleno.
[26] *Hard timeout* es el número de segundos después de lo cual se elimina el flujo de la tabla y del hardware sin importar el número de paquetes que concuerden con el flujo

existe un cambio en la configuración o en el estado de un puerto, como por ejemplo si el usuario bajó un puerto.
El mensaje `Error` es usado para notificar al controlador problemas diversos.

1.3.2.3. Mensajes Simétricos

Son mensajes que pueden enviarse en cualquier dirección bien sea del controlador al *switch* o viceversa. Algunos de ellos y su propósito se explican a continuación:
Los mensajes `hello` son intercambiados cuando se establece por primera vez la conexión entre el *switch* y el controlador.
Los mensajes `echo` son utilizados como medidor de la latencia de las conexiones y como prueba de actividad de un dispositivo (se puede asociar al funcionamiento del comando *ping).* Tanto el *switch* como el controlador pueden generar este mensaje y siempre existen tanto una respuesta como una solicitud de este tipo.
Los mensajes `experimenter` proveen una manera estandarizada en un *switch* OpenFlow de extender funcionalidad en el espacio de tipos de mensajes. Se planea en futuras revisiones del protocolo trabajar más a detalle este mensaje.

1.4. MININET

Mininet es una herramienta de software, capaz de emular una red virtual con código de *kernel*, *switches* y aplicaciones, todo dentro de la misma máquina que puede ser de tipo virtual, física o en la nube [15].
La interacción entre el usuario y Mininet se da a través de una CLI y su API respectiva, para poder personalizarlo, compartirlo o desplegarlo en producción sobre hardware real. Por estas características Mininet se puede utilizar en ambientes de desarrollo, investigación y enseñanza.
Su fortaleza y mayor escenario de aplicación es la experimentación con OpenFlow y en general con sistemas que trabajan con SDN. Esta herramienta brinda versatilidad en cuanto al armado y emulación de una red completa, las distintas opciones de la herramienta se desarrollan en el siguiente capítulo del presente Proyecto.

1.5. DHCP

1.5.1. GENERALIDADES

El protocolo de configuración dinámica de *hosts* (DHCP) provee parámetros de configuración a un *host* solicitante dentro de un ambiente de red. Un *host* es cualquier equipo conectado a una red con capacidades de proveer o utilizar servicios dentro de dicha red [16].

DHCP consta de dos componentes: el protocolo que se encarga de entregar la configuración de parámetros específica para un *host* desde el servidor; y un mecanismo de asignación de direcciones de red a los *hosts*.

La arquitectura de DHCP es de tipo Cliente-Servidor, y es de gran importancia el no confiar esta tarea a cualquier equipo, pues la diversidad de hardware que puede solicitar una dirección IP no permite que existan el azar o suposiciones como criterio de asignación.

El punto anterior resalta la necesidad de tener un sistema de consulta (*polling*[27]) y descubrimiento constante de modo que no se reutilice direcciones ya asignadas y ocurran errores. DHCP brinda la posibilidad de asignar direcciones dinámicamente o estáticamente.

Pueden tener una asignación dinámica para *hosts* que no se conecten regularmente o por largo tiempo. Generalmente existe todo un *pool* (rango) con direcciones de esta naturaleza a disposición de este tipo de *hosts*; por otra parte, se puede tener una asignación manual útil para equipos, como por ejemplo servidores, que de preferencia siempre tendrán una dirección fija.

El formato de los mensajes DHCP es heredado de los mensajes BOOTP[28] y también permite que haya interoperabilidad con clientes de tipo BOOTP, los cuales pueden ser atendidos por un servidor DHCP.

---

[27] *Polling* corresponde a un encuestado dirigido a los equipos de red cuyos parámetros asignados se manejan via DHCP.

[28] BOOTP (*Bootstrap Protocol*) usado para asignación de parámetros de red vía UDP.

1.5.2. FORMATO DE MENSAJE DHCP

La Figura 1.18 indica los distintos campos y tamaños en octetos de un mensaje DHCP. El tamaño en bytes de los respectivos campos se indica de igual manera. La Tabla 1.1 presenta el funcionamiento de cada campo y su tamaño en octetos con una descripción de la información contenida:

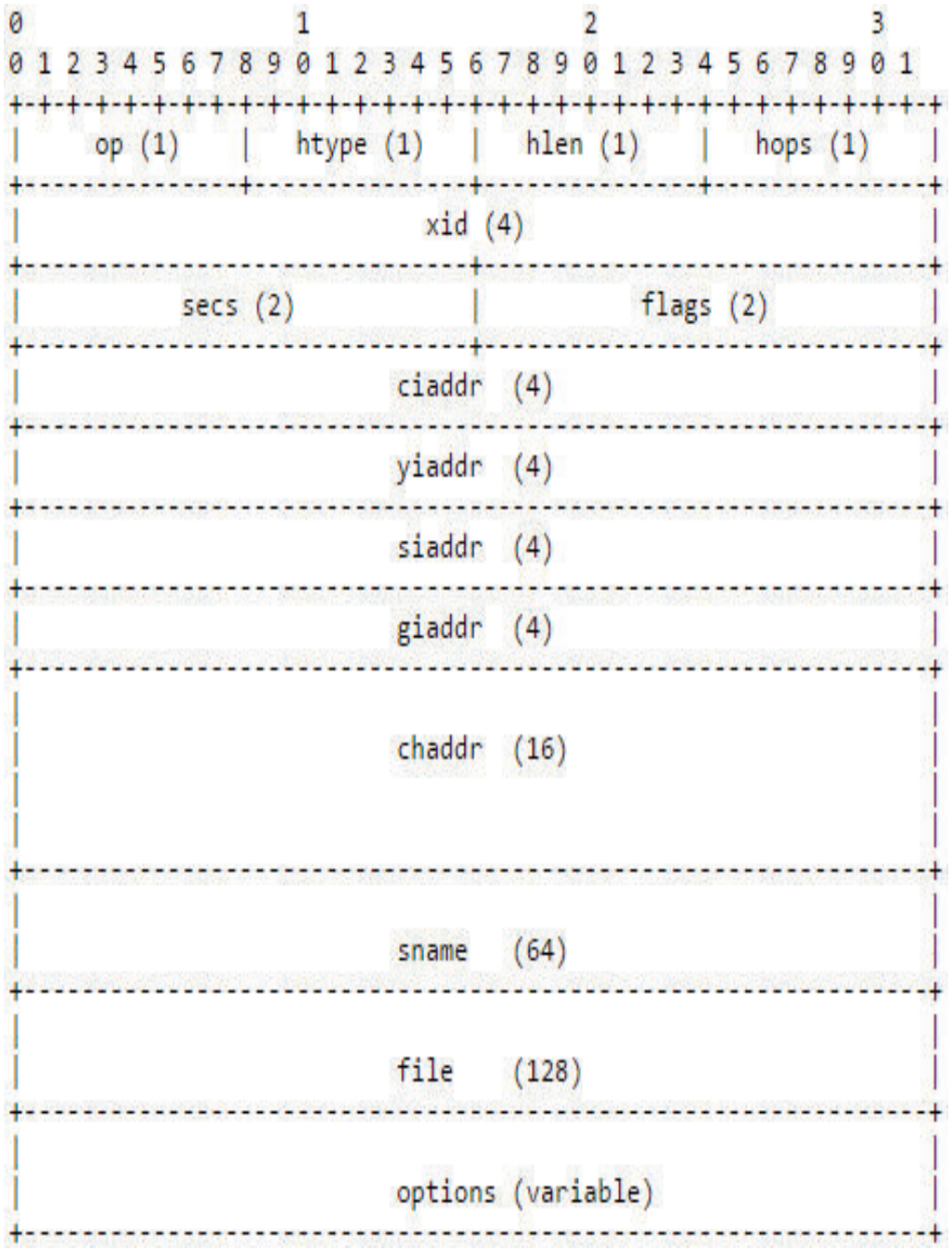

Figura 0.18. Mensaje DHCP en octetos [16]

CAMPO	OCTETOS	DESCRIPCIÓN
Op	1	Código OP del mensaje / tipo de mensaje 1 = BOOTREQUEST 2 = BOOTREPLY
Htype	1	Tipo de dirección hardware (ej. 1 = Ethernet de 10 Mbps)
Hlen	1	Longitud de dirección hardware (ej. 6 para Ethernet de 10Mbps)
Hops	1	Establecido en cero, uso opcional en agentes relay[29]
Xid	4	ID de transacción, un número aleatorio escogido por el cliente, usado entre cliente y servidor para asociar solicitudes y respuestas entre ellos.
Secs	2	Llenado por el cliente, indica la cantidad de segundos transcurridos desde la adquisición o renovación de una dirección.
Flags	2	Banderas
Ciaddr	4	Dirección IP del cliente, contiene información sólo si el cliente está en estado BOUND[30], RENEW[31] o REBINDING[32] y entonces es capaz de responder una solicitud ARP[33]
Yiaddr	4	Dirección IP del cliente
Siaddr	4	Dirección IP del servidor más cercano, retornado a partir de un mensaje DHCPOFFER con un mensaje DHCPACK desde el servidor
Giaddr	4	Dirección IP del Agente Relay
Chaddr	16	Dirección hardware del cliente
Sname	64	Nombre de host del servidor, opcional
File	128	Nombre del archivo de boot, que puede ser NULL para DHCPDISCOVER, o ruta completa para DHCPOFFER
Options	variable	Campo opcional de parámetros que posee documentación específica para su relleno, aún se encuentra en desarrollo

Tabla 0.1. Descripción de cada campo de un mensaje DHCP

---

[29] Un agente *relay* se encarga de retransmitir mensajes DHCP o BOOTP entre subredes.
[30] El estado *BOUND* implica que el cliente ya es restringido al arrendamiento de una dirección y opera normalmente.
[31] El estado *RENEW* ocurre cuando el cliente está tratando de renovar su arrendamiento.
[32] El estado *REBINDING* ocurre si el cliente no ha podido renovar su contrato de arrendamiento con el servidor que originalmente le concedió parámetros, y ahora busca una extensión de arrendamiento con cualquier servidor que pueda atenderlo.
[33] ARP (*Address Resolution Protocol*) es un protocolo de capa de Red que asocia una dirección Hardware a una dirección IP.

1.5.3. TIPOS DE MENSAJE DHCP

Para el funcionamiento del protocolo cliente-servidor DHCP existe un intercambio de mensajes. En la Tabla 1.2 se define la función de cada mensaje dentro del proceso de funcionamiento de DHCP.

MENSAJE	USO
DHCPDISCOVER	Mensaje de tipo *broadcast* generado por el cliente para la localización de servidores DHCP disponibles
DHCPOFFER	Servidor hacia cliente, respuesta a DISCOVER con una oferta de parámetros de configuración
DHCPREQUEST	Cliente hacia servidor, opciones: (1) solicitando los parámetros ofertados e implícitamente rechazando ofertas de otros, (2) Confirmando que la dirección asignada es correcta ej. Después de un reinicio de sistema, o (3) extensión del préstamo de una dirección en particular.
DHCPACK	Servidor a cliente, contiene parámetros de configuración que incluyen la dirección comprometida
DHCPNAK	Servidor hacia cliente, indica que la dirección cliente ahora es incorrecta, porque se ha movido a otra subred, el préstamo ha expirado, entre otras causas.
DHCPDECLINE	Cliente a servidor, indica que la dirección propuesta ya está en uso
DHCPRELEASE	Cliente a servidor, renuncia a la dirección alquilada y cancela el tiempo restante de arrendamiento en el lado del servidor
DHCPINFORM	Cliente hacia servidor, petición de parámetros de configuración local cuando el cliente ya tiene dirección de red externamente configurada.

Tabla 0.2. Mensajes del protocolo DHCP

1.5.4. PROCESO DE ASIGNACIÓN DE DIRECCIÓN DE RED

En el proceso de asignación de una dirección de red (incluidos parámetros adicionales como direcciones de puerta de enlace, DNS, entre otros) ocurren distintos intercambios de mensajes para lograr exitosamente la asignación, los cuales se describen a continuación.

Como primer paso el cliente hace un *broadcast* con un mensaje de tipo DHCPDISCOVER dentro de la subred. El mensaje DHCPDISCOVER puede incluir opciones con valores como dirección de red y tiempo de arrendamiento. En caso de haber un agente BOOTP de *relay* es posible pasar dicho mensaje a servidores de otras subredes.

A continuación, cada servidor puede responder con un DHCPOFFER incluyendo una dirección disponible en el campo YIADDR y parámetros adicionales en las opciones DHCP.

Un servidor no requiere reserva de la dirección en este punto, puede ser conveniente para el protocolo hacerlo y así no asignar a otro cliente dicha dirección.

Es tarea del servidor comprobar que la dirección ofertada no se encuentre en uso (puede hacerlo mediante un mensaje de tipo ICMP).

Un servidor tendrá la opción de deshabilitar el sondeo de direcciones asignadas. El servidor transmite un DHCPOFFER a través de un *relay,* si fuese el caso.

Después de esto el cliente recibirá tantos mensajes DHCPOFFER como servidores haya en la subred.

El cliente escoge a uno de los ofertantes para solicitar parámetros de configuración, en base a la información contenida en el mensaje DHCPOFFER.

Entonces el cliente envía un mensaje tipo DHCPREQUEST que debe incluir la opción de identificación del servidor indicando el servidor seleccionado y optativamente incluya parámetros deseados adicionales.

La opción de dirección IP escogida debe coincidir en este momento con el valor de YIADDR del mensaje DHCPOFFER del servidor.

El mensaje DHCPREQUEST que claramente establece cual fue el servidor escogido debe ser retransmitido vía *relay* hacia las ubicaciones de red que pudiesen ofertar el servicio de DHCP.

Para garantizar que el agente de retransmisión BOOTP reenvíe el mensaje DHCPREQUEST al mismo conjunto de servidores DHCP que recibieron el primer mensaje DHCPDISCOVER.

En este mensaje DHCPREQUEST debe coincidir el valor del campo SECS contenido en la cabecera del mensaje DHCP para de este modo poder ser enviado a la misma dirección tipo *broadcast* a la que fue enviado el mensaje DHCPDISCOVER original.

En caso que el cliente no reciba ningún mensaje DHCPOFFER vuelve a transmitir mensajes DHCPDISCOVER después de un tiempo de espera.

Posteriormente, los servidores reciben el mensaje *broadcast* DHCPREQUEST desde el cliente, en caso de no haber sido seleccionados toman este mensaje como una notificación de declinación de la oferta por parte del cliente.

El servidor seleccionado por el mensaje DHCPREQUEST del cliente confirma la unión del cliente para almacenarlo permanentemente mediante un mensaje DHCPACK que contiene todos los parámetros de configuración para el cliente solicitante.

Una combinación entre el campo identificador de cliente o CHADDR y la dirección asignada se toman como identificador único del arrendamiento y dicho identificador es usado tanto por el cliente como el servidor para referencia en cualquier mensaje DHCP.

Los parámetros contenidos en el mensaje DHCPACK no deben tener conflicto con los parámetros del mensaje DHCPOFFER al cual ya respondió el cliente.

La comprobación por parte del servidor no se realiza en este punto; en caso que el servidor seleccionado no puede satisfacer los requerimientos del mensaje DHCPREQUEST el servidor deberá responder inmediatamente con un mensaje DHCPNAK.

Puede darse el caso en que el servidor marcará direcciones como no disponibles y además el caso en que el servidor podría marcar como disponible una dirección en un mensaje DHCPOFFER si el servidor no recibe un mensaje DHCPREQUEST desde ese cliente. A continuación el cliente recibe el mensaje DHCPACK con los parámetros de configuración. El cliente debería comprobar los parámetros (vía ARP por ejemplo) y conoce la duración del préstamo de la dirección.

En este punto, el cliente ya está configurado. Si el cliente detecta que la dirección ya está en uso se debe enviar un mensaje DHCPDECLINE e iniciar el proceso desde el inicio.

Para evitar el tráfico de red excesivo se procura esperar un tiempo de diez segundos antes de reanudar el proceso.

El cliente puede recibir un mensaje DHCPNAK, lo cual le indica que se reiniciará el proceso de configuración.

El criterio generalizado para no generar demasiadas peticiones en caso de no lograr una configuración correcta dependerá de los tiempos y se escogerá adecuadamente, y debe contemplar que se dé un tiempo suficiente para que el mensaje DHCPREQUEST tenga una probabilidad alta de contactar al servidor y a su vez no cause que el cliente espere demasiado y termine dándose por vencido.

El cliente puede optar por renunciar al alquiler de una dirección de red mediante el envío de un mensaje DHCPRELEASE al servidor.

El cliente identifica el contrato de arrendamiento a ser liberado junto con su identificador de cliente (CHADDR) y la dirección de red en el mensaje DHCPRELEASE.

Si el cliente utilizó un identificador de cliente cuando obtuvo el contrato de arrendamiento, se deberá utilizar el mismo identificador de cliente en el mensaje DHCPRELEASE.

Los pasos para la asignación de una dirección inicialmente se resumen en la Figura 1.19.

En caso de un apagado secuencial y adecuado del sistema, es decir cerrando apropiadamente el sistema operativo se indica en la Figura 1.19 la presencia del mensaje DHCPRELEASE previo al descarte del arrendamiento correspondiente.

El proceso para adquirir una dirección que previamente se le asignó a dicho cliente, también conocido como renovación, se simplifica en gran parte como se observa en la Figura 1.20, pues solamente es necesario reanudar la transacción a partir de la llegada de un mensaje de tipo DHCPREQUEST.

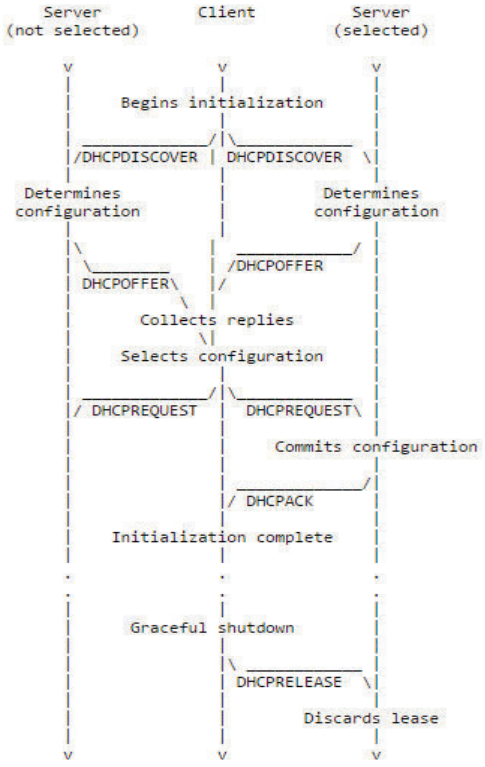

Figura 0.19. Diagrama de intercambio de mensajes para asignación inicial de una dirección

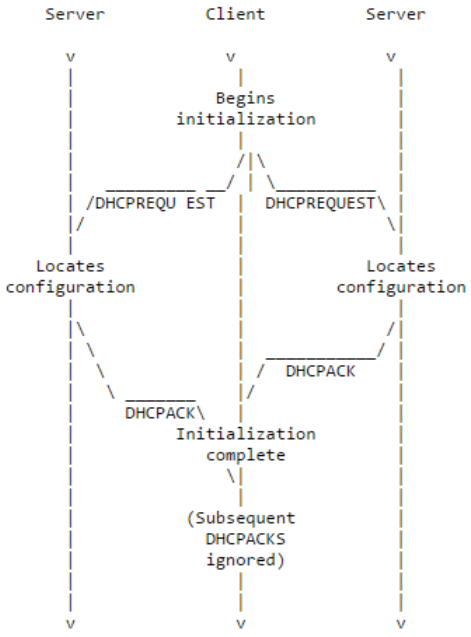

Figura 0.20. Diagrama de intercambio de mensajes en caso de reúso de dirección

CAPÍTULO II

2. DESARROLLO DE LA APLICACIÓN

En este capítulo se indican los prerrequisitos necesarios para el funcionamiento de la aplicación, una breve guía de configuración de los equipos y sistemas empleados para desarrollar y probar la aplicación, y el código desarrollado de la aplicación prototipo que cumple con las tareas de asignación de los parámetros de red mediante el protocolo DHCP.

2.1. AMBIENTE DE FUNCIONAMIENTO DEL CONTROLADOR RYU

El controlador Ryu está disponible en el repositorio GitHub[34] para su instalación directa en cualquier equipo de escritorio que tenga instalado un sistema operativo Linux.

El sistema operativo escogido para el desarrollo de la aplicación fue inicialmente Ubuntu 12.04, el cual posteriormente fue sustituido por Ubuntu 14.04 LTS al haberse encontrado conflictos con librerías para el funcionamiento de Open vSwitch (OVS) que es parte de la máquina virtual de Mininet, la importancia de este componente y de la herramienta de emulación Mininet será detallada más adelante.

2.1.1. INSTALACIÓN DE RYU

Previo a la instalación de Ryu es necesario disponer del manejador de paquetes Pip[35], el cual permite instalar cualquier librería de Python necesaria en el transcurso del desarrollo de aplicaciones. Para el desarrollo de la aplicación prototipo se requirió del paquete `paramiko`. El paquete `paramiko` [17] dispone de una implementación del protocolo SSH mediante una interfaz escrita en Python. Para instalar Pip es

---

[34] GitHub es un sitio web donde se puede usar o contribuir con código y con la ventaja del manejo de versiones.

[35] Pip es un sistema de manejo de paquetes especializado en paquetes escritos en Python.

necesario descargar el paquete `get-pip.py` [18] y posteriormente, en el directorio donde se encuentra el archivo descargado, se debe ejecutar el Código 2.1 en una terminal con privilegios de usuario `root` para instalar el paquete.

```
1 # python get-pip.py
2 # pip install paramiko
```

Código 2.1. Instalación de `pip` y de `paramiko`

Para instalar el *framework* de Ryu es recomendable realizar este proceso desde GitHub directamente mediante una acción denominada *clone*[36]. En el Código 2.2 se muestran los comandos necesarios para clonar e instalar Ryu.

```
1 % git clone git://github.com/osrg/ryu.git
2 % cd ryu; python ./setup.py install
```

Código 2.2. Instalación de Ryu

En caso de presentarse errores en la instalación es necesario instalar las dependencias listadas en el Código 2.3.

```
1 % sudo pip install gevent routes webob
```

Código 2.3. Dependencias necesarias para el funcionamiento de Ryu

Adicionalmente es recomendable actualizar todos los repositorios y paquetes del sistema mediante los comandos indicados en el Código 2.4.

```
1 # apt-get update
2 # apt-get upgrade
```

Código 2.4. Actualización de repositorios de Ubuntu

2.2. AMBIENTE PARA EMULAR UNA RED SDN DE PRUEBAS

La aplicación Ryu correrá bajo el sistema nativo y Mininet correrá dentro de una máquina virtual, se indica el diagrama de la red que se empleará para este propósito en la Figura 2.1.

