Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Alejandro Sánchez Carrasco

RouteFlow: Migración al Software Defined Networking

Proyecto Final de Grado en Ingeniería Telemática
Tutor:Jose Muñoz Tapia
Barcelona, Junio 2016

RouteFlow

Migración al Software Defined Networking

Autor: Alejandro Sánchez Carrasco
Tutor: Jose L. Muñoz Tapia

Abstract

This project seeks to highlight what is the Software Defined Networking and which components form it. After the subject knowledge has been exposed, an application based in in software defined networks called RouteFlow will also be explained in detail. This application is part of an Open Source project created by the Centro de Pesquisa e Desenvolvimento (CPqD) and its purpose is to create an IP routing platform with centralized control for networks managed with the OpenFlow protocol. Finally, some instructions are listed to configure the application and the application is tested in a virtual network of example.

Resum

Aquest projecte pretén donar a conèixer què és el *Software Defined Networking* i quins components el formen. Un cop exposats els coneixements sobre el tema, s'explicarà també en detall una aplicació anomenada RouteFlow basada en les xarxes definides per software. Aquesta aplicació pertany a un projecte Open Source creat pel Centro de Pesquisa e Desenvolvimento (CPqD) i té com a objectiu crear una plataforma de enrutament IP amb control centralitzat per a xarxes gestionades amb el protocol OpenFlow. Finalment, es detallen les instruccions a seguir per a configurar l'aplicació i es posa a prova el seu funcionament amb una xarxa virtual d'exemple.

Resumen

Este proyecto pretende dar a conocer qué es el *Software Defined Networking* y qué componentes lo forman. Una vez se hayan expuesto los conocimientos sobre el tema, se explicará también en detalle una aplicación llamada RouteFlow basada en las redes definidas por software. Esta aplicación pertenece a un proyecto Open Source creado por el Centro de Pesquisa e Desenvolvimento (CPqD) y su propósito es crear una plataforma de enrutamiento IP con control centralizado para redes gestionadas con el protocolo OpenFlow. Finalmente, se detallan las instrucciones a seguir para configurar dicha aplicación y se pone a prueba su funcionamiento con una red virtual de ejemplo.

Dedicatoria

Este trabajo quiero dedicarlo a mi pareja y a mi familia, por aguantarme durante la realización del mismo y ayudarme a seguir adelante. En especial quisiera dedicárselo a mis padres, porque con su esfuerzo y su confianza en mí han hecho posible el que yo llegara hasta aquí.

Agradecimientos

Quiero agradecer a mi tutor de proyecto Jose L. Muñoz, por darme a conocer el *Software Defined Networking*, ya que me ha hecho ver la importancia que tendrá esta tecnología en los próximos años, y por ayudarme en el proceso de realización del presente documento. También agradecerle la dedicación y empeño que ha puesto siempre en las diferentes asignaturas que me ha impartido, para que tanto yo como mis compañeros llegáramos a aprender de verdad en sus clases. Debo agradecer también a todos los profesores que al igual que Jose, se han esforzado para que tanto yo como mis compañeros pudiéramos llegar hasta aquí.

Historial de revisiones y registro de aprobación

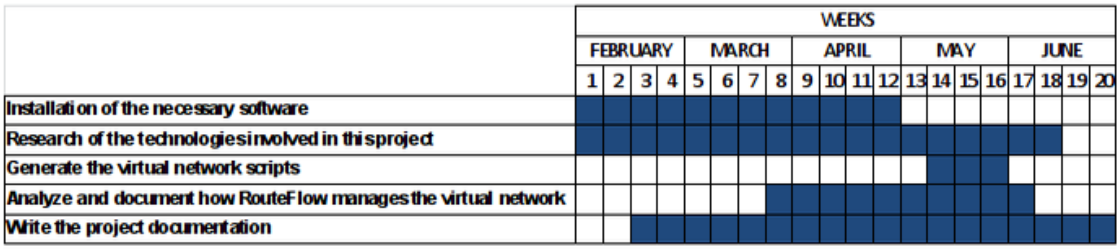
Revisión	Fecha	Propósito
0	15/02/2016	Creación del documento
1	24/03/2016	Revisión del documento
2	21/04/2016	Revisión del documento
3	26/05/2016	Revisión del documento
4	14/06/2016	Revisión del documento
5	24/06/2016	Revisión del documento