La dirección IP de la interfaz `vboxnet0` la asigna automáticamente VirtualBox, por esto el resto de interfaces se las configura dentro de la misma red (192.168.56.0/24). Las direcciones escogidas son arbitrarias, y sirven exclusivamente para

[36] En GitHub la acción *clone* hace una copia del código fuente desde el sitio web hacia el equipo desde el cual se ejecutó la acción.

comunicación entre la máquina virtual y el sistema nativo, no influyen ni interfieren el momento de la puesta en marcha de la aplicación prototipo DHCP.

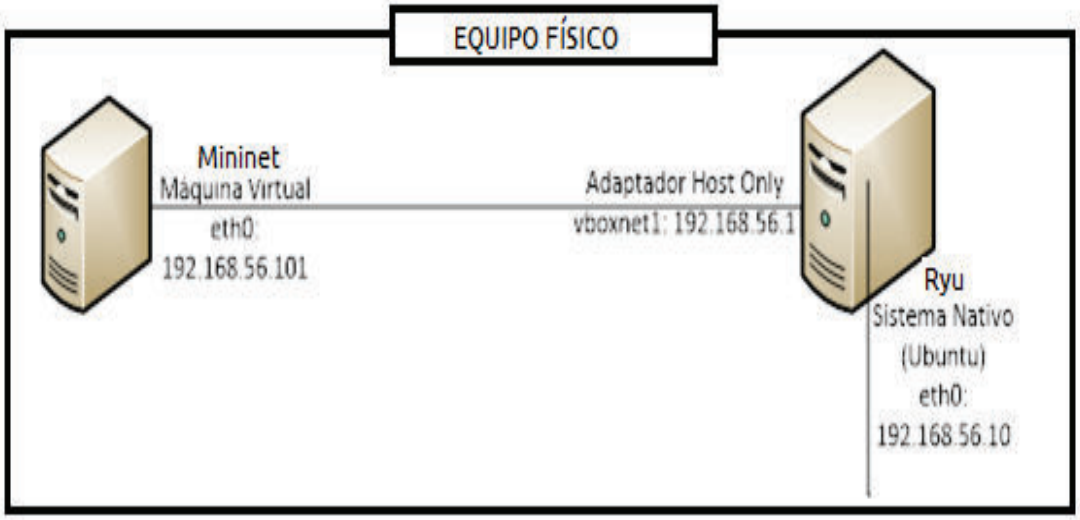

Figura 2.1. Diagrama de Red para conectar Mininet y Ryu

Es necesario configurar la interfaz `vboxnet0` y luego se deben configurar las interfaces de red del equipo físico y de la máquina virtual, para finalmente instalar Mininet.

En el Anexo A se indica cómo configurar la interfaz `vboxnet0` y adicionalmente se indica cómo establecer los parámetros de las interfaces de red (`eth0`) del sistema nativo y de la máquina virtual, así como también se encuentra una guía para la instalación de Mininet y de los requisitos previos de conectividad entre el sistema nativo y la máquina virtual.

2.3. CONOCIMIENTOS PREVIOS A LA FASE DE DESARROLLO

2.3.1. GENERALIDADES DE OPENFLOW VERSIÓN 1.0

Se ha escogido el protocolo OpenFlow en su versión 1.0 dado su compatibilidad con todos los *switches* que soportan dicho protocolo, aunque presente una desventaja al observarse que la versión 1.3 de dicho protocolo la usan un número mayor de desarrolladores de aplicaciones para diversos controladores.

Los cambios que deberían realizarse en una aplicación desarrollada para la versión 1.3, para poder trabajar con la versión 1.0 principalmente se deben a la sintaxis empleada en algunas de las sentencias usadas en ambas versiones.

Primero se analizarán las particularidades de la versión 1.0 del protocolo OpenFlow [19] a mayor detalle que lo previamente realizado en el capítulo 1 del presente documento, con énfasis en el proceso de conexión del controlador y el *switch,* y a continuación se detallarán los mensajes producidos por el tráfico de paquetes, los eventos generados por la llegada de dichos mensajes [20] y su manejo en Ryu.

La implementación del protocolo OpenFlow utilizada por Ryu está constituida por estructuras de datos, que dan forma a los mensajes que utiliza. La cabecera del mensaje OpenFlow se muestra en la Figura 2.2.

```
1  /* Header on all OpenFlow packets. */
2  struct ofp_header {
3      uint8_t version; /* OFP_VERSION. */
4      uint8_t type; /* One of the OFPT_ constants. */
5      uint16_t length; /* Length including this ofp_header. */
6      uint32_t xid; /* Transaction id associated with this packet.
7                       Replies use the same id as was in the request
8                       to facilitate pairing. */
9  };
10 OFP_ASSERT(sizeof(struct ofp_header) == 8);
```

Figura 2.2. Cabecera de un mensaje OpenFlow

El valor `version` indica la versión del protocolo usada, para la versión 1.0 el valor es `0x01` mientras que el campo `length` indica la longitud total del mensaje para delimitar un mensaje del siguiente. En cuanto a `type` indica el tipo de mensaje que transporta y el campo `xid` corresponde a un identificador único de transacción que coincidirá entre un paquete de petición y su respuesta para permitir el emparejamiento adecuado.

2.3.2. PROCESO DE CONEXIÓN

En esta sección se presenta el proceso de conexión entre el controlador y el *switch*.

El primer paso en la comunicación inicial entre el *switch* y el controlador consiste en el envío de un mensaje de tipo `HELLO`. Este mensaje no tiene datos y solamente contiene una cabecera, es usado tanto por el *switch* como por el controlador para

identificar y negociar que versión del protocolo se usará comprobando que sea soportada por ambos. La cabecera del mensaje `HELLO` denominada `OFPT_HELLO` se indica en la Figura 2.3.

```
/* OFPT_HELLO. This message includes zero or more
   hello elements having variable size.  */
struct ofp_hello {
struct ofp_header header;
/* Hello element list */
struct ofp_hello_elem_header elements[0]; /* List of elements */
};
```

Figura 2.3. Cabecera del mensaje `HELLO`

La versión de OpenFlow se almacena en el campo `ofp_header` y se establece el valor más alto de las versiones soportadas por el *switch*. La Figura 2.4 muestra la estructura de un elemento de tipo `version_bitmap` usado para indicar la versión de OpenFlow propuesta por el controlador al *switch* siguiendo un formato de tipo TLV (tipo, longitud, valor). En el elemento `bitmap` se indica la versión soportada.

```
/* Version bitmap Hello Element */
struct ofp_hello_elem_versionbitmap {
uint16_t type;           /* OFPHET_VERSIONBITMAP. */
uint16_t length;         /* Length in bytes of this element. */
        /* Followed by:
         * - Exactly (length - 4) bytes containing the bitmaps,
         * then Exactly (length + 7)/8*8 - (length) (between 0
         * and 7) bytes of all-zero bytes */
uint32_t bitmaps[0]; /* List of bitmaps - supported versions */
};
```

Figura 2.4. Elemento `version_bitmap` de la cabecera `OFPT_HELLO`

El *switch* debe ser capaz de recibir y procesar los mensajes `HELLO` enviados desde el controlador, el cual usará el formato y estructura indicados. Una vez que el *switch* recibe este mensaje establece la versión a ser usada para el intercambio de mensajes con el controlador.

Cuando la versión indicada por el controlador es soportada por el *switch,* se continúa con el proceso de conexión entre ambos, caso contrario el *switch* envía un mensaje `ERROR` con el campo tipo establecido en `OFPET_HELLO_FAILED` y el campo código con el valor `OFPHFC_INCOMPATIBLE`.

En la sección 2.3.6.3 se presenta un ejemplo del manejo de los mensajes de tipo ERROR. A partir de este punto el proceso de conexión se completa y el *switch* puede intercambiar mensajes con el controlador y viceversa.

2.3.3. CONSULTA DE LAS CAPACIDADES DEL SWITCH

Si el controlador desea consultar los parámetros (también conocidos como capacidades) del *switch* emplea los mensajes de tipo `features`. Esta tarea también se conoce como *handshake* y consiste en el envío desde el controlador de un mensaje FEATURES_REQUEST construido con la estructura indicada en la Figura 2.5, a lo cual el *switch* responde con un mensaje FEATURES_REPLY. El campo `datapath_id` es un identificador único, en donde los 48 bits más bajos representan la dirección MAC del *switch*, mientras que los 16 bits más altos pueden ser usados para el identificador de VLAN que especifica una instancia del *switch*.

```
/* Switch features. */
struct ofp_switch_features {
    struct ofp_header header;
    uint64_t datapath_id; /* Datapath unique ID. The lower 48-bits are for
                             a MAC address, while the upper 16-bits are
                             implementer-defined. */
    uint32_t n_buffers; /* Max packets buffered at once. */
    uint8_t n_tables; /* Number of tables supported by datapath. */
    uint8_t pad[3]; /* Align to 64-bits. */
    /* Features. */
    uint32_t capabilities; /* Bitmap of support "ofp_capabilities". */
    uint32_t actions; /* Bitmap of supported "ofp_action_type"s. */
    /* Port info.*/
    struct ofp_phy_port ports[0]; /* Port definitions. The number of ports
                                     is inferred from the length field in
                                     the header. */
};
OFP_ASSERT(sizeof(struct ofp_switch_features) == 32);
```

Figura 2.5. Estructura de los mensajes FEATURES_REQUEST y FEATURES_REPLY

El campo `n_tables` permite indicar el número de tablas que soporta el *switch*.

El controlador sabrá cuántas tablas soporta el *switch* leyendo la información proporcionada en la respuesta FEATURES_REPLY. El campo `capabilities` posee una lista de banderas que representan parámetros propios del *switch*, los cuales se presentan en la Figura 2.6.

La Tabla 2.1 indica el uso de cada bandera, dependiendo del parámetro que se desee obtener se establece el número de identificador adecuado en el mensaje `FEATURES_REQUEST` para obtener como respuesta el parámetro deseado en un mensaje `FEATURES_REPLY`.

```
/* Capabilities supported by the datapath. */
enum ofp_capabilities {
    OFPC_FLOW_STATS = 1 << 0, /* Flow statistics. */
    OFPC_TABLE_STATS = 1 << 1, /* Table statistics. */
    OFPC_PORT_STATS = 1 << 2, /* Port statistics. */
    OFPC_STP = 1 << 3, /* 802.1d spanning tree. */
    OFPC_RESERVED = 1 << 4, /* Reserved, must be zero. */
    OFPC_IP_REASM = 1 << 5, /* Can reassemble IP fragments. */
    OFPC_QUEUE_STATS = 1 << 6, /* Queue statistics. */
    OFPC_ARP_MATCH_IP = 1 << 7 /* Match IP addresses in ARP pkts. */
};
```

Figura 2.6. Banderas del campo `capabilities`

BANDERA	VALOR	USO
OFPC_FLOW_STATS	0	Estadísticas del flujo
OFPC_TABLE_STATS	1	Estadísticas de las tablas
OFPC_PORT_STATS	2	Estadísticas del puerto
OFPC_STP	3	*Spanning Tree*[37] 802.1d
OFPC_RESERVED	4	Siempre valor cero
OFPC_IP_REASM	5	Reensamblar fragmentos IP
OFPC_QUEUE_STATS	6	Estadísticas de la cola
OFPC_ARP_MATCH_IP	7	Coincidencia de IP por ARP

Tabla 2.1. Valores de la bandera del campo `capabilities`

El campo `actions` enumera las acciones soportadas por el *switch* [19]. En la Tabla 2.2 se enumeran estas posibles acciones. Se debe recalcar que toda acción con la marca `required` es obligatoria para el *switch.* El campo `ports` corresponde a un arreglo estructurado que ofrece una descripción de los puertos físicos disponibles en el *switch* con soporte para OpenFlow.

[37] *Spanning Tree* es un protocolo de gestión de enlaces cuya función es evitar bucles.

CAMPO	BITS	PAQUETES ASOCIADOS	INFORMACIÓN ADICIONAL
Puerto de ingreso	(depende de la implementación)	Todos	Número que representa un puerto de entrada, inicia en 1
Dirección Ethernet fuente	48	Todos en puertos habilitados	
Dirección Ethernet destino	48	Todos en puertos habilitados	
Tipo Ethernet	16	Todos en puertos habilitados	Un *switch* OpenFlow debe coincidir en tipo tanto en 802.2 como en Ethernet
ID de VLAN	12	Todo paquete Ethernet con código 0x8100	
Prioridad de VLAN	3	Todo paquete Ethernet con código 0x8100	Utiliza el campo PCP[38] de la información de VLAN
IP fuente	32	Todo paquete IP y ARP	Puede contener máscara de subred
IP destino	32	Todo paquete IP y ARP	Puede contener máscara de subred
Protocolo IP	8	Todo paquete IP, IP sobre Ethernet y ARP	Usa los 8 bits más bajos de ARP
Bits ToS[39] de IP	6	Todo paquete IP	Usa los 6 bits más altos
Puerto fuente / Tipo ICMP	16	Todo paquete TCP, UDP e ICMP	Usa los 8 bits más bajos
Puerto destino / Código ICMP	16	Todo paquete TCP, UDP e ICMP	Usa los 8 bits más bajos

Tabla 2.2. Acciones OpenFlow soportadas por un *switch*

[38] PCP se utiliza para un codificado de la prioridad.
[39] (ToS) Tipo de Servicio es un campo usado para determinar prioridad de paquetes.

2.3.4. MENSAJE PACKET_IN

La Figura 2.5 muestra la estructura del mensaje `PACKET_IN`.

```
/* Packet received on port (datapath -> controller). */
struct ofp_packet_in {
    struct ofp_header header;
    uint32_t buffer_id; /* ID assigned by datapath. */
    uint16_t total_len; /* Full length of frame. */
    uint16_t in_port; /* Port on which frame was received. */
    uint8_t reason; /* Reason packet is being sent (one of OFPR_*) */
    uint8_t pad;
    uint8_t data[0]; /* Ethernet frame, halfway through 32-bit word,
                        so the IP header is 32-bit aligned. The
                        amount of data is inferred from the length
                        field in the header. Because of padding,
                        offsetof(struct ofp_packet_in, data) ==
                        sizeof(struct ofp_packet_in) - 2. */
};
OFP_ASSERT(sizeof(struct ofp_packet_in) == 20);
```

Figura 2.7. Estructura de `PACKET_IN`

Para el desarrollo del presente Proyecto y por la naturaleza de la tarea que cumplirá la aplicación, el mensaje que debe procesarse y analizarse es el mensaje asíncrono `PACKET_IN`. Este tipo de mensajes se generan cuando se recibe un paquete en un `datapath` (el *switch*) para los cuales no hay una regla asociada y entonces deben ser enviados al controlador para su manejo.

El campo `buffer_id` lo utiliza el *switch* para identificar si un paquete se encuentra almacenado en un *buffer*[40], de ser así se incluyen en el campo de datos bytes adicionales con información de dicho *buffer*; caso contrario al no encontrarse almacenado en un *buffer* no se modifica el campo de datos y el valor de `buffer_id` se establece en -1.

El campo `reason` permite indicar el motivo por el cual un paquete se envía al controlador, ya que el *switch* no supo qué acción realizar con dicho paquete. Este campo tiene dos posibles valores, `OFPR_NO_MATCH` para el caso en que no se encuentre una regla que determina la acción a realizarse con el paquete, o puede tener el valor `OFPR_ACTION` cuando sí existe una acción explícita. El campo `pad` consiste en relleno para cumplir con la longitud estandarizada del mensaje

[40] Un *buffer* es una cola de almacenamiento temporal de un paquete para evitar su pérdida en caso de congestión.

OpenFlow. Finalmente el campo `data` encapsula la trama Ethernet, donde además se incluye el relleno adecuado para cumplir con un alineamiento múltiplo de 32 bits.

2.3.5. MENSAJE PACKET_OUT

Corresponde a un mensaje enviado desde el controlador hacia el *switch* usado cada vez que el controlador desea enviar un mensaje a través de un `datapath` a un puerto en específico. Su estructura se observa en la Figura 2.8. Existen dos escenarios en los cuales el controlador puede generar este tipo de mensajes al *switch:*

- El primer caso es la construcción de todo un paquete que se desea enviar, para la cual el controlador añade el campo `action_list` en el mensaje que incluirá una lista de acciones para que el *switch* sepa como procesar el paquete, entonces el *switch* realizará las tareas indicadas y nuevamente se tendrá dos opciones: la modificación del paquete para luego ser enviado al puerto indicado o el procesamiento del paquetes basado en una tabla que se indique en la lista de acciones.
- El segundo caso se presenta si existe un *buffer* para paquetes en el *switch.* Si se usa este mecanismo el *switch* utiliza el *buffer* ya creado para el envío del mensaje `PACKET_IN` al controlador. Si coinciden los valores de `buffer_id` de los mensajes `PACKET_IN` y `PACKET_OUT` el *switch* busca el paquete dentro del *buffer* y lo envía a través del puerto adecuado. Una vez que el paquete se envía, la memoria asignada en el *buffer* es liberada.

```
1  /* Send packet (controller -> datapath). */
2  struct ofp_packet_out {
3      struct ofp_header header;
4      uint32_t buffer_id; /* ID assigned by datapath (-1 if none). */
5      uint16_t in_port; /* Packet's input port (OFPP_NONE if none). */
6      uint16_t actions_len; /* Size of action array in bytes. */
7      struct ofp_action_header actions[0]; /* Actions. */
8      /* uint8_t data[0]; */      /* Packet data. The length is inferred
9                                     from the length field in the header.
10                                    (Only meaningful if buffer_id == -1.) */
11 };
12 OFP_ASSERT(sizeof(struct ofp_packet_out) == 16);
```

Figura 2.8. Estructura del mensaje `PACKET_OUT`

2.3.6. EVENTOS EN RYU

Una parte fundamental para el desarrollo de la aplicación DHCP es el manejo de eventos producidos en el controlador producto de la llegada de mensajes OpenFlow para poder llevar a cabo las acciones necesarias para asignar parámetros DHCP a los clientes.

Se explicará los eventos encontrados en el proceso de conexión entre el *switch* y el controlador así como el evento que se produce cuando el controlador recibe un mensaje `PACKET_IN` mediante el uso de métodos capaces de manejar dichos eventos.

Adicionalmente se expone un método que maneja el evento producido por la llegada de mensajes de tipo `ERROR`.

Es fundamental conocer que para manejar un tipo particular de eventos se emplean decoradores, por lo cual cada uno de los eventos mencionados se manipulará mediante esta funcionalidad de Ryu.

El uso de un decorador[41] permita a una aplicación de Ryu registrar que se interesa por un tipo específico de evento y a su vez cumple la tarea de proveer el manejador adecuado [25].

2.3.6.1. Evento Switch Features

El método `switch_features_handler` indicado en el Código 2.5 se ejecutará en caso de que se produzca un evento de tipo `EventOFPSwitchFeatures` desde el respectivo `datapath`.

`Se` mostrará en el *log* de la aplicación que se ha recibido un mensaje de tipo Switch Features, e imprimirá los valores de `datapath`, *buffer*, número de tablas, un identificador y las capacidades del *switch* almacenadas en la enumeración `capabilities` detallada en la sección 2.3.3.

---

[41] Un decorador en Python corresponde a una función que extiende su funcionalidad al recibir como argumento otra función.

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    msg = ev.msg

    self.logger.debug('OFPSwitchFeatures received: '
                      'datapath_id=0x%016x n_buffers=%d '
                      'n_tables=%d auxiliary_id=%d '
                      'capabilities=0x%08x',
                      msg.datapath_id, msg.n_buffers, msg.n_tables,
                      msg.auxiliary_id, msg.capabilities)
```

Código 2.5. Método que maneja el evento `EventOFPSwitchFeatures`

2.3.6.2. Evento Packet In

Este evento se generará en el controlador a raíz de la llegada de un mensaje `PACKET_IN` y de igual manera será manejado por parte de la aplicación a través del método adecuado, un ejemplo del método llamado `packet_in_handler` para manejar este evento se indica en el Código 2.6.

El método `packet_in_handler` se encargará de mostrar en el *log* de la aplicación el estado del paquete recibido.

Dicho método utilizará el contenido del campo `reason`. Los valores posibles para este campo son `NO_MATCH`, `ACTION`, `INVALID TTL`, y `UNKNOWN` en caso de no coincidir con ninguno de los anteriores valores. Estos valores se explican en la sección 2.3.4.

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):

    msg = ev.msg
    dp = msg.datapath
    ofp = dp.ofproto
    if msg.reason == ofp.OFPR_NO_MATCH:
        reason = 'NO MATCH'
    elif msg.reason == ofp.OFPR_ACTION:
        reason = 'ACTION'
    elif msg.reason == ofp.OFPR_INVALID_TTL:
        reason = 'INVALID TTL'
    else:
        reason = 'unknown'
```

```
15
16      self.logger.debug('OFPPacketIn received: '
17                        'buffer_id=%x total_len=%d reason=%s '
18                        'table_id=%d cookie=%d match=%s data=%s',
19                        msg.buffer_id, msg.total_len, reason,
20                        msg.table_id, msg.cookie, msg.match,
21                        utils.hex_array(msg.data))
```

Código 2.6. Método que maneja el evento `EventOFPPacketIn`

2.3.6.3. Evento Error Message

Para tener información adicional de los mensajes que contienen errores en cualquiera de las fases de la ejecución del controlador se puede incluir en la aplicación el método `error_msg_handler` que maneje eventos originados por la recepción de un mensaje de tipo `ErrorMsg`, el cual se muestra en el Código 2.7.

```
1 @set_ev_cls(ofp_event.EventOFPErrorMsg,
2              [HANDSHAKE_DISPATCHER, CONFIG_DISPATCHER, MAIN_DISPATCHER])
3 def error_msg_handler(self, ev):
4     msg = ev.msg
5
6     self.logger.debug('OFPErrorMsg received: type=0x%02x code=0x%02x '
7                       'message=%s',
8                       msg.type, msg.code, utils.hex_array(msg.data))
```

Código 2.7. Método para el manejo del evento `EventOFPErrorMsg`

Se pasa como argumento del decorador `HANDSHAKE_DISPATCHER`, `CONFIG_DISPATCHER` y `MAIN_DISPATCHER` (que corresponden a etapas de la ejecución del controlador de conexión para el caso de las dos primeras y de ejecución en el caso de la tercera).

Esto se realiza para que se detecte un error generado en cualquiera de las etapas previamente enunciadas de funcionamiento del controlador, y a continuación se muestra en el *log* de la aplicación los campos del mensaje `type` y `code` para proporcionar información útil para encontrar soluciones a los problemas que puedan ocurrir.

2.4. DESARROLLO DEL SOFTWARE

2.4.1. MÓDULOS DE LA APLICACIÓN

Para la aplicación se definieron tres módulos con tareas específicas, los cuales se enumeran a continuación:

- Lógica DHCP (`dhcp.py`)
- Ensamblado de paquetes DHCP (`solicitudRespuesta.py`)
- Administrador de direcciones IP (`atiendeIp.py`)

2.4.1.1. Módulo de Lógica DHCP

En el caso del primer módulo, el controlador Ryu posee una librería que contiene la lógica del protocolo DHCP, acorde a lo indicado en el RFC 2131. Esta librería esta almacenada en `ryu/ryu/lib/packet/dhcp.py` y su código fuente se encuentra en [21].

En la etapa inicial de desarrollo se requirió un manejo de errores para el uso de esta librería, dichos errores posteriormente se corrigieron en las versiones más actuales del controlador Ryu (específicamente en la versión 3.26).

Los errores se presentaban si los paquetes recibidos no tenían cierta longitud mínima.

En la última versión de la librería `dhcp.py` [21] ya no se presentan estos fallos y siempre será recomendado ejecutar la versión más reciente del controlador para que no se susciten estos u otros fallos.

2.4.1.2. Módulo de Ensamblado de Paquetes DHCP

El segundo módulo se ocupa de las tareas de recepción, análisis, ensamblado y envío de paquetes necesarios en cada paso definido en las transacciones del protocolo DHCP.

En la Figura 2.9 se presenta un ejemplo de dichos pasos.

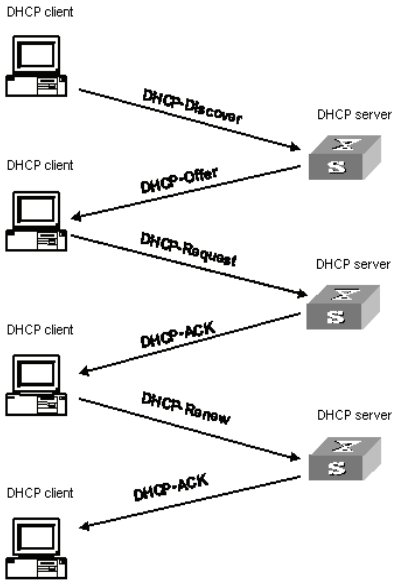

Figura 2.9. Paquetes que participan en una Solicitud DHCP [22]

2.4.1.3. Módulo de Asignación y Disponibilidad de Dirección IP

El tercer módulo realiza la asignación de la dirección IP para el cliente. En la aplicación inicialmente se establece la subred en la cual se realizara el préstamo de direcciones y este módulo concederá direcciones dentro de ese rango indicado.

Además este módulo realiza las exclusiones de direcciones reservadas (como por ejemplo la puerta de enlace o *gateway*, la dirección de difusión o *broadcast*, la dirección del servidor DHCP, entre otras), en donde se puede considerar incluir las direcciones de otros equipos que tienen una dirección IP fija.

2.4.2. SOFTWARE DE DESARROLLO

El IDE[42] escogido para el desarrollo de la aplicación se denomina PyCharm y su instalación se detalla en el Anexo B. Mediante PyCharm se puede importar todas las librerías propias de Ryu y, a excepción de eventos generados por el controlador, se puede trabajar con los métodos y clases propios de Ryu e instanciar fácilmente todas

[42] IDE (*Integrated Development Environment*), usado para proveer un ambiente gráfico de desarrollo con todas las herramientas necesarias integradas.

las librerías de dicho controlador y con esto facilitar el desarrollo. Los eventos del controlador aparecerán como un error dentro de PyCharm, lo cual no ocurre si se ejecuta la aplicación directamente con el *debugger* del controlador Ryu por lo cual no se recomienda ejecutar el código desde PyCharm.

2.4.3. DIAGRAMAS DE FLUJO

Las tareas que cumplirá la aplicación se indican en el diagrama de la Figura 2.10.

Cuando la aplicación `solicitudRespuesta.py` arranca ocurre el intercambio de mensajes `HELLO` para iniciar la conexión entre el controlador y el *switch* y negociar la versión del protocolo OpenFlow.

Una vez establecida la conexión el controlador recibe los paquetes que son enviados por el *switch* y posteriormente éstos son tratados por la aplicación, la cual hace una distinción para procesar únicamente paquetes correspondientes al protocolo DHCP.

La distinción mencionada se realiza mediante un análisis de las cabeceras de cada paquete y solamente si se trata de un paquete DHCP se continúa la ejecución del código, caso contrario se descarta dicho paquete. La aplicación al recibir un paquete de tipo DISCOVER responderá con el respectivo paquete OFFER, pero dicha oferta no contendrá aún los parámetros DHCP requeridos, solamente constituye una confirmación de que se negociará los parámetros DHCP con el servidor indicado (la aplicación prototipo).

Posteriormente el cliente genera un paquete REQUEST que se envía a la aplicación. Con la información recuperada del cliente desde el mensaje REQUEST (su dirección MAC, una dirección IP previa en caso de existir, el identificador de transacción, entre otros) se ensambla un paquete con todos los parámetros DHCP requeridos. El mensaje DHCP REQUEST puede contener una dirección IP previamente asignada o ningún valor de dirección en caso de tratarse de una asignación inicial. El paquete que se genera es de tipo ACK[43] y una vez realizado su envío finaliza la transacción.

---

[43] Las letras ACK son una contracción de la palabra *acknowledge* que en este contexto significa verificado o reconocido, es decir corresponde a una validación.

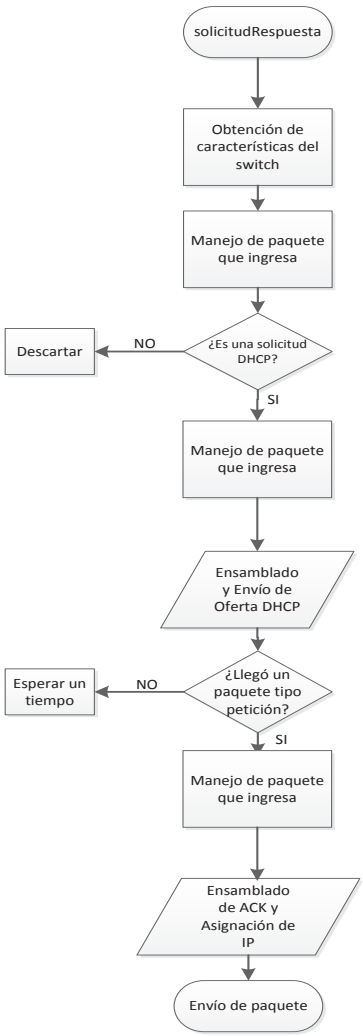

Figura 2.10. Diagrama de Flujo

Para el caso de renovación de una concesión de dirección IP los pasos de la transacción son los mismos que previamente se ha explicado, desde la llegada del paquete de tipo REQUEST hasta el ensamblado del paquete ACK que contiene la dirección renovada junto con los demás parámetros.

Los paquetes que ingresan al controlador (`DISCOVER` y `REQUEST`) están encapsulados en mensajes OpenFlow de tipo `PACKET_IN`, y el envío de paquetes de tipo `OFFER` y `ACK` se encapsulan en mensajees `PACKET_OUT`. Los parámetros que maneja la aplicación prototipo son: máscara de subred, tiempo de préstamo, dirección del servidor DHCP y dirección de servidor DNS, todos ellos se incluirán dentro del paquete `ACK`.

2.4.4. DIAGRAMA DE CLASES

En la Figura 2.11 se muestra el diagrama de clases de la aplicación.

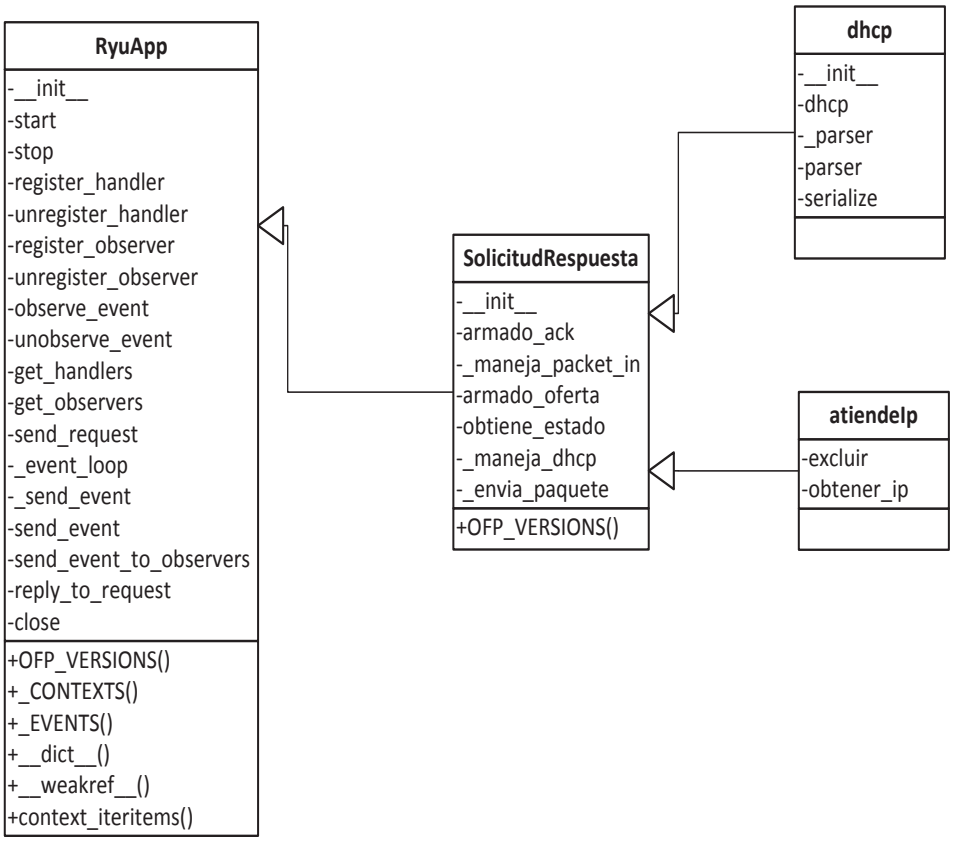

Figura 2.11. Diagrama de Clases

2.4.5. DIAGRAMAS DE SECUENCIA

La Figura 2.12 muestra el diagrama de secuencia [23] para el inicio de la conexión entre el *switch* y el controlador [24].

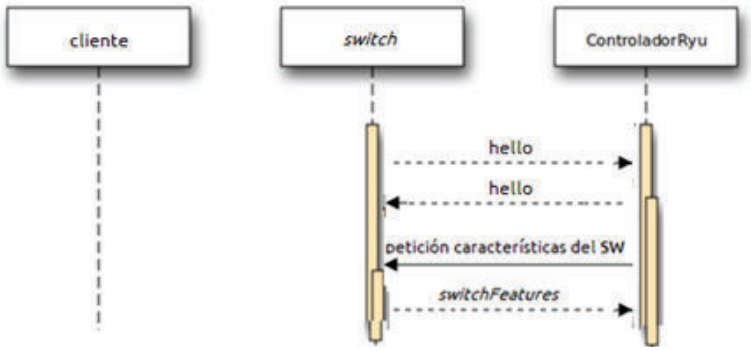

Figura 2.12. Diagrama de secuencia para la conexión entre *switch* y controlador

La Figura 2.13 indica el diagrama de secuencia para la asignación inicial de parámetros DHCP a un cliente solicitante.

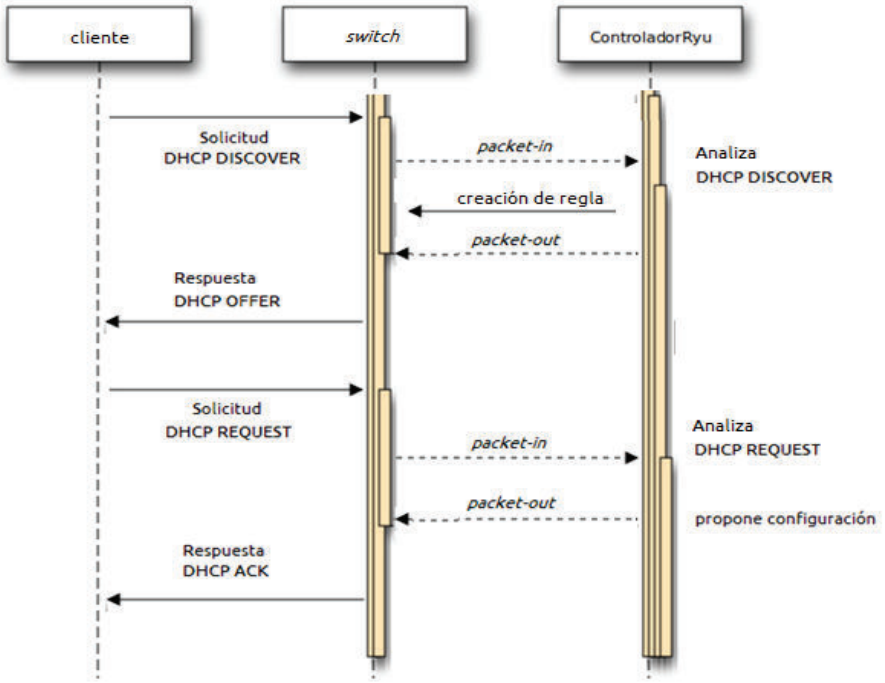

Figura 2.13. Diagrama de secuencia para la asignación de parámetros DHCP

La Figura 2.14 ilustra el diagrama de secuencia para la renovación de una dirección IP mediante la aplicación DHCP.

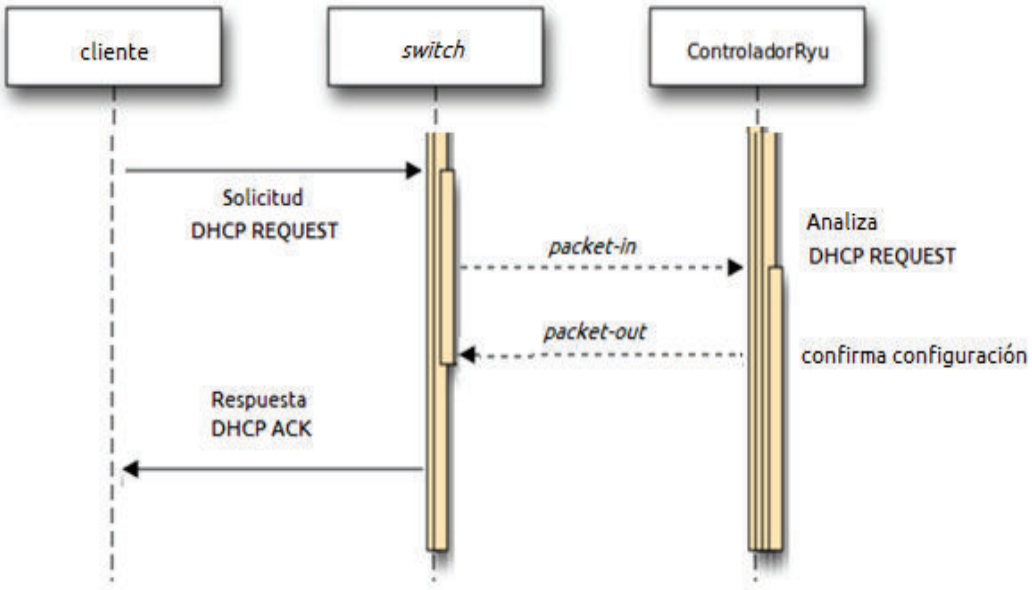

Figura 2.14. Diagrama de secuencia para la renovación de dirección IP

Las líneas entrecortadas en cada diagrama representan mensajes de tipo OpenFlow y son producto de la interacción entre el *switch* y el controlador, mientras que los paquetes UDP de la transacción DHCP se muestran con su respectivo nombre en la parte izquierda de cada diagrama.

2.4.6. EXPLICACIÓN DEL CÓDIGO DESARROLLADO

2.4.6.1. Componentes Utilizados

Inicialmente se brinda una descripción de los componentes utilizados en el desarrollo del módulo de ensamblado de paquetes DHCP:

- `ryu.base app_manager,` corresponde a la definición de una nueva subclase de RuyApp, es el punto inicial de todo *script* Python que funcionará como una aplicación de Ryu.
- `ryu.lib addrconv,` es un objeto que se encarga de la conversión de direcciones IP de texto a binario y viceversa.
- `ryu.controller ofp_event,` contiene todas las definiciones y distinción de los eventos OpenFlow; su principal función es la creación del evento OpenFlow adecuado según el mensaje OpenFlow que recibe como argumento.
- `ryu.controller.handler CONFIG_DISPATCHER,` cumple la tarea de manejador de eventos, y el evento por el cual escucha se produce con la recepción de un mensaje de tipo `SwitchFeatures.` Es utilizado al momento de iniciar la negociación que permite la comunicación entre el controlador y el *switch.*
- `ryu.controller.handler MAIN_DISPATCHER,` se trata de un manejador de eventos, que actúa persistentemente cuando el controlador se encuentra en un estado normal de funcionamiento.
- `ryu.controller.handler set_ev_cls,` encapsula características para actuar como un manejador de eventos, de suma importancia para la comunicación con el controlador.

- `ryu.ofproto ofproto_v1_0`, permite indicar la versión de OpenFlow utilizada en la aplicación.
- `ryu.lib.packet packet`, corresponde a un objeto que codifica y decodifica paquetes.
- `ryu.lib.packet ethernet`, es un objeto que maneja las cabeceras de los paquetes Ethernet.
- `ryu.lib.packet ipv4`, se trata de un objeto que maneja la codificación y decodificación de paquetes IP versión 4.
- `ryu.lib.packet udp`, corresponde a un objeto que se encarga de codificar y decodificar la cabecera de mensajes UDP.
- `ryu.lib.packet dhcp`, el objeto fundamental para la aplicación prototipo DHCP, contiene la lógica de codificación y decodificación de los campos referentes a dicho protocolo.

2.4.6.2. Código del módulo solicitudRespuesta

El módulo `solicitudRespuesta.py` se encarga de la recepción, interpretación, ensamblaje y envío de paquetes UDP que contienen información referente al protocolo DHCP. Una tarea necesaria para este propósito es la creación de flujos para envío y recepción de paquetes. Para identificar el tipo de mensaje DHCP recibido por la aplicación se usa de manera referencial la estructura de la aplicación desarrollada por Andy Hill encontrada en GitHub [25] con ciertas modificaciones que permite configurar los parámetros deseados.

En la línea 1 del Código 2.8 se instancia la clase `SolicitudRespuesta` heredada de `RyuApp`. Toda aplicación Ryu tendrá una línea como esta como punto de partida para que el intérprete del *framework* de Ryu sea capaz de procesar el *script*.

En la línea 2 del Código 2.8 se determina que la aplicación trabajará con la sintaxis y características de la versión 1.0 del protocolo OpenFlow.

```
1 class SolicitudRespuesta(app_manager.RyuApp):
2     OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION]
```

Código 2.8. Parte de la Clase `SolicitudRespuesta`

En la línea 1 del Código 2.9 se define el constructor, con los argumentos `self`, que corresponde a una instancia del mismo objeto, y los argumentos `*args` y `**kwargs` que definen que se puede tener un número indeterminado de argumentos adicionales, tanto de tipo dato único (*args) o un dato de tipo diccionario (**kwargs).

En el resto de líneas del Código 2.9 se establecen los atributos de la clase `SolicitudRespuesta`, correspondientes a los parámetros del protocolo DHCP como son dirección MAC del servidor, dirección IP del servidor, máscara de subred para los clientes, servidor DNS (*Domain Name Service*), tiempo de préstamo de la dirección, y conjunto de direcciones IP disponibles para el préstamo.

En cuanto al tiempo de préstamo se recalca que sus valores de tiempo admisibles están entre: 1 que equivale a un préstamo de 4 minutos 16 segundos y 255 equivalente a un préstamo de 18 horas 8 minutos.

```
1     def __init__(self, *args, **kwargs):
2         super(SolicitudRespuesta, self).__init__(*args, **kwargs)
3         self.red_ip = IPNetwork('192.168.0.1/24')
4         self.direcc_hw = '0a:e4:1c:d1:3e:44'
5         self.servidor_dhcp = IPAddress('192.168.0.10')
6         self.mascara_subred = IPAddress('255.255.0.0')
7         self.direccion_gateway = IPAddress('192.168.0.1')
8         self.dir_broadcast = IPAddress('192.168.0.255')
9         self.dns = IPAddress('8.8.8.8')
10        self.bin_dns = addrconv.ipv4.text_to_bin(self.dns)
11        self.hostname = 'manuel'
12        self.tiempo_arriendo = 1
13        self.bin_mascara_subred =
  addrconv.ipv4.text_to_bin(self.mascara_subred)
14        self.bin_gateway =
  addrconv.ipv4.text_to_bin(self.direccion_gateway)
15        self.bin_servidor =
  addrconv.ipv4.text_to_bin(self.servidor_dhcp)
16    self.pool_direcciones = list(self.red_ip)
17    self.pool_direcciones.reverse()
```

Código 2.9. Constructor de la Clase `SolicitudRespuesta`

La línea 1 del Código 2.10 define un manipulador de evento mediante un decorador que responde al evento OpenFlow generado al recibir un mensaje de tipo `PACKET_IN`.