LISTA DE DISTRIBUCIÓN DEL DOCUMENTO

Nombre	e-mail
Alejandro Sánchez Carrasco	jandur92@gmail.com
Jose L. Muñoz Tapia	jose.munoz@entel.upc.edu

Escrito por:		Revisado y aprobado por:	
Fecha:	25/06/2016	Fecha:	27/06/2016
Nombre:	Alejandro Sánchez Carrasco	Nombre:	Jose L. Muñoz Tapia
Cargo:	Autor del proyecto	Cargo:	Supervisor del proyecto

DIAGRAMA DE GANTT



Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Vivimos en el inicio de una nueva era para las tecnologías de la información, llamada por algunos la nueva revolución digital, en la que se empiezan a vislumbrar nuevos planteamientos que pretenden renovar los esquemas establecidos hasta el momento para ofrecer mejores prestaciones y abrir paso a nuevas tecnologías que están por venir.

Esta explosión de ideas hace de las tecnologías de la información un sector cambiante y en constante crecimiento, cuyas tecnologías evolucionan sin parar, con conceptos cuya definición puede verse modificada con el paso de los meses.

Con un panorama tan frenético uno debe estar constantemente informado para no quedarse desactualizado. Esto hace que a veces sea difícil introducirse en este tipo de temas debido a lo cambiantes que pueden llegar a ser sus contenidos.

Es por ello que este trabajo pretende mostrar una fotografía de una de estas tecnologías de las que se hablaba anteriormente, para ofrecer un punto de partida desde el que poder iniciar el aprendizaje. Esta tecnología se trata del Software Defined Networking, cuyo principio básico son las redes definidas por software. Según los expertos en tecnologías de la información y comunicación, este concepto tendrá un papel muy importante en las futuras generaciones de la telecomunicación, siendo ya una de las nuevas tecnologías que formarán parte de la quinta generación de telefonía móvil. Concretamente, este documento tiene como objetivo presentar una aplicación llamada RouteFlow basada en el paradigma SDN. Dicha aplicación, ofrece la posibilidad de aplicar este paradigma en las redes tradicionales de forma gradual.

Los conocimientos previos sobre *Software Defined Networking* necesarios para la comprensión de los capítulos que forman este documento, pueden encontrarse en los apéndices A y B.

Los capítulos del presente documento se estructurarán de la siguiente manera: el primer capítulo explicará al lector qué es RouteFlow y cómo funciona; el siguiente capítulo explicará cómo crear una red virtual gestionada con RouteFlow, detallando los pasos de instalación y configuración; finalmente, se concluirá el proyecto con un capítulo que incluirá un estudio presupuestario del proyecto, así como las conclusiones deducidas de los conceptos expuestos a lo largo del documento.

Capítulo 2

RouteFlow

2.1. Introducción

RouteFlow es una arquitectura de enrutamiento definida por software [1] y desarrollada por el Centro de Pesquisa e Desenvolvimento (CPqD) que proporciona una plataforma de enrutamiento IP con control centralizado, que puede ser fácilmente personalizable y que funciona con cualquier software de red basado en Linux. Esta plataforma que ofrece servicios de enrutamiento (y puede ofrecer *Routing-as-a-Service*[1]) simplifica enormemente el proceso de configurar un protocolo de enrutamiento, sobre todo cuando hay que hacerlo para un gran número de *switches* distribuidos [3]. Actualmente está siendo mantenido y desarrollado por la comunidad de Internet, dada su condición de proyecto de código abierto.

RouteFlow nace de la necesidad de hacer frente a uno de los mayores obstáculos que se encuentran las nuevas tecnologías: proveer un mecanismo para la interoperabilidad con los sistemas ya existentes. Lo que pretende hacer RouteFlow es ayudar en la migración hacia el uso de *datapaths* que utilicen OpenFlow, ofreciendo servicios de enrutamiento tradicionales basados en OpenFlow. Esto permite desplegar gradualmente dispositivos OpenFlow junto con los dispositivos existentes, en vez de reemplazar una red entera [4].