La línea 2 en el Código 2.10 define un método llamado `_maneja_packet_in`. El motivo por el cual se ha ubicado un guion bajo previo al nombre del método obedece a las convenciones de Python, es una advertencia para futuros colaboradores del

código de no manipular el método sin tener un sólido conocimiento de lo que hace y los mecanismos que emplea para hacerlo.

De la línea 3 a la línea 6 del Código 2.10 se definen variables, con información de dirección MAC, puerto y el paquete que se está analizando, para su posterior uso.

En la línea 8 del Código 2.10 se declara la variable `paq_udp` que contiene la información de la cabecera UDP del paquete recibido que permite identificar si el paquete es de tipo DHCP, esta tarea se logra revisando si proviene del puerto 68.

Posteriormente desde la línea 9 hasta la línea 13 del Código 2.10 se detalla el condicional que permite comprobar si es un paquete de tipo DHCP. Estos paquetes que cumplan la condición serán manejados por el método `_maneja_dhcp`.

La línea 7 del Código 2.10, que se encuentra comentada, puede ser útil en caso de presentarse algún error en el índice de la lista, pues imprime el contenido del objeto `pkt`.

```
1  @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
2      def _maneja_packet_in(self, ev):
3          msg = ev.msg
4          datapath = msg.datapath
5          port = msg.match['in_port']
6          pkt = packet.Packet(data=msg.data)
7  # self.logger.info("packet-in %s" \n % (pkt,))
8          paq_udp = pkt.get_protocol(udp.udp)
9          if paq_udp and paq_udp.src_port == 68:
10             self._maneja_dhcp(datapath, port, pkt)
11         else:
12             return
13         return
```

Código 2.10. Método `_maneja_packet_in`

El Código 2.11 presenta el método `armado_ack`. Este método ensamblará un paquete de tipo DHCP ACK a partir de un paquete DHCP REQUEST. Se arma dicho paquete con sus cabeceras respectivas a través de las clases de `/ryu/ryu/lib/packet/` (Ethernet, IPV4 y UDP) y leyendo la información contenida en la cuarta posición de `pkt` que contiene la información de DHCP. En la línea 15 del Código 2.11 se cambia el valor de `tag`[44] a 05, que es el valor correspondiente a

[44] *Tag* es in valor usado en el protocolo DHCP para reconocer el tipo de mensaje, más información se encuentra en el primer capítulo del presente proyecto.

un paquete DHCP ACK. Desde la línea 7 hasta la línea 10 del Código 2.11 se crea un condicional para manejar renovaciones de direcciones. La condición identifica si el cliente en su campo `CIADDR` posee una dirección asignada en cuyo caso se le otorga la misma dirección, y en el caso que `CIADDR` tenga un valor 0.0.0.0 se sabrá que no existía una asignación previa y por lo tanto se llama al método adecuado para conceder una dirección IP.

Dentro de `paq_ack` se ubicarán todos los parámetros a configurarse mediante una lista de opciones llamada `options` la cual se observa en la línea 17 del Código 2.11. Los elementos presentes en esta lista en orden son: el identificador de tipo de paquete DHCP, el identificador del servidor DHCP (usado para identificar una dirección de un servidor DHCP opcional que se encuentre en la misma red), el tiempo de préstamo, la máscara de subred, el *gateway* y la dirección del servidor DNS. El paquete ensamblado se muestra en pantalla con la instrucción presentada en la línea 24 del Código 2.11.

```
1  def armado_ack(self, pkt):
2          req_eth = pkt.get_protocol(ethernet.ethernet)
3          req_ipv4 = pkt.get_protocol(ipv4.ipv4)
4          req_udp = pkt.get_protocol(udp.udp)
5          req = dhcp.dhcp.parser(pkt[3])
6          req[2].options.option_list.remove(next(opt for opt in
   req[2].options.option_list if opt.tag == 53))
7          if req_dhcp.ciaddr == '0.0.0.0'
8              direcc_ip = atiendeIp.obtener_ip(self.pool_direcciones)
9          else:
10             direcc_ip = req_dhcp.ciaddr
11         opcion_subred = dhcp.option(tag=dhcp.DHCP_SUBNET_MASK_OPT,
                                       value=self.bin_mascara_subred)
12         opcion_gw = dhcp.option(tag=dhcp.DHCP_GATEWAY_ADDR_OPT,
                                   value=self.bin_gateway)
13         opcion_dns = dhcp.option(tag=dhcp.DHCP_DNS_SERVER_ADDR_OPT,
                                    value=self.bin_dns)
14         opcion_tiempo =
   dhcp.option(tag=dhcp.DHCP_IP_ADDR_LEASE_TIME_OPT,
   value=hex(self.tiempo_arriendo))
15         opcion_msj = dhcp.option(tag=dhcp.DHCP_MESSAGE_TYPE_OPT,
                                    value='\x05')
16         opcion_id =
   dhcp.option(tag=dhcp.DHCP_SERVER_IDENTIFIER_OPT,
17                                  value=self.bin_servidor)
           options = dhcp.options(option_list=[opcion_msj, opcion_id,
18 opcion_tiempo, opcion_subred, opcion_gw, opcion_dns])
19         hlen = len(addrconv.mac.text_to_bin(self.direcc_hw))
20         paq_ack = packet.Packet()
```

```
paq_ack.add_protocol(ethernet.ethernet(ethertype=req_eth.ethertype,
21 dst=req_eth.src, src=self.direcc_hw))
        paq_ack.add_protocol(ipv4.ipv4(dst=req_ipv4.dst,
22 src=self.servidor_dhcp, proto=req_ipv4.proto))
        paq_ack.add_protocol(udp.udp(src_port=67, dst_port=68))
23      paq_ack.add_protocol(dhcp.dhcp(op=dhcp.DHCP_BOOT_REPLY,
hlen=hlen, chaddr=req_eth.src, yiaddr=direcc_ip,
        giaddr=self.direccion_gateway, xid=req[2].xid,
24 options=options))
25      self.logger.info("ACK ARMADO: %s \n" % paq_ack)
        return paq_ack
```

Código 2.11. Método `armado_ack`

En el Código 2.12 se presenta el método `armado_oferta`. Este método analiza los paquetes de tipo DHCP DISCOVER para generar el respectivo DHCP OFFER. De manera similar al código previamente explicado, los parámetros de la oferta (DHCP OFFER) se representan como opciones que luego se agrupan en la variable `option_list` observable en la línea 8 del Código 2.12 que a continuación se inserta dentro de la sección de datos correspondiente al protocolo DHCP, este paso se presenta en la parte final de la línea 15 del Código 2.12.

La única opción que se inserta en el paquete OFFER es el identificador de tipo de mensaje correspondiente (cuyo valor hexadecimal es `\x02`) establecido en la línea 7 del Código 2.12.

```
1     def armado_oferta(self, pkt):
2         disc_eth = pkt.get_protocol(ethernet.ethernet)
3         disc_ipv4 = pkt.get_protocol(ipv4.ipv4)
4         disc_udp = pkt.get_protocol(udp.udp)
5         disc = dhcp.dhcp.parser(pkt[3])
6
7         opcion_msj = dhcp.option(tag=dhcp.DHCP_MESSAGE_TYPE_OPT,
  value='\x02')
8         opciones = dhcp.options(option_list=[opcion_msj])
9         hlen = len(addrconv.mac.text_to_bin(self.direcc_hw))
10
11        paq_oferta = packet.Packet()
12        paq_oferta.add_protocol(ethernet.ethernet
                  (ethertype=disc_eth.ethertype, dst=disc_eth.src,
                   src=self.direcc_hw))
13        paq_oferta.add_protocol(ipv4.ipv4(dst=disc_ipv4.dst,
                   src=self.servidor_dhcp, proto=disc_ipv4.proto))
14        paq_oferta.add_protocol(udp.udp(src_port=67, dst_port=68))
15        paq_oferta.add_protocol(dhcp.dhcp(hlen=hlen,
                   op=dhcp.DHCP_BOOT_REPLY,
                   chaddr=disc_eth.src, yiaddr=disc[2].yiaddr,
```

```
                    giaddr=self.direccion_gateway, xid=disc[2].xid,
                    options=opciones))
        self.logger.info("OFERTA ARMADA: %s \n" % paq_oferta)
        return paq_oferta
```

Código 2.12. Método `armado_oferta`

El Código 2.13 modificado de [25] toma como argumento `paq_dhcp` que se obtiene del manejo del evento producido por la llegada del mensaje `PACKET_IN` y se encarga de identificar qué tipo de paquete se tiene mediante el análisis del valor del campo `tag`, encontrado dentro del elemento `options` perteneciente a la lista `paq_dhcp`.

```
def obtiene_estado(self, paq_dhcp):
      self.logger.info("paquete-dhcp %s \n" % (paq_dhcp,))
      estado_dhcp = ord([opt for opt in
paq_dhcp[2].options.option_list if opt.tag == 53][0].value)
        if estado_dhcp == 1:
            state = 'DHCPDISCOVER'
        elif estado_dhcp == 2:
            state = 'DHCPOFFER'
        elif estado_dhcp == 3:
            state = 'DHCPREQUEST'
        elif estado_dhcp == 5:
            state = 'DHCPACK'
        return state
```

Código 2.13. Método `obtiene_estado`

El método `_maneja_dhcp` que se presenta en el Código 2.14 y modificado a partir de [25] muestra en el log de la aplicación el tipo de paquete DHCP recibido y lo redirecciona al método adecuado según sea de tipo DISCOVER o REQUEST. Para esta tarea utiliza el resultado del método `estado_dhcp` pues solamente interesa atender los mensajes mencionados.

```
def _maneja_dhcp(self, datapath, port, pkt):
        paq_dhcp = dhcp.dhcp.parser(pkt[3])
        estado_dhcp = self.obtiene_estado(paq_dhcp)
        self.logger.info("NUEVO PAQUETE DHCP- %s -RECIBIDO: %s \n" %
(estado_dhcp, paq_dhcp))
        if estado_dhcp == 'DHCPDISCOVER':
            self._envia_paquete(datapath, port,
self.armado_oferta(pkt))
        elif estado_dhcp == 'DHCPREQUEST':
            self._envia_paquete(datapath, port, self.armado_ack(pkt))
        else:
            return
```

Código 2.14. Método `_maneja_dhcp`

Finalmente el método `_envia_paquete` indicado en el Código 2.15 es el encargado de serializar los resultados de los paquetes DHCP OFFER y DHCP ACK ensamblados previamente, generando como resultado un paquete UDP que encapsula al protocolo DHCP en la línea 8, y a su vez el paquete UDP es enviado al `datapath` encapsulado en un mensaje `PACKET-OUT` en la línea 9. Finalmente, en la línea 7 se define la acción de envío del paquete por el mismo puerto por el cual fue recibido.

```
1 def _envia_paquete(self, datapath, port, pkt):
2         ofproto = datapath.ofproto
3         parser = datapath.ofproto_parser
4         pkt.serialize()
5         self.logger.info("packet-out %s \n" % (pkt,))
6         data = pkt.data
7         actions = [parser.OFPActionOutput(port=port)]
8         out = parser.OFPPacketOut(datapath=datapath,
                                    buffer_id=ofproto.OFP_NO_BUFFER,
                                    in_port=ofproto.OFPP_CONTROLLER,
                                    actions=actions,
                                    data=data)
9         datapath.send_msg(out)
```

Código 2.15. Método `_envia_paquete`

2.4.6.3. Código del módulo manejaIp

El módulo `manejaIp.py` mostrado en el Código 2.16 reúne funcionalidad que permite, a partir de un rango de direcciones creado previamente, hacer las exclusiones de las direcciones reservadas o pertenecientes a servidores; y además realiza la tarea de escoger una dirección IP de este rango y retirarla del mismo.

Se recalca que el rango de direcciones mencionado tiene la estructura de una lista, para facilitar la tarea de extracción de una dirección, una dirección puede ser extraída sea porque se la excluye inicialmente o porque ha sido asignada a un cliente mediante DHCP.

La línea 1 del Código 2.16 permite el uso del módulo de manejo de redes IP propio de Ryu llamado `IPNetwork`. En las líneas 3 y 4 del Código 2.16 se instancian objetos de tipo `IPNetwork` y `list`, el primero de ellos es una tupla (por lo tanto inmutable) de todas las direcciones que pertenecen a la red pasada como argumento, el segundo

objeto es el resultado de la conversión de este conjunto de datos en una lista para poder realizar operaciones de inserción, modificación o eliminación de direcciones.

Desde la línea 7 hasta la línea 10 del Código 2.16 se define el método `excluir` que acepta como argumentos una dirección IP y un rango de direcciones (`pool`). La línea 9 comprueba si en efecto el dato ingresado es una dirección perteneciente a la lista de direcciones presentes en el rango, mientras que la línea 10 extrae esa dirección de la lista para no producir errores en los que un cliente use una dirección reservada.

La línea 13 hasta la línea 16 del Código 2.16 corresponde al método `obtener_ip`, que se encarga de tomar una dirección de la lista de direcciones disponibles. El método `pop()` en Python obtiene y remueve el elemento en la posición indicada de la lista. Cuando no se indica un índice, el método retorna el último elemento de la lista.

```
1  from netaddr import IPNetwork
2
3  red_ip = IPNetwork('192.168.0.1/24')
4  lista_rango_dir = list(red_ip)
5
6
7  def excluir(dir1, pool):
8
9      assert dir1 in pool
10     pool.remove(dir1)
11
12
13 def obtener_ip(pool):
14
15     direcc_ip = str(pool.pop())
16     return direcc_ip
```

Código 2.16. Módulo `atiendeIp.py`

Para poder utilizar el módulo creado es necesario que se encuentre en el directorio indicado en la Figura 2.15.

```
/usr/local/bin/python2.7/dist-packages/ryu/lib/netconf
```

Figura 2.15. Ubicación escogida para el módulo `atiendeIP.py`

CAPÍTULO III

3. PRUEBAS DE FUNCIONAMIENTO

3.1. PRUEBAS EN RED VIRTUAL

La primera evaluación de la aplicación prototipo DHCP se ha realizado en el mismo ambiente virtual que se utilizó en el desarrollo del código.

3.1.1. DISTRIBUCIÓN DE CLIENTES, SWITCH Y CONTROLADOR

En la Figura 3.1 se indica la ubicación de cada uno de los elementos virtuales dentro de un equipo (PC con Ubuntu) que se emplean para evaluar el funcionamiento de la aplicación prototipo DHCP sobre un switch virtual:

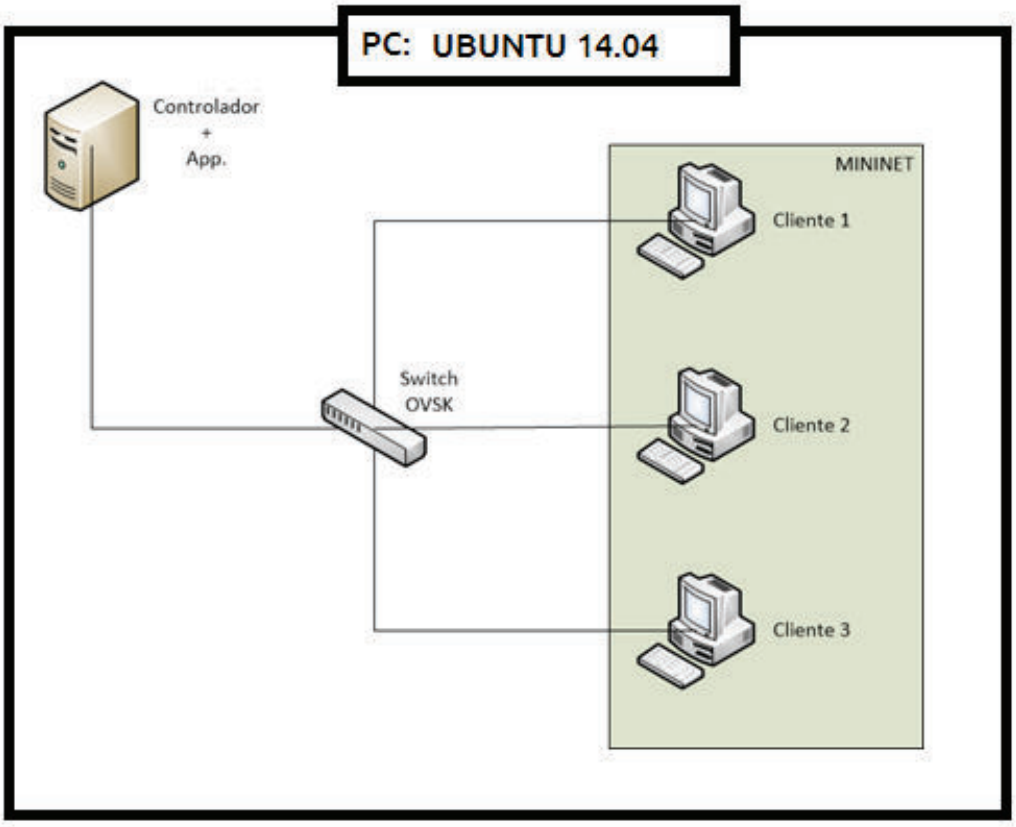

Figura 3.1. Topología de red virtual y sistemas donde residen los elementos para la aplicación DHCP

3.1.2. ARRANQUE DE ELEMENTOS DEL SISTEMA

Como primer paso se procede a iniciar el controlador Ryu. Cada vez que se desea correr una aplicación en Ryu se debe ejecutar el comando `ryu-manager` y a continuación se indica como argumento la ruta del *script* de la aplicación, como se observa en el Código 3.1. Si se desea instanciar varias aplicaciones Ryu para su ejecución se las enumera una a continuación de otra.

```
1 # cd /home/ubuntu/ryu && ./bin/ryu-manager --verbose
  ryu/app/simple_switch.py
```

Código 3.1. Comando para correr aplicaciones Ryu

Es importante indicar que se debe tener privilegios de usuario `root` para ejecutar estos comandos. La primera parte del Código 3.1 se emplea para llegar a la carpeta donde se encuentra instalado Ryu, posteriormente se llama al intérprete adecuado (`ryu-manager`).

El argumento `--verbose` es de gran utilidad para diversos comandos de Linux, se utiliza para obtener en pantalla información más detallada del funcionamiento de la aplicación y en el caso de estudio se usa para obtener detalles de la ejecución de la aplicación.

Finalmente se indica la ruta completa del archivo de Python que contiene la aplicación deseada, en este caso `simple_switch.py`.

3.1.3. FUNCIONAMIENTO DEL CONTROLADOR RYU Y MININET

Una vez iniciada la máquina virtual con Mininet y el controlador Ryu en Ubuntu, y habiendo configurado las direcciones necesarias para la interconectividad entre ambas máquinas se explicarán los pasos a seguirse para comprobar el funcionamiento de la aplicación prototipo DHCP.

Teniendo previamente a la aplicación Ryu en ejecución, se procede a ejecutar el comando indicado en la Figura 3.2 dentro de Mininet para crear una topología sencilla de red.

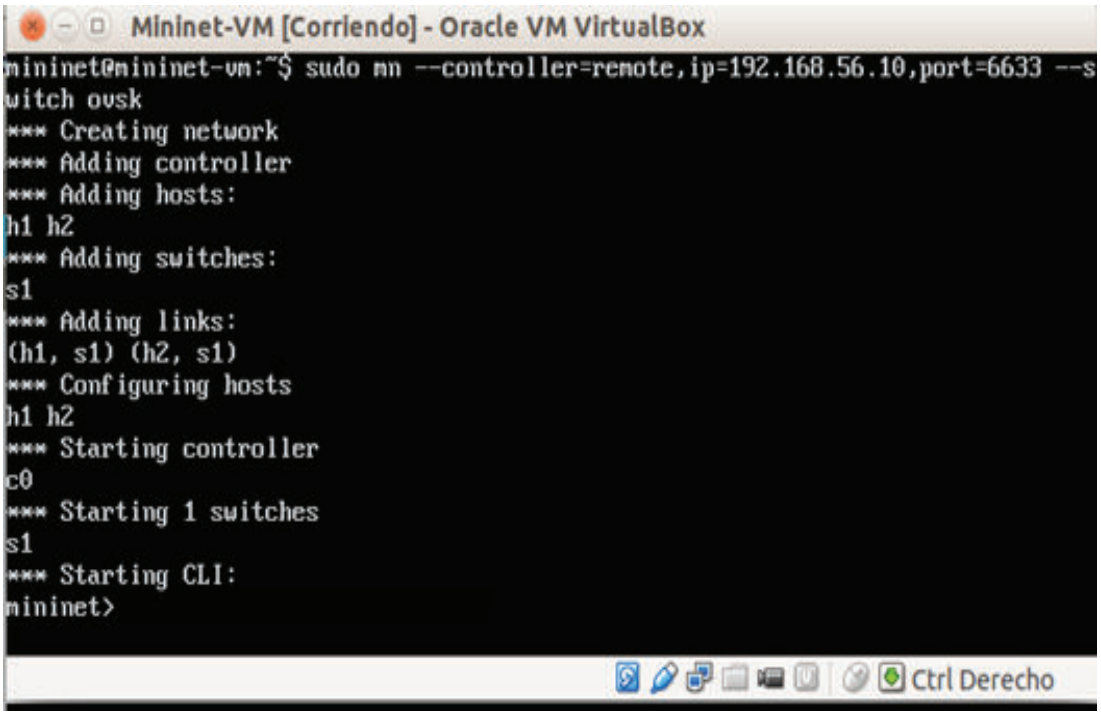

Figura 3.2. Creación de una topología sencilla en Mininet y asociación con el controlador Ryu

En el comando presentado en la Figura 3.2, la palabra `mn` se utiliza para ejecutar Mininet con una topología por defecto compuesta por dos clientes y un *switch* con sus respectivos enlaces.

En la línea de comando se le pasa dos argumentos a dicho comando que son `--controller` y `--switch`. El valor del primer argumento indica que el controlador funcionará en un sistema remoto por lo cual se indica tanto su dirección IP como el puerto donde escucha el controlador.

Generalmente el puerto 6633 es utilizado por el protocolo OpenFlow en sus diversas aplicaciones, así como el puerto 6653 en ciertas implementaciones que trabajan con dicho protocolo.

El valor `ovsk` del parámetro `--switch` indica que se empleará un switch de tipo software, en este caso Open vSwitch, el cual se encuentra incluido en la máquina virtual de Mininet.

Una vez ejecutado este comando en el lado del controlador se observará que se da el proceso de *handshake* y la comunicación inicia.

3.1.4. ANÁLISIS DE FUNCIONAMIENO DE LA APLICACIÓN DHCP

En esta sección se analizará la ejecución de la aplicación prototipo y también los mensajes que muestra la aplicación prototipo DHCP en el *framework* de Ryu, además de la interpretación y explicación de los eventos que los originan.

3.1.4.1. Realización de Pruebas en Mininet

Para poder iniciar una transacción DHCP desde Mininet se agrega un nuevo cliente, sin ninguna configuración inicial.

El Código 3.2 muestra los pasos a seguir para poder añadir un cliente a la topología creada en la sección 3.1.3.

```
1 mininet> py net.addHost('h3')
2 <Host h3: pid=3405>
3 mininet> py net.addLink(s1, net.get('h3'))
4 <Mininet.link.Link object at 0x1737090>
5 mininet> py s1.attach('s1-eth3')
```

Código 3.2. Comandos de Mininet para agregar un nuevo cliente

La primera línea del Código 3.2 se precede con la palabra `py` y se utiliza para añadir un nuevo cliente llamado `h3`.

El texto encerrado entre signos de mayor y menor <> que se encuentra en las líneas 2 y 4 del Código 3.3 corresponde a *outputs*[45] propios de Mininet.

En la línea 3 del Código 3.2 se crea el enlace entre el cliente y el *switch* y en la línea 5 se activa la interfaz de red que asocia al cliente h3 con el respectivo puerto del *switch*.

Finalmente para forzar el pedido de una configuración vía DHCP por parte del cliente recientemente agregado (que se trata de un cliente Linux), se ejecuta el comando indicado en el Código 3.3.

```
1 mininet> h3 dhclient h3-eth0
```

Código 3.3. Petición de configuración DHCP en el nuevo cliente

[45] Un *output* es un mensaje de salida, generalmente de salida en pantalla.

3.1.4.2. Resultados Observados en el Controlador

En el *debug* de la aplicación Ryu se observará la llegada y salida de cada uno de los mensajes de la transacción DHCP, el Código 3.4 muestra la salida de pantalla obtenida:

```
EVENT ofp_event->SolicitudRespuesta EventOFPPacketIn

 paquete-dhcp (None, None,
 dhcp(boot_file='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
 00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
 \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
 00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
 \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
 00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
 \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
 00\x00\x00',chaddr='fe:53:db:21:57:63',ciaddr='0.0.0.0',flags=0,giaddr='0.
 0.0.0',hlen=6,hops=0,htype=1,op=1,options=options(magic_cookie='99.130.83.
 99',option_list=[option(length=1,tag=53,value='\x01'),
 option(length=10,tag=12,value='mininet-vm'),
 option(length=13,tag=55,value='\x01\x1c\x02\x03\x0f\x06w\x0c,/\x1ay*')],op
 tions_len=64),secs=0,siaddr='0.0.0.0',sname=u'\x00\x00\x00\x00\x00\x00\x00
 \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
 00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
 \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
 00\x00',xid=272493670,yiaddr='0.0.0.0'))


NUEVO PAQUETE DHCP- DHCPDISCOVER -RECIBIDO: (None, None,
dhcp(boot_file='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00',chaddr='fe:53:db:21:57:63',ciaddr='0.0.0.0',flags=0,giaddr='0.
0.0.0',hlen=6,hops=0,htype=1,op=1,options=options(magic_cookie='99.130.83.
99',option_list=[option(length=1,tag=53,value='\x01'),
option(length=10,tag=12,value='mininet-vm'),
option(length=13,tag=55,value='\x01\x1c\x02\x03\x0f\x06w\x0c,/\x1ay*')],op
tions_len=64),secs=0,siaddr='0.0.0.0',sname=u'\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00',xid=272493670,yiaddr='0.0.0.0'))


OFERTA ARMADA:
ethernet(dst='fe:53:db:21:57:63',ethertype=2048,src='0a:e4:1c:d1:3e:44'),
ipv4(csum=0,dst='255.255.255.255',flags=0,header_length=5,identification=0
,offset=0,option=None,proto=17,src='192.168.0.1',tos=0,total_length=0,ttl=
255,version=4), udp(csum=0,dst_port=68,src_port=67,total_length=0),
dhcp(boot_file='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
```

```
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00',chaddr='fe:53:db:21:57:63',ciaddr='0.0.0.0',flags=0,giaddr='0.
0.0.0',hlen=6,hops=0,htype=1,op=2,options=options(magic_cookie='99.130.83.
99',option_list=[option(length=0,tag=54,value='\xc0\xa8\x00\x01'),
option(length=0,tag=53,value='\x02'),
option(length=0,tag=51,value='5000'),
option(length=0,tag=12,value='manuel'),
option(length=0,tag=6,value='\x08\x08\x08\x08'),
option(length=0,tag=3,value='\xc0\xa8\x00\x01'),
option(length=0,tag=1,value='\xff\xff\xff\x00')],options_len=64),secs=0,si
addr='192.168.0.1',sname='',xid=272493670,yiaddr='192.168.0.10')

packet-out
ethernet(dst='fe:53:db:21:57:63',ethertype=2048,src='0a:e4:1c:d1:3e:44'),
ipv4(csum=63995,dst='255.255.255.255',flags=0,header_length=5,identificati
on=0,offset=0,option=None,proto=17,src='192.168.0.1',tos=0,total_length=32
8,ttl=255,version=4),
udp(csum=11048,dst_port=68,src_port=67,total_length=308),
dhcp(boot_file='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00',chaddr='fe:53:db:21:57:63',ciaddr='0.0.0.0',flags=0,giaddr='0.
0.0.0',hlen=6,hops=0,htype=1,op=2,options=options(magic_cookie='99.130.83.
99',option_list=[option(length=4,tag=54,value='\xc0\xa8\x00\x01'),
option(length=1,tag=53,value='\x02'),
option(length=4,tag=51,value='5000'),
option(length=6,tag=12,value='manuel'),
option(length=4,tag=6,value='\x08\x08\x08\x08'),
option(length=4,tag=3,value='\xc0\xa8\x00\x01'),
option(length=4,tag=1,value='\xff\xff\xff\x00')],options_len=64),secs=0,si
addr='192.168.0.1',sname='',xid=272493670,yiaddr='192.168.0.10')

EVENT ofp_event->SolicitudRespuesta EventOFPPacketIn
paquete-dhcp (None, None,
dhcp(boot_file='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00',chaddr='fe:53:db:21:57:63',ciaddr='0.0.0.0',flags=0,giaddr='0.
0.0.0',hlen=6,hops=0,htype=1,op=1,options=options(magic_cookie='99.130.83.
99',option_list=[option(length=1,tag=53,value='\x03'),
option(length=4,tag=54,value='\xc0\xa8\x00\x01'),
option(length=4,tag=50,value='\xc0\xa8\x00\n'),
```

```
option(length=10,tag=12,value='mininet-vm'),
option(length=13,tag=55,value='\x01\x1c\x02\x03\x0f\x06w\x0c,/\x1ay*')],op
tions_len=64),secs=0,siaddr='0.0.0.0',sname=u'\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00',xid=272493670,yiaddr='0.0.0.0'))


NUEVO PAQUETE DHCP- DHCPREQUEST -RECIBIDO: (None, None,
dhcp(boot_file='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00',chaddr='fe:53:db:21:57:63',ciaddr='0.0.0.0',flags=0,giaddr='0.
0.0.0',hlen=6,hops=0,htype=1,op=1,options=options(magic_cookie='99.130.83.
99',option_list=[option(length=1,tag=53,value='\x03'),
option(length=4,tag=54,value='\xc0\xa8\x00\x01'),
option(length=4,tag=50,value='\xc0\xa8\x00\n'),
option(length=10,tag=12,value='mininet-vm'),
option(length=13,tag=55,value='\x01\x1c\x02\x03\x0f\x06w\x0c,/\x1ay*')],op
tions_len=64),secs=0,siaddr='0.0.0.0',sname=u'\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00',xid=272493670,yiaddr='0.0.0.0'))


ACK ARMADO:
ethernet(dst='fe:53:db:21:57:63',ethertype=2048,src='0a:e4:1c:d1:3e:44'),
ipv4(csum=0,dst='255.255.255.255',flags=0,header_length=5,identification=0
,offset=0,option=None,proto=17,src='192.168.0.1',tos=0,total_length=0,ttl=
255,version=4), udp(csum=0,dst_port=68,src_port=67,total_length=0),
dhcp(boot_file='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00',chaddr='fe:53:db:21:57:63',ciaddr='0.0.0.0',flags=0,giaddr='19
2.168.0.200',hlen=6,hops=0,htype=1,op=2,options=options(magic_cookie='99.1
30.83.99',option_list=[option(length=0,tag=53,value='\x05'),
option(length=0,tag=51,value='8640'),
option(length=4,tag=54,value='\xc0\xa8\x00\x01'),
option(length=4,tag=50,value='\xc0\xa8\x00\n'),
option(length=10,tag=12,value='mininet-vm'),
option(length=13,tag=55,value='\x01\x1c\x02\x03\x0f\x06w\x0c,/\x1ay*')],op
tions_len=64),secs=0,siaddr='192.168.0.1',sname='',xid=272493670,yiaddr='1
92.168.0.10')

packet-out
```

```
ethernet(dst='fe:53:db:21:57:63',ethertype=2048,src='0a:e4:1c:d1:3e:44'),
ipv4(csum=63995,dst='255.255.255.255',flags=0,header_length=5,identificati
on=0,offset=0,option=None,proto=17,src='192.168.0.1',tos=0,total_length=32
8,ttl=255,version=4),
udp(csum=409,dst_port=68,src_port=67,total_length=308),
dhcp(boot_file='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00',chaddr='fe:53:db:21:57:63',ciaddr='0.0.0.0',flags=0,giaddr='19
2.168.0.200',hlen=6,hops=0,htype=1,op=2,options=options(magic_cookie='99.1
30.83.99',option_list=[option(length=1,tag=53,value='\x05'),

option(length=4,tag=51,value='8640'),
option(length=4,tag=54,value='\xc0\xa8\x00\x01'),
option(length=4,tag=50,value='\xc0\xa8\x00\n'),
option(length=10,tag=12,value='mininet-vm'),
option(length=13,tag=55,value='\x01\x1c\x02\x03\x0f\x06w\x0c,/\x1ay*')],op
tions_len=64),secs=0,siaddr='192.168.0.1',sname='',xid=272493670,yiaddr='1
92.168.0.10')
```

Código 3.4. Paquetes que participan en una transacción DHCP completa

Se puede observar que cada una de las tuplas[46], con su respectivo tipo y valor, representa secuencialmente cada una de las cabeceras del paquete capturado (en el caso de estudio `ethernet`, `ipv4`, `udp` y `dhcp`).

Además se puede apreciar que dichos datos se encuentran ordenados tal como se ubican en el paquete original, este hecho será notorio en el análisis de los paquetes en Wireshark presentado más adelante.

La salida de pantalla expuesta muestra cómo se dio un nombre distintivo a cada uno de los paquetes que participan en la transacción para facilitar la identificación de cada etapa de la negociación DHCP.

Para el caso de paquetes entrantes previo a mostrarse el contenido del paquete que llega el controlador el *debug* indica la generación del evento `EventOFPPacketIn` asociado a la llegada de un paquete, como se observa en la línea 1 del Código 3.4.

---

[46] Una tupla es un conjunto inmutable de datos, al contrario de una lista que sí admite modificaciones en sus elementos.

3.1.5. ANÁLISIS DE PAQUETES EN WIRESHARK

3.1.5.1. Paquetes del Protocolo TCP / IP

Al analizar los paquetes con Wireshark, se aprecia que los mensajes generados por la aplicación guardan coherencia con lo que los clientes reciben e interpretan. Para el caso de estudio se analizará la información de los paquetes REQUEST y ACK de la transacción DHCP.

En el caso de un paquete REQUEST, el cual se observa en la Figura 3.3 se aprecia que el parámetro `CIADDRESS` se encuentra en 0.0.0.0, que pudiera contener una dirección IP en el caso que el cliente estuviera solicitando una renovación, por lo tanto el cliente mostrado solicita una dirección IP por primera vez. Además se puede resaltar que el campo `MESSAGE TYPE` contiene el identificador adecuado para este tipo de paquete (el valor 1 asociado a DHCP REQUEST).

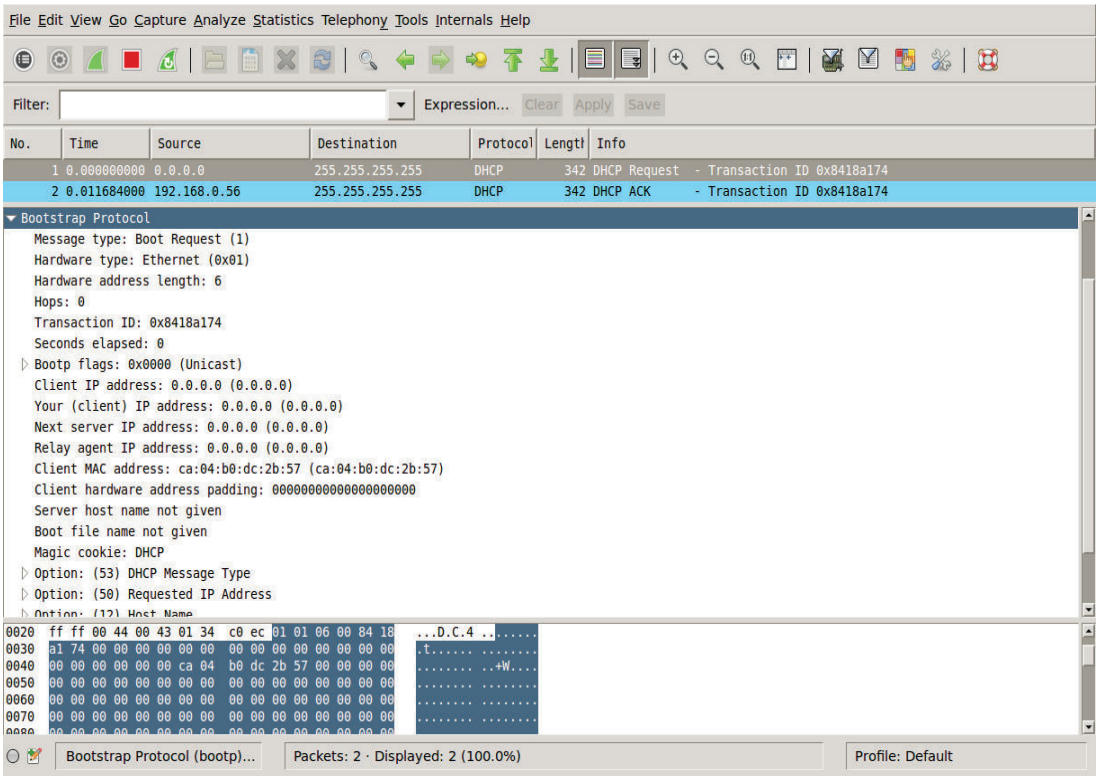

Figura 3.3. Paquete DHCP REQUEST en Wireshark

En la Figura 3.4 se tiene la captura del paquete de tipo ACK que es una respuesta correspondiente al paquete REQUEST previamente indicado, esto se ratifica con el

valor de `Transaction ID` que coincide para ambos casos, el campo `YIADDR` contiene el valor de dirección IP que será asignado al cliente.

El paquete DHCP ACK tiene una importancia muy grande en la transacción DHCP a partir de las observaciones realizadas en las pruebas de funcionamiento, pues en el caso de renovar la configuración de un cliente se observa que el mensaje generado no es un mensaje de tipo DISCOVER sino de tipo REQUEST, la lógica de la aplicación está pensada para contener todos los parámetros requeridos en el paquete ACK. SE observa que el campo `MESSAGE TYPE` contiene el identificador adecuado para este tipo de paquete, el valor 2 llamado BOOT REPLY que para el protocolo DHCP se asocia a un paquete de tipo ACK.

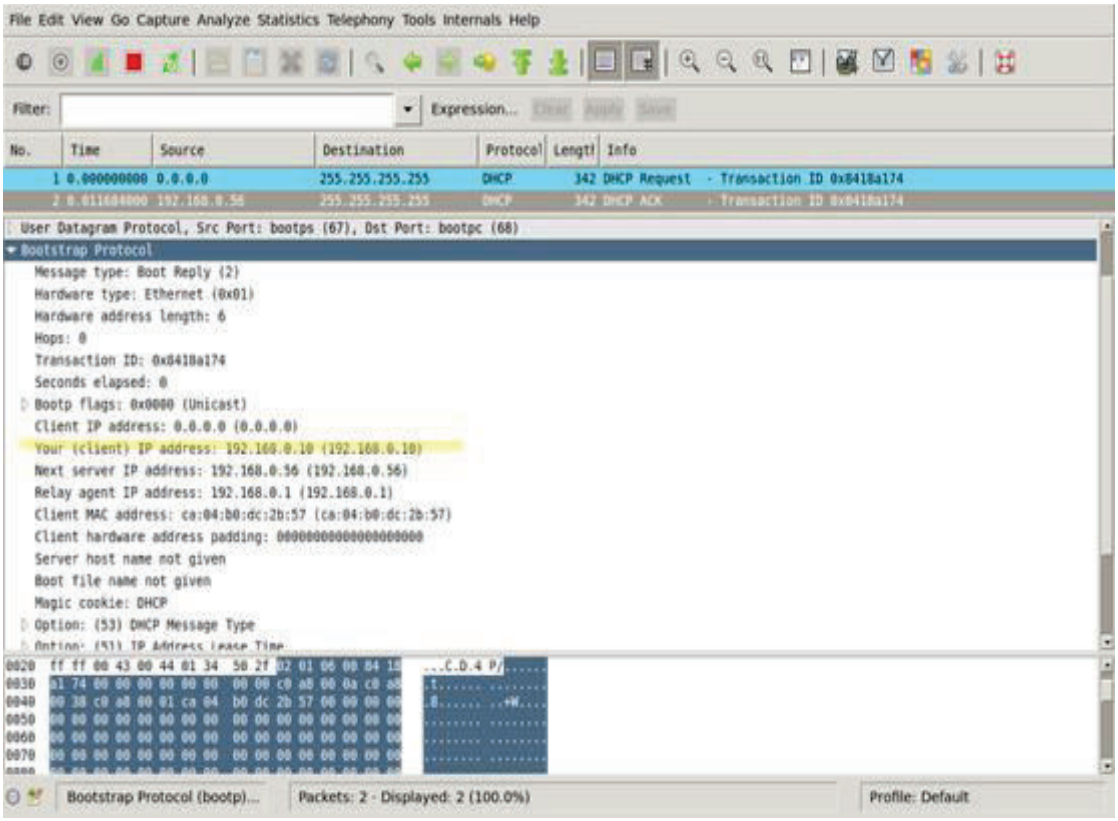

Figura 3.4. Paquete DHCP ACK en Wireshark

3.1.5.2. Mensajes del Protocolo OpenFlow

A continuación se hace un análisis de lo ocurrido desde la perspectiva de la arquitectura SDN, para lo cual se analizan los mensajes OpenFlow generados en la transacción DHCP.

Para capturar mensajes del protocolo OpenFlow es necesario añadir a Wireshark la funcionalidad de captura de mensajes OpenFlow, lo cual se encuentra explicado en el Anexo D.

También es posible encontrar paquetes de Ubuntu de versiones de Wireshark más recientes [26] para lograr analizar tráfico de este protocolo.

En el filtro de Wireshark se debe escribir `openflow_v1,` ya que de este modo se observará solamente los mensajes de la versión 1.0 del protocolo OpenFlow. En la Figura 3.5 se puede identificar un mensaje de tipo `packet_in`, en donde la dirección de origen corresponde al *switch* y la dirección destino es la del controlador Ryu.

En la información del protocolo OpenFlow se observa la versión del protocolo, la longitud del mensaje y un identificador de transacción.

En el campo **`reason`** se observa que no existe una regla para este paquete al tener el valor `OFPR_NO_MATCH`, la misma situación se dará para todo paquete que ingrese al controlador a través del *switch* ya que es de interés de la aplicación analizar todos los paquetes entrantes. El mensaje de la Figura 3.5 es de tipo `PACKET_IN`.

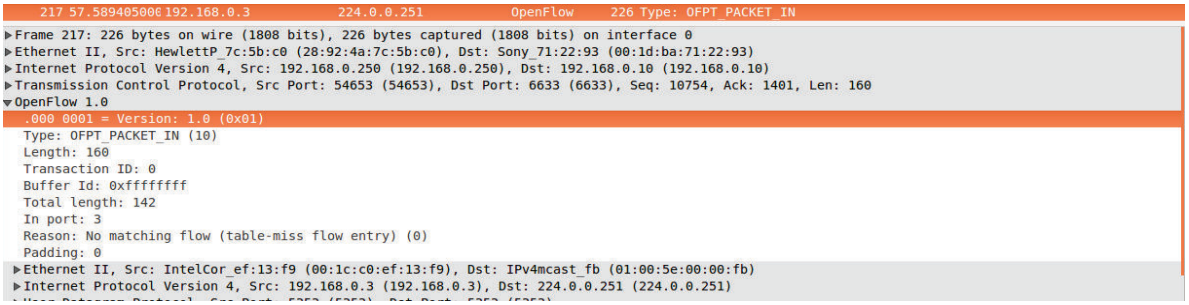

Figura 3.5. Mensaje `packet_in` en Wireshark

En la Figura 3.6 se muestra un mensaje de tipo `PACKET_OUT`, en los parámetros de OpenFlow se resalta los atributos versión, longitud e identificador de transacción.

Se puede destacar el atributo `IN_PORT` que indica que el mensaje se origina en el puerto del controlador y la acción indica que el mensaje saldrá por el puerto 3 (`OUTPUT PORT`) del *switch*, dicha información se detalla en el parámetro `Actions type` y obedece a la acción indicada por la aplicación para los paquetes salientes observada en el Código 2.15 del método `_envia_paquete`.