Los *routers* tradicionales utilizan protocolos distribuidos para compartir información de accesibilidad con los dispositivos vecinos. Cada uno de estos dispositivos contiene un motor de enrutamiento que construye un mapa de la topología de red mediante la información recopilada de sus vecinos y genera un conjunto de reglas sobre cómo enviar los paquetes. RouteFlow funciona de forma similar creando instancias de motores de enrutamiento virtuales (normalmente Quagga) y asociándolas con los *datapaths*.

La idea que hay detrás del diseño de RouteFlow se basa en el paradigma de las SDN en el que el plano de datos se separa del plano de control. El plano de control de RouteFlow consiste en una serie de *routers* virtuales cada uno de estos asociado a un *datapath*; estos últimos redirigen todo el tráfico del protocolo de red a su *router* virtual asociado. De esta manera, el *router* virtual aprende las rutas como si estuviera recibiendo el tráfico directamente y luego las añade a la tabla de flujos del *datapath*.

[1] concepto que consiste en ofrecer como servicio el cálculo personalizado de rutas, ofreciendo a los usuarios la posibilidad de elegir el tipo de control que tienen sobre el enrutamiento de su tráfico. [2]

En la figura 2.1 se muestra la relación entre el motor de enrutamiento, RouteFlow y un *datapath* gestionado con RouteFlow. Posteriormente, se detallan los pasos que se siguen para instalar una nueva ruta de red:

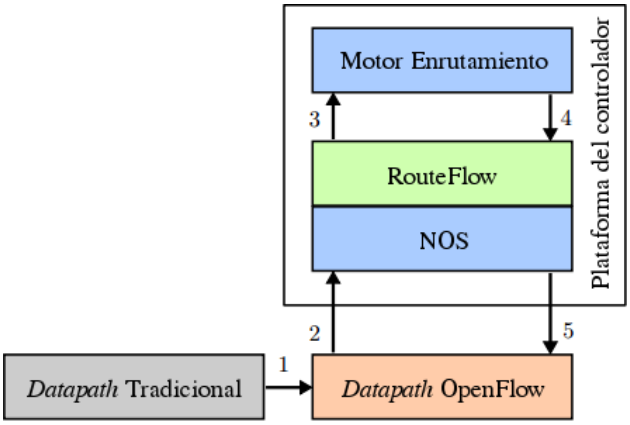


Figura 2.1: Secuencia de eventos al instalar una nueva ruta.

1. El *datapath* tradicional envía una actualización de ruta mediante protocolos de enrutamiento estándar (BGP, OSPF, RIP).
2. El *datapath* OpenFlow reconoce el protocolo de enrutamiento y reenvía el mensaje al controlador.
3. RouteFlow recibe el mensaje a través del *Network Operating System* (*NOS*) de OpenFlow y se lo pasa al motor de enrutamiento.
4. El motor de enrutamiento determina si se debe almacenar la nueva ruta en el *datapath* OpenFlow y se lo notifica a RouteFlow.
5. RouteFlow convierte el mensaje del motor de enrutamiento en uno de modificación de flujo OpenFlow y lo envía al *datapath*. Desde este momento en adelante, todo el tráfico que llegue al *datapath* y coincida con esta ruta, será remitido directamente hacia su destino, sin pasar por el controlador.

Los *routers* virtuales son Máquinas Virtuales Linux (*Virtual Machines*) funcionando con software de enrutamiento tradicional. Cada puerto del *datapath* está asociado con una única interfaz de la VM. Cuando un *datapath* recibe tráfico del plano de control por uno de sus puertos, lo redirige a su controlador para que lo envíe al puerto de la VM correspondiente. De la misma manera, los paquetes del protocolo de red enviados por la VM son direccionados al puerto correspondiente del *datapath*. El comportamiento de reenvío del *datapath* se define a partir de la información obtenida de sondear el *host*[2] y las tablas de rutas de la VM. Esta información es utilizada para crear las entradas de la tabla de flujos del *datapath*.