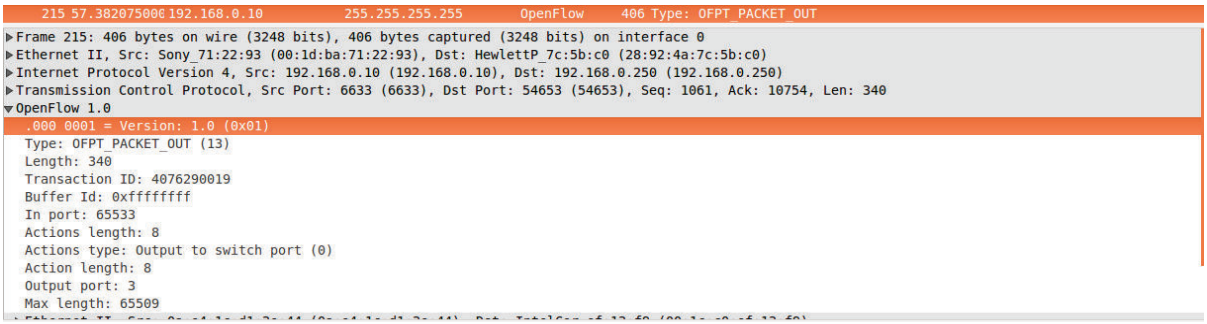

Figura 3.6. Mensaje `packet_out` en Wireshark

3.1.6. REGLAS GENERADAS EN OPEN VSWITCH

Para observar la creación de flujos en el *switch* virtual se ejecuta el comando indicado en el Código 3.5 con privilegios de usuario `root` dentro de la máquina virtual de Mininet.

```
1 # ovs-appctl -t ovs-vswitchd bridge/dump-flowsm s1
```

Código 3.5. Comando para mostrar los flujos del *switch* `s1`

El resultado de la ejecución de este comando una vez realizada la conexión inicial se muestra en la Figura 3.7.

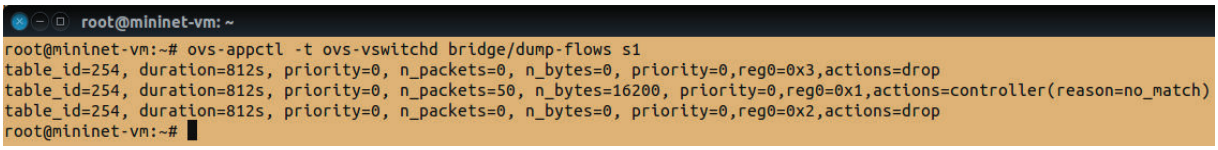

Figura 3.7. Flujos cuando inicia la conexión

Cuando el cliente ha solicitado un préstamo de dirección IP mediante DHCP y ha recibido dicha concesión se aprecia la creación de un flujo mostrado en la Figura 3.8.

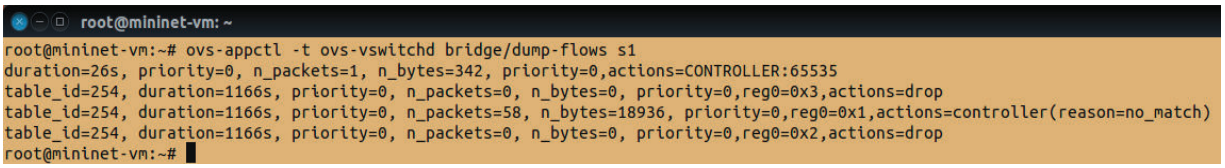

Figura 3.8. Flujos existentes con la presencia de un cliente que solicita parámetros DHCP

El flujo creado para el tráfico de paquetes del cliente solicitante también se puede observar mediante la ejecución del Código 3.6.

```
1 # ovs-ofctl dump-flows s1
```

Código 3.6. Comando para mostrar los flujos en `s1`

El resultado de la ejecución del Código 3.6 se muestra en la Figura 3.9, donde se muestra solamente flujos creados por la existencia de una acción explícita a realizarse.

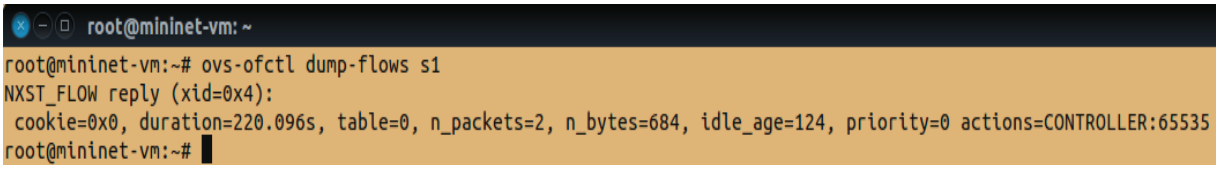

Figura 3.9. Flujo correspondiente al cliente solicitante con acciones explícitas

El número de flujos que se observen al ejecutar el comando del Código 3.6 equivale al número de clientes conectados a Open vSwitch a los cuales se les ha concedido un préstamo de dirección IP, que para el caso expuesto fue un cliente.

3.2. PRUEBAS CON RED FÍSICA

Para las pruebas definitivas del funcionamiento de la aplicación prototipo DHCP se emplea una red física conformada por un switch físico, un primer equipo que contiene el controlador y tres equipos correspondientes a clientes DHCP.

3.2.1. SWITCH HP PROCURVE 3500

3.2.1.1. Características del Switch

El switch utilizado para realizar las pruebas es un HP ProCurve modelo 3500 24 [27]. Corresponde a un modelo con soporte nativo para la arquitectura SDN, posee 24 puertos y funcionalidad de capa 3 de alto desempeño con posibilidad de expansión modular para enlaces de subida de tipo 10Gb Ethernet [22] y posee soporte de OpenFlow tanto en su versión 1.0 como en versión 1.3.

3.2.1.2. Configuración del Switch

La configuración necesaria para que el switch pueda trabajar adecuadamente con la aplicación DHCP consiste en tres pasos: la configuración de una VLAN, la configuración del controlador y la configuración de una instancia OpenFlow. Previo a

la realización de estos tres pasos es necesario conectarse a la interfaz de comandos del *switch*, el procedimiento para Linux se indica en el Código 3.7.

```
1 sudo apt-get install ckermit
2 sudo kermit -l /dev/ttyUSB0
3 set speed 9600
4 set carrier-watch off
5 set flow-control none
6 set parity none
7 connect
```

Código 3.7. Conexión con el Switch

En la Línea 1 del Código 3.7 se instala el programa C-Kermit [28] que permite la conexión al *switch* mediante el uso de un adaptador USB a serial en el sistema operativo Ubuntu. La línea 2 del Código 3.7 inicia el establecimiento de parámetros para la conexión al switch, y una vez ejecutado se configura cada uno de los parámetros. Desde la línea 3 hasta la línea 6 del Código 3.7 se indica la velocidad de conexión, se desactiva la detección de portadora dado que esta señal se enciende después de finalizado el marcado de la conexión, posteriormente se desactiva el control de flujo dado que el programa se encarga de manejar este parámetro de manera automática para evitar errores, por el mismo motivo luego se desactiva el uso de paridad. Una vez configurados los parámetros en la línea 8 del Código 3.7 se establece la conexión con el *switch*. En caso de no conocer la dirección IP del switch, una vez establecida la conexión con el *switch* vía puerto serial con el procedimiento anteriormente indicado, se puede ejecutar el comando `show ip` para obtenerla.

Una vez conocida la dirección IP del switch es posible conectarse al mismo mediante `ssh` o `telnet`[47]. Para configurar la VLAN, una vez dentro del CLI del switch se ingresa el comando `config` para ingresar al modo de configuración. Para mostrar las VLAN definidas se ejecuta el comando `show vlan` y para mostrar la configuración y puertos de una VLAN se ejecuta el mismo comando acompañado de la ID de la VLAN requerida. Es necesaria la creación de una nueva VLAN [29] que se utilizará

[47] Tanto `ssh` como `telnet` son protocolos para acceso remoto a un sistema, siendo `ssh` más seguro que `telnet` por utilizar cifrado de información.

para trabajar con la aplicación DHCP desarrollada, para crear una VLAN nueva se ejecuta el comando presentado en el Código 3.8:

```
vlan <id_de_vlan> name openflow10
```

Código 3.8. Creación de una VLAN

Al ejecutar el Código 3.8 se debe evitar el uso de un número identificador de VLAN preexistente para no borrar la VLAN previa y además se sugiere asignar un nombre descriptivo a la VLAN.

Una vez creada la VLAN (para este caso llamada openflow10) se debe indicar los puertos pertenecientes a la misma, esto se logra ingresando el comando indicado en el Código 3.9.

```
vlan <vlan_id> untagged <ports>
```

Código 3.9. Asignación de puertos a la VLAN creada

El segundo parámetro a configurarse es la información del controlador, para lo cual se debe crear una VLAN explícita para esta tarea, ya que la VLAN del controlador no puede ser la misma de la instancia OpenFlow, alternativamente se puede mantener al controlador en la VLAN por defecto (VLAN 1) para no crear una nueva VLAN.

El Código 3.10 especifica la dirección IP del controlador, el identificador de la VLAN donde se encuentra el controlador y un identificador para el controlador cuyo valor puede establecerse entre 1 y 256.

```
controller-id 1 ip 192.168.0.10 controller-interface vlan 1
```

Código 3.10. Configuración de parámetros del controlador en el *switch*

El tercer parámetro a establecerse es la instancia OpenFlow, se entiende como instancia a un conjunto de valores referentes al protocolo OpenFlow que se encuentran agrupados y pueden ser apuntados hacia un controlador para su correcto funcionamiento, un ejemplo de valor configurable en una instancia es la versión del protocolo OpenFlow.

Para mostrar las instancias OpenFlow existentes se emplea el comando `show openflow`, una vez conocida esta información se ingresa a la VLAN que se desea configurar con el comando `vlan <id_de_vlan>` y se verifica la configuración

OpenFlow que tiene actualmente mediante el comando `show openflow <id_de_vlan>`. Una vez dentro de la instancia OpenFlow creada, para asociarla al controlador se ingresa a la instancia y se ejecutan los comandos indicados en el Código 3.11. Los valores a establecerse son el puerto donde escucha el controlador, la VLAN donde se encuentran los puertos donde se ubicarán los clientes y el identificador del controlador a usarse.

```
1 instance prototipo
2 listen-port 6633
3 member vlan <id_de_vlan>
4 controller-id 1
```

Código 3.11. Configuración de la instancia OpenFlow

Una vez configurada la instancia con sus respectivos parámetros y permaneciendo dentro de la instancia se habilita la misma ejecutando las instrucciones mostradas en el Código 3.12. El *switch* ahora se encuentra preparado para la conexión con el controlador.

```
1 enable
2 exit
```

Código 3.12. Habilitación de la instancia OpenFlow

3.2.2. RESULTADOS OBTENIDOS

Para realizar la petición DHCP se ejecutan en cada uno de los clientes Linux las instrucciones indicadas en el Código 3.13, la línea 1 del Código 3.13 borra las configuraciones previas de dirección IP que pueden existir en el equipo, mientras que la línea 2 del Código 3.13 fuerza una petición de tipo DHCP para la interfaz indicada.

```
1 ip addr flush dev eth0
2 dhclient eth0
```

Código 3.13. Comandos para realizar petición DHCP

Se presentan los parámetros obtenidos en el caso de tres respectivos clientes solicitando parámetros a través del paquete DHCP ACK con la aplicación prototipo en la Figura 3.10, la Figura 3.11 y la Figura 3.12.

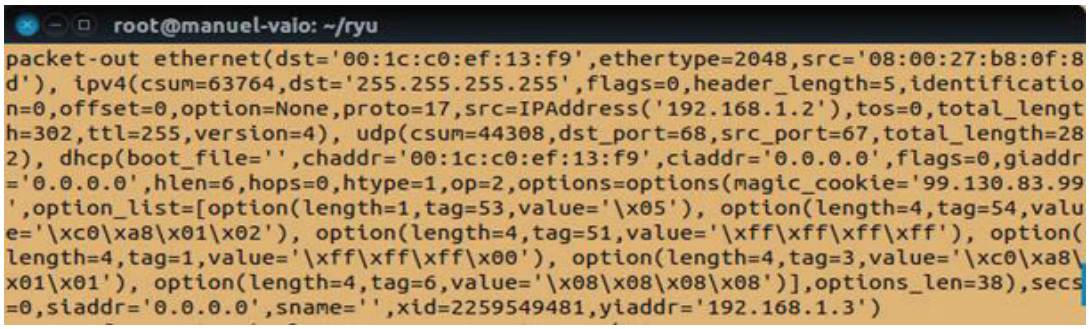

Figura 3.10. Parámetros para el cliente 1 en el paquete DHCP ACK

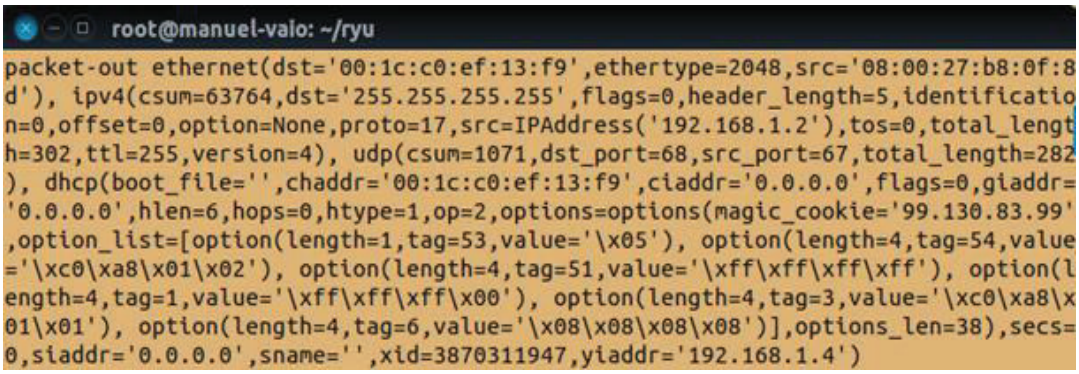

Figura 3.11. Parámetros para el cliente 2 en el paquete DHCP ACK

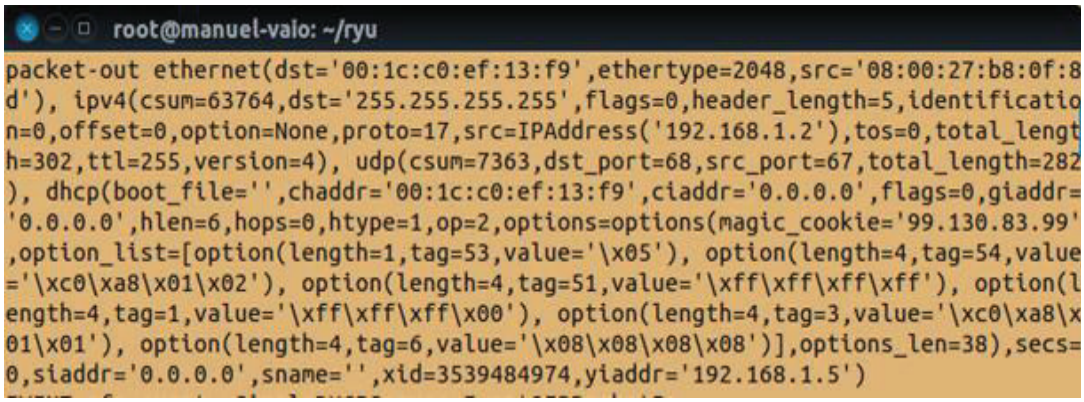

Figura 3.12. Parámetros para el cliente 3 en el paquete DHCP ACK

Se observa en la Figura 3.10 que en el campo `yiaddr` se provee la dirección IP 192.168.1.3 al primer cliente solicitante, para el caso de la Figura 3.11 este campo contiene la dirección IP 192.168.1.4 que será usada por el cliente 2 y finalmente en la Figura 3.12 se observa en el campo `yiaddr` la dirección IP 192.168.1.5 que corresponde al cliente 3.

Dentro del *switch* se crean tantos flujos como clientes se hayan conectado a la instancia que maneja OpenFlow, por estos flujos circulará todo el tráfico que no

contenga una regla explícita (es decir su campo `reason` contiene el valor `OFPR_NO_MATCH`), para así permitir que todos los paquetes que llegan al *switch* sean analizados por el controlador.

En la Figura 3.13 se observa el caso en que dos clientes se encuentran conectados al *switch* y el flujo que corresponde al primero de ellos.

```
Flow 1
Match
 Incoming Port : 3                    Ethernet Type    : Any
 Source MAC    : Any                  Destination MAC  : e0db55-953126
 Destination MAC Mask      : 000000-000000
 VLAN ID       : Any                  VLAN priority    : Any
 Source IP Address         : Any
 Destination IP Address    : Any
 IP Protocol   : Any                  IP ToS Bits      : Any
 Source Port   : Any                  Destination Port : Any
Attributes
 Priority      : 32768                Duration         : 2252269 seconds
 Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
 Byte Count    : 12784                Packet Count     : 132
 Controller ID : 1                    Cookie           : 0x0
 Flow Location : Software
 Hardware Index: NA
 Reason Code        : 15
 Reason Description : Rule cannot be accelerated in hardware
Actions
   Output                  : 4
```

Figura 3.13. Flujo correspondiente al cliente 1

La Figura 3.14 muestra el flujo correspondiente al segundo cliente.

```
Flow 2
Match
 Incoming Port : 4                    Ethernet Type    : Any
 Source MAC    : Any                  Destination MAC  : 001cc0-ef13f9
 Destination MAC Mask      : 000000-000000
 VLAN ID       : Any                  VLAN priority    : Any
 Source IP Address         : Any
 Destination IP Address    : Any
 IP Protocol   : Any                  IP ToS Bits      : Any
 Source Port   : Any                  Destination Port : Any
Attributes
 Priority      : 32768                Duration         : 2252269 seconds
 Hard Timeout  : 0 seconds            Idle Timeout     : 0 seconds
 Byte Count    : 12806                Packet Count     : 133
 Controller ID : 1                    Cookie           : 0x0
 Flow Location : Software
 Hardware Index: NA
 Reason Code        : 15
 Reason Description : Rule cannot be accelerated in hardware
Actions
   Output                  : 3
```

Figura 3.14. Flujo correspondiente al cliente 2

El tipo de dato `Ethernet Type` y su valor `any` muestran que la distinción de paquetes DHCP no se realiza a este nivel (de una validación de reglas en el *switch*) sino en la aplicación que reside en el controlador.

La cantidad de flujos creados puede ser observada en el *switch* al ejecutar el comando indicado en el Código 3.14 que permite observar las instancias OpenFlow creadas, como se aprecia en la Figura 3.15.

```
1 show openflow
```

Código 3.14. Mostrar instancias de OpenFlow en el *switch*

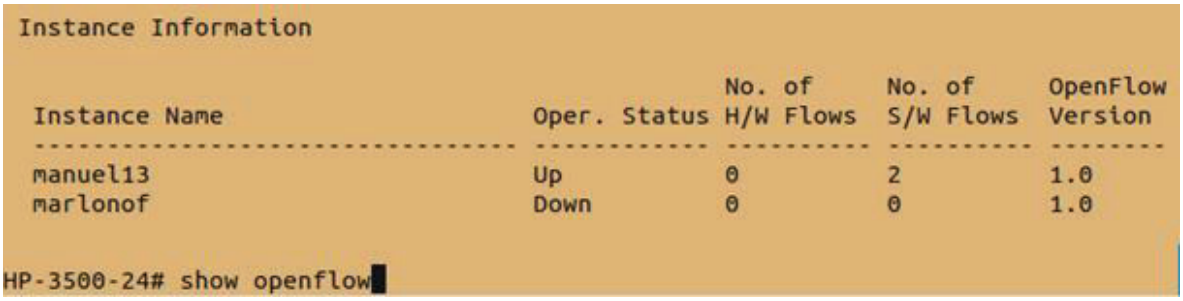

Figura 3.15. Instancias OpenFlow con su respectivo número de flujos

3.3. SERVIDOR LINUX DHCP3-SERVER

Para contrastar el funcionamiento de la aplicación prototipo DHCP sobre Ryu se ha decidido probar el funcionamiento de una aplicación tradicional que brinde el servicio de DHCP.

Se escogió el servidor DHCP3-SERVER (conocido como `isc-dhcp-server` a partir de Ubuntu 14.04) que se trata de una solución para Linux, cuya instalación y configuración se detalla en el Anexo C.

Su funcionamiento, del mismo modo que con la aplicación DHCP, se analiza mediante la ejecución de la instrucción `dhclient` desde un cliente Linux solicitante.

3.3.1. RESULTADOS OBTENIDOS

Se puede verificar el funcionamiento de este servidor mediante el análisis del archivo donde se registran los eventos del sistema operativo Ubuntu `/var/log/syslog`, en dichos eventos se encuentran los pasos de la transacción DHCP, que se observan en la Figura 3.16.

En los sistemas Linux es de gran utilidad revisar los archivos de tipo `log` para constatar el funcionamiento de un servicio.

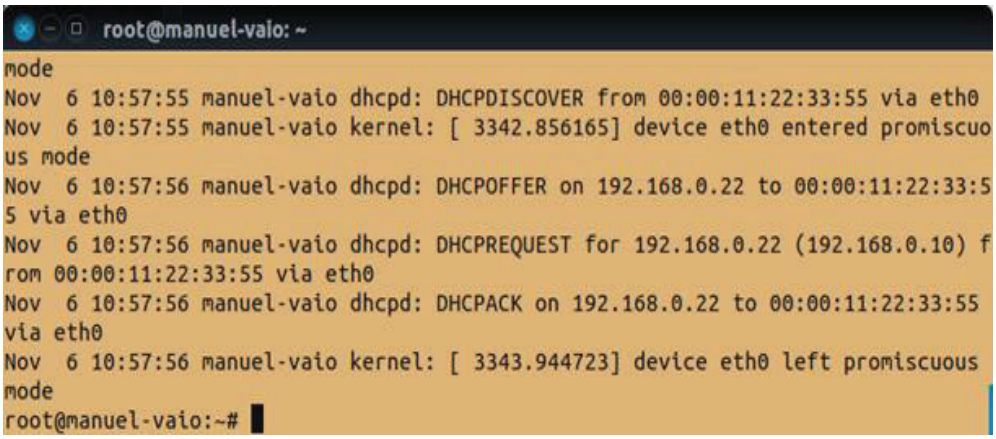

Figura 3.16. Registro del sistema de la transacción DHCP atendida por ISC DHCP SERVER

3.4. COMPARACIÓN DE FUNCIONAMIENTO ENTRE LA APLICACIÓN PROTOTIPO Y EL SERVIDOR DHCP3

El parámetro que se analizará a partir de los resultados obtenidos es el tiempo que requiere un cliente para tener conectividad una vez que adquiere parámetros de red mediante DHCP.

3.4.1. TIEMPOS DE ASIGNACIÓN

3.4.1.1. Tiempo de asignación en la aplicación prototipo DHCP

El análisis se ha dividido en dos escenarios: el primero corresponde al tiempo que toma una asignación inicial de parámetros de red mediante DHCP y el segundo escenario muestra el tiempo requerido para una renovación de dirección IP.

A continuación se muestra los resultados obtenidos en el primer escenario de estudio.

La Tabla 3.1 indica los valores obtenidos en las pruebas a partir de la ejecución de la instrucción del Código 3.15 que incluye la opción `-v` (`verbose`) para corroborar que ya está asignada la dirección IP. Los valores de tiempo tabulados han sido cronometrados.

```
1 dhclient eth0 -v
```

Código 3.15. Solicitud de parámetros DHCP en cliente Linux

NÚMERO DE PRUEBA	TIEMPO DE ASIGNACIÓN DE DIRECCIÓN IP Y PARÁMETROS
1	2.40 s
2	3.34 s
3	3.01 s
4	2.78 s
5	2.57 s
6	3.39 s
7	2.26 s
8	2.10 s
9	2.32 s
10	2.83 s

Tabla 3.1. Tiempo de funcionamiento de la aplicación Ryu

La media aritmética para el tiempo requerido para la asignación de dirección IP y verificación de conectividad se indica en la Figura 3.17.

También se calcula la desviación estándar de los datos recolectados en la Figura 3.18.

$$\bar{x} = \frac{X1 + X2 + X3 + \cdots Xn}{N}$$

$$\bar{x} = \frac{2.4 + 3.34 + 3.01 + 2.78 + 2.57 + 3.39 + 2.26 + 2.1 + 2.32 + 2.83}{10}$$

$$\bar{x} = 2.9\ seg$$

Figura 3.17. Cálculo de la media aritmética para la asignación en la aplicación DHCP

$$\sigma = \sqrt{\frac{(X1 - \bar{x})^2 + (X2 - \bar{x})^2 + \cdots + (Xn - \bar{x})^2}{N}}$$

$$\sigma = \sqrt{\frac{0{,}3^2 + 0{,}64^2 + 0{,}31^2 + 0{,}08^2 + 0{,}13^2 + 0{,}69^2 + 0{,}44^2 + 0{,}6^2 + 0{,}38^2 + 0{,}13^2}{10}}$$

$$\sigma = 0.426$$

Figura 3.18. Desviación estándar de tiempo de asignación en aplicación DHCP

3.4.1.2. Tiempo requerido para el servidor DHCP3

La Tabla 3.2 contiene los valores obtenidos en las pruebas de funcionamiento realizadas sobre el servidor tradicional mediante la ejecución del Código 3.15 desde un cliente Linux en similares condiciones a las indicadas para la aplicación DHCP en la sección 3.4.1.1.

NÚMERO DE PRUEBA	TIEMPO DE ARRANQUE DEL SERVIDOR
1	3.11 s
2	2.91 s
3	1.95 s
4	2.84 s
5	2.35 s
6	3.64 s
7	2.37 s
8	2.26 s
9	1.68 s
10	2.18 s

Tabla 3.2. Tiempo de funcionamiento del servidor ISC

Se calcula la media aritmética de los datos recopilados para el servidor tradicional en la Figura 3.19.

$$\bar{x} = \frac{X1 + X2 + X3 + \cdots Xn}{N}$$

$$\bar{x} = \frac{3.11 + 2.91 + 1.95 + 2.84 + 2.35 + 3.64 + 2.37 + 2.26 + 1.98 + 2.18}{10}$$

$$\bar{x} = 2.5\, seg$$

Figura 3.19. Media aritmética para asignación en el servidor ISC

Se ilustra la desviación estándar de los tiempos obtenidos al utilizar el servidor tradicional en la Figura 3.20.

$$\sigma = \sqrt{\frac{(X1 - \bar{x})^2 + (X2 - \bar{x})^2 + \cdots + (Xn - \bar{x})^2}{N}}$$

$$\sigma = \sqrt{\frac{0.55^2 + 0.35^2 + 0.61^2 + 0.28^2 + 0.21^2 + 0.19^2 + 1.38^2 + 0.3^2 + 0.58^2 + 0.38^2}{10}}$$

$$\sigma = 0.584$$

Figura 3.20. Desviación estándar del tiempo de asignación en ISC

Se puede concluir a partir de los tiempos recopilados que el tiempo requerido para la asignación de una dirección en la aplicación DHCP y la comprobación de que existe conectividad es mayor en medio segundo al tiempo transcurrido para la realización de esta tarea empleando el servidor tradicional.

Dicha situación se debe al tiempo que se requiere para la creación de la regla que indica el envío del mensaje `PACKET-OUT` con los parámetros a asignarse al cliente.

La diferencia de tiempo entre el servidor tradicional y la aplicación desarrollada no resulta ser un inconveniente mayor en cuanto a la demora que exista en la asignación ya que generalmente el proceso necesario para una configuración ocurre durante el arranque del sistema operativo y una espera de tres segundos implica que la dirección ya se encontrará asignada una vez que concluya el arranque del sistema operativo.

3.4.2. TIEMPOS DE RENOVACIÓN

3.4.2.1. Tiempo de renovación en la aplicación prototipo DHCP

A continuación se analizará el tiempo que toma la renovación de una dirección para un cliente. Normalmente un cliente DHCP enviará al *switch* un paquete de tipo REQUEST cuando el tiempo de préstamo transcurrido llega al 50% del valor configurado con una petición de renovación.

Para las pruebas se ha establecido un tiempo corto para poder recopilar la mayor cantidad de valores de renovación sin tener que esperar un tiempo prolongado. El valor establecido para el vencimiento del préstamo de la dirección IP es de 256

segundos, por lo cual el tiempo que se espera para que exista la generación de un paquete DHCP REQUEST por parte del cliente que desea la renovación será de aproximadamente 128 segundos.

Para la medición del tiempo se utiliza un cronómetro que arranca el momento que al cliente se le ha asignado una dirección IP y se detiene al momento que en Wireshark se recibe el paquete REQUEST con la petición de renovación del préstamo.

Los resultados obtenidos se detallan a continuación en la Tabla 3.3.

NÚMERO DE PRUEBA	TIEMPO DE RENOVACIÓN
1	132 s
2	129 s
3	133 s
4	129 s
5	135 s
6	130 s
7	127 s
8	132 s
9	131 s
10	132 s

Tabla 3.3. Tiempo de renovación de dirección IP en la aplicación prototipo

La media aritmética de los datos recopilados se muestra en la Figura 3.21.

$$\bar{x} = \frac{X1 + X2 + X3 + \cdots Xn}{N}$$

$$\bar{x} = \frac{132 + 129 + 133 + 129 + 135 + 130 + 127 + 132 + 131 + 132}{10}$$

$$\bar{x} = 131\ seg$$

Figura 3.21. Tiempo de renovación de préstamo en la aplicación prototipo

La desviación estándar del tiempo de renovación se indica en la Figura 3.22.

$$\sigma = \sqrt{\frac{(X1 - \bar{x})^2 + (X2 - \bar{x})^2 + \cdots + (Xn - \bar{x})^2}{N}}$$

$$\sigma = \sqrt{\frac{1^2 + 2^2 + 2^2 + 2^2 + 4^2 + 1^2 + 4^2 + 1^2 + 0^2 + 1^2}{10}}$$

$$\sigma = 2.19$$

Figura 3.22. Desviación estándar del tiempo de renovación en la aplicación

Los resultados obtenidos se encuentran dentro de los valores esperados, se tuvieron retardos aceptables que oscilan entre los 3 y 5 segundos, quizá considerables para el tiempo de préstamo escogido que fue pequeño, pero que en un escenario real son imperceptibles pues siempre es una buena práctica establecer un tiempo de préstamo de algunas horas.

3.4.2.2. Tiempo de renovación en el servidor ISC DHCP3

Para el caso del servidor tradicional las pruebas se realizaron en condiciones similares a las expuestas en la sección 3.4.2.1. Se ha establecido el tiempo de vencimiento del préstamo de direcciones IP igualmente en 256 segundos.

Los resultados obtenidos se encuentran en la Tabla 3.4.

NÚMERO DE PRUEBA	TIEMPO DE RENOVACIÓN
1	129 s
2	128 s
3	130 s
4	131 s
5	129 s
6	128 s
7	129 s
8	130 s
9	131 s
10	129 s

Se muestra el cálculo de la media aritmética de los tiempos requeridos para renovación en el servidor tradicional en la Figura 3.23.

También se indica el cálculo para obtener la desviación estándar estimada para el tiempo de renovación al momento de ultilizar el servidor tradicional para esta tarea en la Figura 3.24.

$$\bar{x} = \frac{X1 + X2 + X3 + \cdots Xn}{N}$$

$$\bar{x} = \frac{129 + 128 + 130 + 131 + 129 + 128 + 129 + 130 + 131 + 129}{10}$$

$$\bar{x} = 129.5\ seg$$

Figura 3.23. Media aritmética del tiempo de renovación en el servidor tradicional

$$\sigma = \sqrt{\frac{(X1 - \bar{x})^2 + (X2 - \bar{x})^2 + \cdots + (Xn - \bar{x})^2}{N}}$$

$$\sigma = \sqrt{\frac{0{,}5^2 + 1{,}5^2 + 0{,}5^2 + 2{,}5^2 + 0{,}5^2 + 1{,}5^2 + 0{,}5^2 + 0{,}5^2 + 1{,}5^2 + 0{,}5^2}{10}}$$

$$\sigma = 1.204$$

Figura 3.24. Desviación estándar en la renovación para el servidor tradicional

La mejora percibida es de uno a dos segundos menos para la renovación en el servidor tradicional con respecto a la aplicación desarrollada y se encuentra en un rango previsible dado que se trata de una solución para brindar el servicio DHCP muy difundida y con colaboradores que la mantienen en constante mejora, aun así la mejora percibida contrastada con la aplicación prototipo no muestra diferencias significativas, hay que recordar que el proceso de renovación se cumple a la mitad del tiempo de préstamo establecido, es decir que mientras los retardos sean mucho menores a la mitad del tiempo de préstamo el cliente no experimentará una pérdida de conectividad por motivo de renovación de dirección IP.

Al analizar tanto el tiempo de asignación de parámetros como el tiempo de renovación con cada una de las soluciones mencionadas se concluye que si bien el servidor tradicional brinda una atención más pronta a las solicitudes DHCP el grado

de personalización de los paquetes logrado con la aplicación prototipo justifica que el uso de la misma implique la adición de retardos de unos pocos segundos para atender a los solicitantes y se puede explicar que los retardos experimentados son causados por características propias del manejo del protocolo OpenFlow como es el caso de la creación de reglas que son elementos de tipo software y por lo tanto obedecerán a las capacidades del equipo bajo el que opera la aplicación.

CAPÍTULO IV

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- Una vez finalizado el presente proyecto se ha logrado desarrollar un prototipo de aplicación que brinda el servicio de DHCP utilizando Redes Definidas por Software, capaz de brindar concesiones de direcciones a equipos recientemente conectados a la red mediante una solicitud DHCP y asimismo capaz de renovar las concesiones de equipos ya conectados cuyo tiempo de concesión está próximo a expirar. Este prototipo ha sido evaluado y se encuentra funcionando tanto en un ambiente de tipo virtual como en un ambiente físico.

- Los tiempos que toman para la aplicación prototipo y para un servidor tradicional presentan una diferencia pequeña con respecto al tiempo total de concesión del préstamo que usualmente se considera para el manejo de parámetros de red establecidos mediante DHCP que son mayores para la aplicación desarrollada debido a la naturaleza del paradigma de las SDN que requiere la creación de reglas los mensajes OpenFlow que encapsulan la información del protocolo DHCP.

- Se pudo constatar la potencialidad que ofrece el trabajar bajo el paradigma de las SDN al ejecutar varias aplicaciones a la vez, en el caso del presente proyecto la aplicación DHCP prototipo se ejecutó al mismo tiempo que la aplicación `simple_switch` propia de Ryu. Si bien cada una de las aplicaciones realizan tareas distintas se pueden ejecutar sobre un mismo computador, presentando la ventaja que ambas pueden hacer uso de los recursos computacionales que posee dicho equipo en su totalidad pero

manteniendo una virtualización de servicios separada en la cual cada aplicación cumple sus tareas específicas.

- La implementación del módulo `solicitudRespuesta.py` requiere un profundo entendimiento del protocolo DHCP y de los pasos de la transacción para realizar el establecimiento de parámetros de red, ya que el controlador escogido provee herramientas que permiten la extracción de la carga útil que contiene la información del protocolo DHCP lo cual facilita el proceso de manipulación del paquete para luego ensamblar un paquete con los parámetros que se desea asignar.

- El módulo `atiendeIp.py` realiza la asignación de una dirección IP válida a partir de la extracción de dicha dirección de una lista y además realiza las exclusiones de direcciones reservadas que se indiquen mediante un llamado del método `excluir` definido dentro de este módulo de manera sencilla para evitar incluir retardos a la ejecución de la aplicación cuyo valor radica en la rapidez con la que atiende la solicitud DHCP, evitando el acceso a archivos o bases de datos que ralentizarían la ejecución de la aplicación.

- El controlador Ryu utilizado en el presente proyecto presenta ventajas al estar desarrollado e interpretar aplicaciones escritas en Python, pues este es un lenguaje bastante sencillo en su sintaxis y poderoso en su funcionalidad por lo que la tarea de codificar la idea principal a partir de la diagramación de las funciones que cumplirá la aplicación se realiza en un tiempo corto. Este controlador, al tratarse de un trabajo con constantes contribuciones de sus desarrolladores, tiene una comunidad de personas que brindan ayuda continua y de gran utilidad para las interrogantes que se plantean en los foros y listas de correo electrónico de desarrollo, lo cual constituyó un factor fundamental para la implementación del presente prototipo.

- Para obtener la ubicación dentro del paquete de los parámetros a establecerse por parte de la aplicación fue necesario el análisis del contenido de dicho paquete, para lo cual se utiliza una instancia del objeto `logger` definido para que se muestre en el *debug* de la aplicación. Al obtener un detalle del contenido del paquete en el *debug* se facilita observar cada uno de los valores que posee la lista de los campos encapsulados en el protocolo DHCP, para de este modo poder saber la ubicación del dato que se desea establecer mediante el índice del campo que sea de interés.

- Al utilizar el controlador que corre en un computador y el *switch* HP ProCurve en las pruebas en ambiente físico se determinó la importancia de la ubicación correcta del programa que monitorea el tráfico que circula por la red, en el caso del presente proyecto Wireshark, en función del tipo de paquete que se desea estudiar a detalle, pues los protocolos observables en el lado de la interfaz del controlador y en la interfaz del cliente son distintos y deben ser tratados por lo tanto con criterios diferentes, sobre todo en cuanto a las cabeceras de paquetes y la información que contienen para ambos casos de estudio según sea el interés del desarrollador.

- En cuanto a la cantidad de clientes admisibles por parte de la aplicación prototipo DHCP se observó que si bien en la etapa de pruebas en una red virtual se optó por el uso de dos clientes para constatar el funcionamiento de la aplicación ya que entre estos dos clientes ya fue posible evaluar la asignación de direcciones adecuadas y conectividad entre ambos equipos, el número de clientes que se admite por parte de la aplicación puede ser mayor y solamente se restringirá por el número de puertos pertenecientes a la VLAN que tiene OpenFlow habilitado en el *switch* o a su vez la cantidad de clientes que puede atender la aplicación conectados simultáneamente al *switch* se limitará al tamaño del rango de direcciones (pool) instanciado en la aplicación.

- Se resalta que la fase de pruebas de la aplicación prototipo DHCP fue fundamental para la depuración de errores que se presentaban con los parámetros cuyos valores deseados no se configuraban. Un ejemplo de esta situación se presentó con el tiempo de préstamo, el cual al realizar una conversión de un número decimal a hexadecimal generaba tiempos muy grandes de préstamo que no correspondían al valor en segundos deseado y posteriormente con la observación del formato de dicho tiempo dentro del paquete en Wireshark se optó por una conversión caracter por caracter que si generaba el valor deseado.

4.2. RECOMENDACIONES

- Se recomienda en la fase inicial de desarrollo de la aplicación la lectura del libro *OpenFlow Cookbook* [20] que contiene una explicación detallada y sencilla sobre el funcionamiento del protocolo OpenFlow con una familiarización inicial con todos los elementos que maneja dicho protocolo, tales como los mensajes y eventos. Teniendo un conocimiento referencial de los componentes de OpenFlow una vez que se escoja un controlador se facilita comprender sus particularidades y el uso práctico de los componentes de dicho protocolo para lograr desarrollar aplicaciones.

- Se recomienda realizar un estudio minucioso del protocolo DHCP que resulta fundamental para el entendimiento del funcionamiento de la aplicación desarrollada y adicionalmente se recomienda tener un conocimiento referencial de los protocolos que encapsulan a DHCP, como es el caso de UDP e IPv4 para interpretar adecuadamente los parámetros que cada uno de ellos posee para poder saber en cuál de estos protocolos se encuentra encapsulada la información requerida en cada etapa de la transacción necesaria para la obtención de parámetros mediante DHCP.

- Se recomienda a futuros desarrolladores suscribirse a la lista de correos de desarrollo del controlador escogido, la cantidad de información que se puede encontrar en este medio es actualizada regularmente y presenta modelos e ideas fundamentales al iniciarse en el desarrollo bajo el respectivo *framework.* Adicionalmente a esto es recomendable tener sólidos conocimientos del paradigma de programación a usarse, en el caso del presente proyecto la programación orientada a objetos, para acelerar el proceso de codificación de la aplicación.

- Se recomienda tener conocimientos básicos de la configuración de los equipos físicos a utilizarse en la fase de pruebas de toda aplicación que trabaje con SDN, pues al ser una tecnología que aún no tiene una difusión tan grande si bien hay varios fabricantes de equipos que soportan operar bajo este paradigma existe la posibilidad de errores a nivel de *hardware*. También se sugiere que mientras menos instancias se encuentren activas en el *switch* es menor la probabilidad de que se produzcan errores con el manejo de paquetes, lo ideal es tener solamente la instancia a usarse en estado activo o de preferencia solo exista dicha instancia y eliminar las demás en caso de no poder identificar un posible error que se presente solamente en el ambiente de pruebas en una red física.

- Es recomendable manejar el concepto de Alta Disponibilidad para despliegues en escenarios reales, pues el hecho de tener un ente centralizado ofrece grandes beneficios de los cuales ya se ha hablado pero presenta la enorme desventaja de depender de un solo equipo, que en el caso de falla puede dejar inoperante a toda la red. A breves rasgos se mencionó la solución *zookeeper* presente en Ryu para brindar este servicio y sería muy útil estudiarla a profundidad para así poder implementarla y brindar un respaldo al funcionamiento continuo del controlador mediante su ejecución en varios equipos.

- Se recomienda establecer un tiempo de préstamo que no se encuentre en un rango menor que veinte o treinta minutos, pues la cantidad de direcciones que brinda la aplicación DHCP puede proveer a varios clientes y resulta innecesario establecer préstamos de corta duración con el propósito de optimizar el uso de direcciones. Adicionalmente el establecer un préstamo de corta duración implicaría la generación de renovaciones que ocurrirían minutos después de la asignación, lo cual puede representar una sobrecarga para el controlador en el caso de existir varios clientes conectados simultáneamente. Los criterios de un tiempo de préstamo adecuado son variados pero con lo observado en las pruebas realizadas y el tiempo que un cliente utiliza una dirección IP en una situación real se sugieren rangos de entre 6 y 8 horas, cabe recalcar que la aplicación operará normalmente si se establece préstamos cortos.

- Se sugiere realizar en la aplicación un historial de las direcciones asignadas después de un tiempo prolongado de ejecución, para poder dimensionar la cantidad de clientes solicitantes que usualmente se encuentran de forma simultánea en la red y de este modo determinar si el rango de direcciones definido es del tamaño adecuado o poder redefinir este parámetro para que se ajuste al escenario de despliegue de la solución.

- Se recomienda realizar una revisión dentro del *switch* por si existe un servicio de DHCP activo en el mismo, en el caso del desarrollo de la aplicación DHCP el *switch* utilizado poseía su propio servicio DHCP que debió ser desactivado, si no se realiza esta comprobación se podría estar recibiendo resultados del servicio del *switch* y no de la aplicación desarrollada, pues un cliente DHCP hace una selección a partir del primer mensaje de tipo OFFER o ACK que reciba y no se puede establecer una prioridad para que el cliente escoja, por lo cual la única opción es tener solamente la aplicación prototipo desarrollada en ejecución. Asimismo si existiese un servidor conectado al *switch* se debe deshabilitar para evitar la situación previamente expuesta.