RouteFlow se ejecuta como una aplicación sobre el *NOS* de OpenFlow. Actualmente, RouteFlow es compatible con múltiples implementaciones del *NOS* de OpenFlow. Los creadores del proyecto, junto con otros colaboradores, están trabajando para mejorar la compatibilidad con el protocolo de Quagga y con otras implementaciones del *NOS* de OpenFlow. Sin embargo, el desarrollo de RouteFlow para soportar más protocolos es menos activo.

Siguiendo las mejores prácticas del diseño de aplicaciones de la nube[3], RouteFlow se basa en una base de datos escalable y tolerante a fallos, que centraliza (i) el estado central del panel de control de RouteFlow (p.ej. asociaciones de recursos), (ii) la visión de la red (lógica, física, y especifica del protocolo), y (iii) cualquier base de información (p.ej. histogramas/previsiones de tráfico, monitorización de flujos, políticas de administración) utilizada para desarrollar aplicaciones de control de rutas [5].

[2]término usado en informática para referirse a las computadoras conectadas a una red.
[3]aplicaciones que están alojadas en servidores de Internet.

2.2. Arquitectura de RouteFlow

La arquitectura de la plataforma RouteFlow la forman tres componentes principales: el Cliente RouteFlow (RFClient), que se encarga de controlar la VM; el Servidor RouteFlow (RFServer), el cual controla las asociaciones entre *datapaths* y VMs; y el Proxy RouteFlow (RFProxy), encargado de controlar los *datapaths*. Existe un módulo adicional llamado RFMonitor que se utiliza para la configuración de varios controladores en un mismo *switch*.

Estos elementos se comunican mediante el Protocolo RouteFlow (RFP), el cual define las interacciones entre los distintos componentes de RouteFlow. La comunicación entre procesos (IPC) se realiza con MongoDB [6], una base de datos NoSQL, que también se utiliza para almacenar el estado y la configuración de la plataforma. La interconexión de los elementos de RouteFlow con la base de datos se realiza con un *bridge*[4] denominado de "control". Para el reenvío del tráfico procedente del plano de control, entre los *datapaths* y las VMs, se utiliza una instancia de Open vSwitch [7]. En la figura 2.2 se puede observar la relación que tienen los componentes principales junto con el *software* que necesita RouteFlow para funcionar.

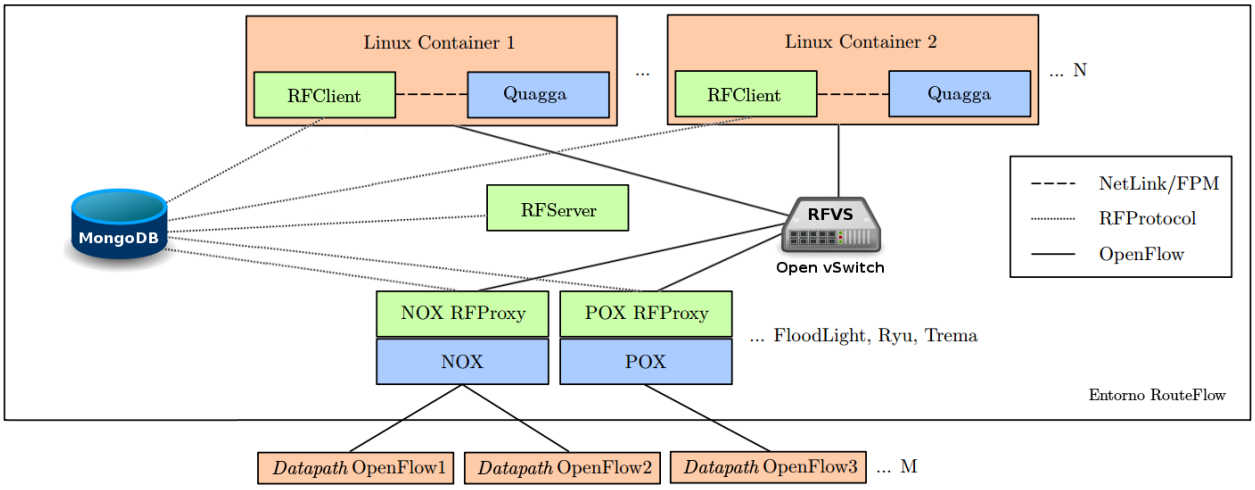


Figura 2.2: Arquitectura de RouteFlow.