REFERENCIAS BIBLIOGRÁFICAS

[1] Open Networking Foundation. (2011) openflow.org. [Online]. http://archive.openflow.org/wp/learnmore/ (Consultado el 17 de Noviembre 2014)

[2] Varios Autores. (2014) SDX Central. [Online]. https://www.sdxcentral.com/resources/sdn/sdn-controllers/ (Consultado el 17 de Noviembre 2014)

[3] Open Networking Foundation. (2014, Junio) www.opennetworking.org. [Online]. https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf (Consultado el 20 de Noviembre 2014)

[4] Michael Cohen. (2011, Noviembre) Presentación Sllideshare. [Online]. http://es.slideshare.net/mscohen02/floodlight-overview (Consultado el 25 de Noviembre 2014)

[5] Yoshihiro Kaneko. (2014, Mayo) Ryu Wiki. [Online]. https://github.com/osrg/ryu/wiki (Consultado el 26 de Noviembre 2014)

[6] University of Oregon. (2014) Network Startup Resource Center. [Online]. https://nsrc.org/workshops/2014/nznog-sdn/raw-attachment/wiki/WikiStart/Ryu.pdf (Consultado el 13 de Dicembre 2014)

[7] OSRG. (2014, Septiembre) Integración de Snort. [Online]. https://github.com/osrg/ryu/blob/master/doc/source/snort_integrate.rst (Consultado el 21 de Diciembre 2014)

[8] Varios Autores. GitHub de Ryu. [Online]. https://github.com/osrg/ryu (Consultado el 29 de Diciembre 2014)

[9] IEEE. (2004, Septiembre) IEEE 802.1 ab. [Online]. http://www.ieee802.org/3/frame_study/0409/blatherwick_1_0409.pdf (Consultado el 29 de Diciembre 2014)

[10] Python Software Foundation. (2014) Python. [Online]. https://pypi.python.org/pypi/eventlet/0.15.2 (Consultado el 3 de Enero 2015)

[11] GMANE. Information about gmane.network.ryu.devel. [Online]. http://dir.gmane.org/gamen.network.ryu.devel (Consultado el 3 de Enero 2015)

[12] Open Networking Foundation. (2015) ONF Specifications: OpenFlow. [Online]. https://www.opennetworking.org/sdn-resources/openflow (Consultado el 4 de Enero 2015)

[13] ONF. (2011, Febrero) openflow.org. [Online]. http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf (Consultado el 3 de Enero 2015)

[14] Flowgrammable. (2015) Flowgrammable. [Online]. http://flowgrammable.org/sdn/openflow/message-layer/packetin/ (Consultado el 13 de Enero 2015)

[15] Mininet. (2015) mininet.org. [Online]. http://mininet.org/ (Consultado el 15 de Enero 2015)

[16] Bucknell University. (1997, Marzo) IETF - Dynamic Host Configuration Protocol. [Online]. https://www.ietf.org/rfc/rfc2131.txt (Consultado el 31 de Enero 2015)

[17] Paramiko. A Python implementation of SSHv2. [Online]. http://www.paramiko.org/ (Consultado el 2 de Febrero 2015)

[18] Bootstrap. (2015) Pip Installation. [Online]. https://bootstrap.pypa.io/get-pip.py (Consultado el 3 de Mayo 2015)

[19] OpenNetworking. (2009) openflow-spec-v1.0.0. [Online]. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf (Consultado el 10 de Febrero 2015)

[20] Kingston Smiler, *OpenFlow Cookbook*.: Packt Publishing, 2015. (Consultado el 8 de Agosto 2015)

[21] Nippon Telegraph and Telephone Corporation. (2014) Ryu 3.25 Documentation. [Online]. http://ryu.readthedocs.org/en/latest/ryu_app_api.html (Consultado el 16 de Febrero 2015)

[22] Nippon Telegraph and Telephone Corporation. (2015) Ryu GitHub. [Online]. https://github.com/osrg/ryu/blob/master/ryu/lib/packet/dhcp.py (Consultado el 27 de Abril 2015)

[23] H3C. (2015) H3C S3100-52P Ethernet Switch Operation Manual-Release 1500(V1.02). [Online]. http://www.h3c.com/portal/Technical_Support___Documents/Technical_Documents/Switches/H3C_S3100_Series_Switches/Configuration/Operation_Manual/H3C_S3100-52P_OM-Release_1500(V1.02)/201306/786871_1285_0.htm (Consultado el 7 de Septiembre 2015)

[24] WebSequenceDiagrams. (2012, Agosto) How to create sequence diagrams. [Online]. http://www.websequencediagrams.com/blog/?id=2 (Consultado el 19 de Mayo 2015)

[25] FlowGrammable. (2015) Message Layer / Hello. [Online]. http://flowgrammable.org/sdn/openflow/message-layer/hello/ (Consultado el 5 de Marzo 2015)

[26] Andy Hill. (2015, Abril) GitHub andyhky/ryu-dhcp. [Online]. https://github.com/andyhky/ryu-dhcp/blob/master/dhcp.py (Consultado el 10 de Agosto 2015)

[27] OpenDayLight Foundation. (2015, Agosto) Open Flow Plugin. [Online]. https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:OF1.3_Enabled_Wireshark (Consultado el 24 de Octubre 2015)

[28] Hewlett Packard. (2014, Marzo) HP OpenFlow 1.3 Administrator Guide. [Online]. http://sdn.ifmo.ru/hardware-switch/testbed/technical-documentation/hp-openflow-1-3-administrator-guide (Consultado el 3 de Octubre 2015)

[29] The Kermit Project Columbia University. (2011, Julio) Kermit Unix Manual Page and Tutorial. [Online]. http://www.columbia.edu/kermit/ckututor.html (Consultado el 26 de Septiembre 2015)

[30] OpenFlow.org. Configuring HP ProCurve. [Online]. http://archive.openflow.org/wk/index.php/Configuring_HP_Procurve#VLAN_Con

figuration (Consultado el 30 de Septiembre 2015)

[31] Mininet. (2015) Virtual Machines - Mininet. [Online]. https://github.com/mininet/mininet/wiki/Mininet-VM-Images (Consultado el 4 de Abril 2015)

[32] Ubuntu Handbook. (2015) How to Install PyCharm IDE in Ubuntu 14.04/15.04. [Online]. http://ubuntuhandbook.org/index.php/2015/07/install-pycharm-ubuntu-1404/ (Consultado el 7 de Marzo 2015)

[33] INTEF España. Aulas en red, aplicaciones y servicios Linux. [Online]. http://www.ite.educacion.es/formacion/materiales/85/cd/linux/m2/servidor_dhcp_dhcp3server.html (Consultado el 8 de Agosto 2015)

[34] Brent Salisbury - Installing Wireshark On Linux for OpenFlow Packet Captures. (2013, Febrero) NetworkStatic. [Online]. http://networkstatic.net/installing-wireshark-on-linux-for-openflow-packet-captures/ (Consultado el 18 de Septiembre 2015)

[35] Ryu 3.26 documentation. (2015) OpenFlow v1.3 Messages and Structures. [Online]. http://ryu.readthedocs.org/en/latest/ofproto_v1_3_ref.html (Consultado el 4 de Junio 2015)

[36] saravana815. (2012, Febrero) Wordpress - Linux dhcp client simulation tool. [Online]. https://sargandh.wordpress.com/2012/02/23/linux-dhcp-client-simulation-tool/ (Consultado el 18 de Septiembre 2015)

[37] saravana815. (2015, Octubre) GitHub - DHCP simulation tool. [Online]. https://github.com/saravana815/dhtest (Consultado el 18 de Septiembre 2015)

ANEXOS

ANEXO A: INSTALACIÓN DE MININET EN UBUNTU 14.04 LTS

ANEXO B: INSTALACIÓN DE PYCHARM EN UBUNTU 14.04 LTS

ANEXO C: INSTALACIÓN DE DHCP3-SERVER EN UBUNTU 14.04 LTS

ANEXO D: CAPTURA DE MENSAJES OPENFLOW DESDE WIRESHARK

ANEXO A

INSTALACIÓN DE MININET EN UBUNTU 14.04 LTS

Se presenta a continuación un tutorial con las indicaciones para la configuración e instalación de Mininet.

INSTALACIÓN DE MININET

Para poder probar de manera virtual el funcionamiento de la red SDN bajo la cual operará la aplicación se acude a Mininet, una herramienta flexible para la emulación de un ambiente de red que soporta todos los protocolos y hardware emulado necesario para el despliegue del escenario donde actúa la aplicación prototipo desarrollada.

Para instalar y configurar Mininet existen dos posibilidades, una máquina virtual donde esté pre configurado Mininet o una instalación limpia sobre el sistema.

INSTALACIÓN POR MÁQUINA VIRTUAL

En caso de optar por el uso de una máquina virtual, la opción recomendad en la página oficial del proyecto Mininet [15], se procede a descargar la imagen desde el sitio oficial [30], teniendo en cuenta la arquitectura del computador donde se instalará Mininet (sea x86 para 32 bits o x64 para 64 bits, siendo la última la adecuada para la mayoría de equipos nuevos).

Posteriormente se escoge un sistema de virtualización, donde puede ser de preferencia utilizar un sistema con que el usuario tenga más familiaridad, y asimismo se da la sugerencia de escoger el sistema VirtualBox por ser software libre en caso de no conocer previamente sobre sistemas de virtualización.

La sugerencia brindada por parte de la guía de instalación de suscribirse a la lista de correos de la comunidad Mininet es conveniente para poder aclarar dudas que surjan

en cuanto a su manejo y capacidades adicionales que brinden solución a problemas que puedan surgir durante el manejo del entorno de *Mininet.*

Al abrir la imagen de la máquina virtual con el sistema de virtualización escogido ya está lista para su uso.

INSTALACIÓN DIRECTA EN EL SISTEMA NATIVO

La segunda opción resulta similar al proceso por el cual se instala el controlador Ryu, mediante el repositorio de código GitHub en un sistema limpio, es decir sin instalaciones previas de Mininet.

Se comprobó durante el desarrollo de este proyecto, que es muy importante que la versión del sistema sea la más actual (en el caso del presente proyecto Ubuntu 14.04) para el correcto funcionamiento del componente Open vSwitch.

Como primer paso se clona el código fuente con el comando indicado en el Código A-1:

```
1 git clone git://github.com/mininet/mininet
```

Código A – 1. Obtención del Código Fuente de Mininet

Posteriormente se ingresa el comando para instalar Mininet indicado en el Código A-2*:*

```
1 mininet/util/install.sh [options]
```

Código A – 2. Script de Instalación de Mininet

En [`options`] se puede agregar opciones adicionales listadas a detalle en la página de descarga de Mininet como es el caso de la opción –a, que instala todo el paquete adicional de Mininet que incluye componentes como Open vSwitch y POX, otro controlador SDN. Una vez hecho esto es posible ingresar a Mininet, siempre con privilegios de root, mediante el comando `mn`.

Para el proyecto se escogió correr Mininet en una máquina virtual, en su versión de 32 bits dado que la máquina donde fue instalada carecía de la opción de habilitar VT-x/AMD-V en el BIOS de la máquina; mientras que Ryu se encuentra en el sistema nativo, para este caso Ubuntu. Es necesario entonces establecer la forma de comunicación entre la máquina virtual y el sistema nativo.

Posteriormente a realizar varias evaluaciones con distintos métodos de interconexión se escogió la alternativa de crear una interfaz de tipo *host only.* A continuación se indica el proceso para crear y configurar dicha interfaz.

En la línea de comandos de Ubuntu se ingresa el comando indicado en el Código A-3:

```
1 VBoxManage hostonlyif create
```

Código A – 3. Creación de interfaz *host-only*

Una vez ingresado el comando se recomienda verificar la creación de dicha interfaz mediante el comando ifconfig (habiendo encendido previamente la máquina virtual respectiva).

Se observará una interfaz similar a la observada en la Figura

```
manuel@manuel-vaio:~$ VBoxManage hostonlyif create
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Interface 'vboxnet1' was successfully created
manuel@manuel-vaio:~$ ifconfig vboxnet1
vboxnet1  Link encap:Ethernet  HWaddr 0a:00:27:00:00:01
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figura A-1. Creación de la interfaz *host only*

CONFIGURACIÓN DE LA INTERFAZ HOST ONLY VBOXNET0

Al abrir VirtualBox y cargar la imagen de la máquina virtual de Mininet se edita la configuración de red, en la Figura A-2 se muestra la ubicación de las pestaña de configuración.

Figura A-2. Configuración de vboxnet0 1

En la Figura A-3 se observa la ventana emergente de configuración, una vez dentro de esta ventana se escoge la pestaña Red, y en las opciones de conectado se

escoge adaptador sólo anfitrión. Al solo existir `vboxnet0` se escoge por defecto dicha interfaz.

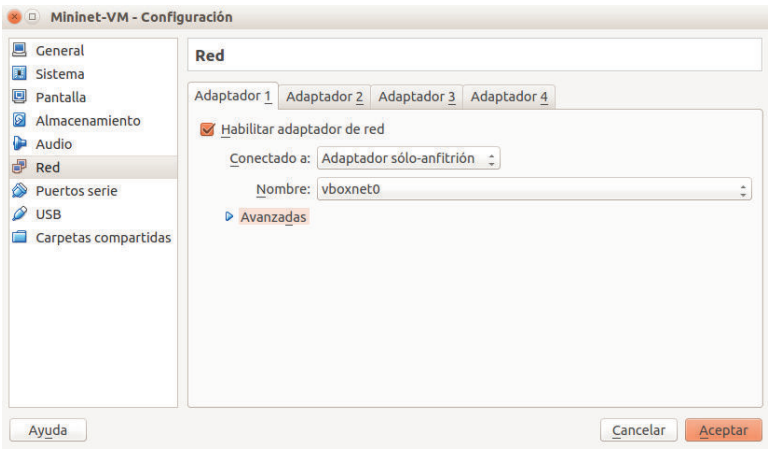

Figura A-4. Configuración de vboxnet0 2

Como se indicó previamente VirtualBox se encarga de darle una dirección mediante un servicio DHCP interno de dicho programa, se debe tener en cuenta la subred de dicha dirección para que coincida la subred en las demás interfaces a establecerse. A continuación se configura tanto en la máquina virtual como en el sistema nativo los adaptadores `eth0` respectivos.

CONFIGURACIÓN DE LAS INTERFACES ETH0

Tanto en la máquina virtual como en el sistema nativo es necesario configurar la interfaz `eth0`, que es la que en ambos casos apunta hacia la interfaz `vboxnet0` y debe pertenecer a la misma subred.

Para configurarla se corre el comando indicado en la Figura A-4, dependiendo de si se trata del sistema nativo o de la máquina virtual según el diagrama propuesto previamente.

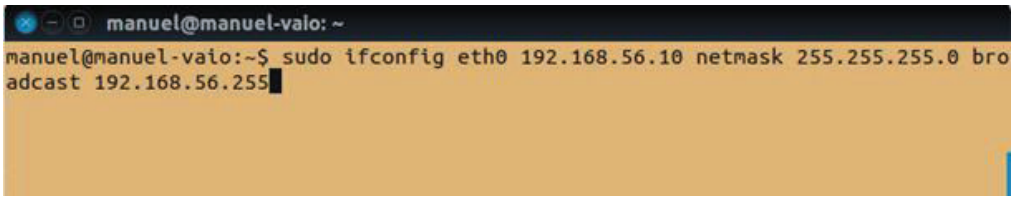

Figura A-4. Configuración de la interfaz `eth0`

Finalmente se recomienda realizar pruebas de conectividad ejecutando el comando `ping` tanto desde la máquina virtual hacia Ubuntu como en el sentido contrario.

En el caso de Mininet la interfaz `eth0` puede previamente tener una dirección asignada por el DHCP interno de VirtualBox, si su dirección configurada pertenece a la misma red se puede obviar el paso de configurar de nuevo `eth0`, en todo caso es recomendado realizar una prueba de conectividad.

ANEXO B

INSTALACIÓN DE PYCHARM EN UBUNTU 14.04 LTS

El IDE[48] escogido para el desarrollo de la aplicación prototipo es PyCharm, en su versión *community (*gratuita*)*. Este programa posee un editor inteligente que ayuda a completar código automáticamente, subraya errores mientras se escriben y además provee una corrección autónoma de errores de tipo.

Una ventaja evidente que ofrece PyCharm, una vez instalado en el sistema nativo que contiene el controlador (Ryu) es que inicialmente el programa busca todas las librerías y compiladores Python instalados, con lo cual es transparente el uso y llamado a clases propias del controlador sin mostrarse como errores, algo que puede suceder en otros editores de código.

Para la instalación de PyCharm [31] se ha escogido añadir el repositorio PPA, este proceso se indica en la Figura B-1:

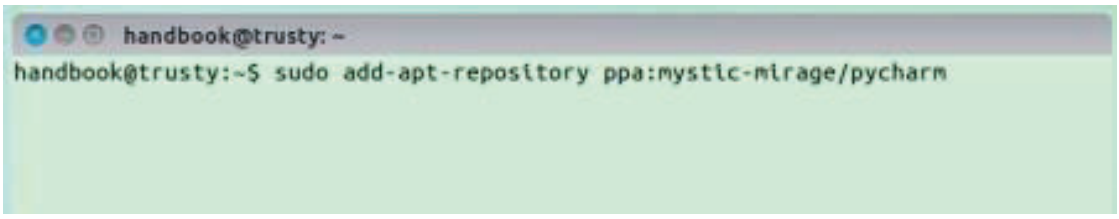

Figura B – 1. Adición del Repositorio PPA

Posteriormente, se realiza una actualización de caché de paquetes del sistema y se instala el IDE, este proceso se encuentra respectivamente indicado en la Figura B-2:

```
sudo apt-get update

sudo apt-get install pycharm
```

Figura B – 2. Comandos de Instalación de PyCharm

Las aplicaciones de Ryu, al tratarse de scripts Python, pueden ser fácilmente copiadas en la carpeta respectiva para el controlador, por lo cual una vez realizado el script en el IDE de PyCharm lo mejor es correrlo en Ryu para poder identificar el comportamiento de la aplicación de mejor manera; se recalca que el uso de

---

[48] IDE (*Integrated Development Environment*), usado para proveer un ambiente gráfico de desarrollo con todas las herramientas necesarias integradas.

PyCharm es de suma utilidad para la parte de sintaxis y llamado de clases adecuado, pero la ejecución desde el IDE puede diferir del comportamiento completo de la aplicación una vez interpretada por el controlador.

ANEXO C

INSTALACIÓN DE DHCP3-SERVER EN UBUNTU 14.04 LTS

La instalación en Linux (para el caso de este proyecto en su versión Ubuntu 14.04 LTS) es sencilla, se ingresará en consola el comando indicado en el Código C – 1:

```
1 sudo apt-get install dhcp3-server
```

Código C – 1. Instalación de `dhcp3-server`

Para configurar los distintos parámetros que manejará el servidor hay que modificar su archivo de configuración, ubicado en `/etc/dhcp3/dhcpd.conf`.

Los distintos parámetros a configurarse son los mismos que se han considerado en el presente Proyecto. Entre ellos se pueden mencionar rangos de direcciones, máscara de red, servidor DNS, entre otros.

A continuación en el Código C-2 se presenta un ejemplo explicado de dichos parámetros:

```
1 subnet 192.168.1.0 netmask 255.255.255.0 {
2 option routers 192.168.1.254;
3 option domain-name-servers 8.8.8.8, 195.53.123.57;
4 range 192.168.1.60 192.168.1.90;
5 }
```

Código C – 2. Ejemplo de parámetros en `dhcpd.conf`

En la línea 1 del Código C-2 se indica tanto la subred en la que se trabajará como la máscara de subred que la define. En la línea 2 se indica la puerta de enlace predeterminada que se utilizará. La línea 3 indica opciones de servidores DNS y finalmente la línea 4 indica el rango de direcciones posibles a ser concesionadas a los clientes.

El análisis de los campos del archivo `dhcpd.conf` es muy intuitivo, en el Código C-3 se brinda un archivo comentado elaborado por [32].

```
1 # Sample configuration file for ISC dhcpd for Debian
2 # $Id: dhcpd.conf,v 1.4.2.2 2002/07/10 03:50:33 peloy Exp
3
4 # Opciones de cliente y de dhcp aplicables por defecto a todas las
  secciones
5
6 # Estas opciones pueden ser sobreescritas por otras en cada sección
7 option domain-name-servers 195.53.123.57; # DNS para los clientes
  (atenea)
```

```
option domain-name &quot;ieslapaloma.com&quot;; # Nombre de dominio para los clientes
option subnet-mask 255.255.255.0; # Máscara por defecto para los clientes
default-lease-time 600; # Tiempo en segundos del 'alquiler'
max-lease-time 7200; # Máximo tiempo en segundos que durará el 'alquiler'

# Especificación de un rango
subnet 192.168.1.0 netmask 255.255.255.0 {
range 192.168.1.60  192.168.1.80; # Rango de la 60 a la 80 inclusive
option broadcast-address 192.168.1.255; # Dirección de difusión
option routers 192.168.1.254; # Puerta de enlace
option domain-name-servers 80.58.0.33; # DNS (ej: el de telefónica)
default-lease-time 6000; # Tiempo de prestamo
}
# Configuración particular para un equipo
host aula5pc6 {
hardware ethernet 00:0c:29:1e:88:1d; # Dirección MAC en cuestión
fixed-address 192.168.1.59; # IP a asignar (siempre la misma)
}
```

Código C – 3. Ejemplo comentado de configuración de `dhcpd.conf`

Para iniciar o detener este servidor DHCP, se debe trabajar con el script de arranque de este servicio, como es el caso con muchos de los servicios de Ubuntu. Este script se ubica en `/etc/init.d`. El Código C-4 indica los comandos que se debe correr tanto para iniciar como para detener este servicio.

```
// Arrancar o reiniciar el servidor DHCP
sudo /etc/init.d/dhcp3-server restart

// Parar el servidor DHCP
sudo /etc/init.d/dhcp3-server stop
```

Código C – 4. Comandos parar iniciar o detener el servidor `dhcp`

Siempre es recomendable verificar que los rangos de direcciones a disposición se encuentren en la misma subred que la dirección del servidor DHCP, caso contrario será imposible para los clientes llegar al servidor por esta inconsistencia.

Asimismo puede que no sea posible levantar este servicio si algún servicio similar, como es el caso de DNSMASQ está activo. En tal caso, para detener DNSMASQ se ejecuta el comando `/etc/init.d/dnsmasq stop`.

ANEXO D

CAPTURA DE MENSAJES OPENFLOW DESDE WIRESHARK

El Código D-1 indica los comandos a ejecutarse como requisitos para lograr capturar mensajes OpenFlow [33]:

```
apt-get install wireshark-dev wireshark mercurial git
git clone git://openflow.org/openflow.git
cd openflow
./boot.sh
./configure
make && make install
cd utilities/wireshark_dissectors/openflow
gedit packet-openflow.c
```

Código D – 1. Comandos de configuración para OpenFlow en Wireshark

En el archivo `packet-openflow.c` se modifica la función `proto_reg_handoff_openflow()` como se indica en el Código D-2:

```
void proto_reg_handoff_openflow()
{
openflow_handle =
create_dissector_handle(dissect_openflow,
proto_openflow);
dissector_add_uint(TCP_PORT_FILTER,
global_openflow_proto, openflow_handle);
}
```

Código D – 2. Modificación de la función `proto_reg_handoff_openflow()`

Una vez realizados estos cambios se ejecutan los comandos indicados en el Código D-3:

```
make && make install
cp /var/packet-openflow.so
/usr/lib/wireshark/libwireshark1/plugins/
wireshark
```

Código D – 3. Cambios previos a la ejecución de Wireshark

ANEXO E

El código de la aplicación se encuentra en el CD adjunto.