2.2.1. La VM y el Cliente RouteFlow

Las *Virtual Machines* pueden ser cualquier sistema Linux capaz de ejecutar un motor de enrutamiento. Normalmente, se utilizan *Linux Containers* (LXC) [7] o otro tipo de contenedores virtuales ligeros. Los protocolos de red son implementados por motores de enrutamiento *Open Source*, como por ejemplo Quagga. Estos motores de enrutamiento envían actualizaciones de ruta al kernel[5] de Linux, haciendo que la máquina virtual Linux funcione como un *router*.

El RFClient es el componente de RouteFlow que interactúa con la VM. Su función es controlar el estado de la VM y comunicar al RFServer la información de configuración y el estado de la tabla de enrutamiento. Utiliza el protocolo Netlink[6] de Linux, o en versiones más actuales el protocolo *Forwarding Path Manager*[7] (FPM), para detectar cambios

[4] dispositivo de interconexión de redes que opera en la capa 2 del modelo OSI y de funcionalidad muy parecida a la de un *switch*. La diferencia más importante entre un *bridge* y un *switch* es que los primeros normalmente tienen un número pequeño de interfaces (de dos a cuatro), mientras que los *switches* pueden llegar a tener docenas.

[5] software que constituye una parte fundamental del sistema operativo. Es el principal responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o dicho de otra manera, es el encargado de gestionar recursos, a través de servicios de llamada al sistema.

[6] los *sockets* Netlink son interfaces del kernel de Linux que se utilizan para comunicaciones entre procesos (IPC) entre los procesos del kernel y los del espacio de usuario, y también entre los propios procesos del espacio de usuario, de manera similar a los *sockets* de dominio Unix.

[7] interfaz propuesta por el grupo Open Source Routing (OSR), que tiene por objetivo permitir a los motores de enrutamiento exponer la información de encaminamiento de manera independiente de la plataforma mediante un administrador.

en el host o en las tablas de rutas y luego enviar las rutas aprendidas al RFServer utilizando mensajes *RouteMod* del protocolo RouteFlow. Si es necesario, el RFClient puede resolver direcciones IP de *gateways*[8] intentando establecer una conexión TCP con ellas, forzando a la VM para que envíe una petición ARP[9] y así resolver la dirección MAC.

El RFClient también controla la configuración de las interfaces de la VM, activándolas o desactivándolas en función de los mensajes *PortConfig* del protocolo de RouteFlow y envía mensajes *PortRegister* al RFServer para informarle cuando una interfaz se ha activado.

Existen cuatro tipos de actualizaciones de ruta que puede recibir el RFClient: vecino nuevo, borrar vecino, ruta nueva y borrar ruta. Estas actualizaciones tienen unos atributos en común: un destino, un prefijo y un *gateway*. El destino puede ser una red o un *host*, el cual se especifica con el prefijo. Contra más pequeña es una red, más largo es el prefijo. Siguiendo esa lógica, el prefijo de un vecino será el prefijo más largo, ya que se refiere a un único destino. El *gateway* es el siguiente salto a realizar para acceder a las direcciones que especifiquen el destino junto con el prefijo, aunque este u otro router también podría ser el destino final. Los RFClient están escritos en C++ y corren como un *daemon* dentro de las VMs.

2.2.2. Proxy RouteFlow

El RFProxy es una aplicación OpenFlow que se encuentra en el NOS y que se encarga de administrar los *datapaths*. Interactúa con las plataformas de control de OpenFlow para intercambiar mensajes del protocolo OpenFlow con los *datapaths*, instalando y borrando flujos, y recibiendo información de sus estados.

Cuando los *datapaths* se unen a la topología, esto es detectado por el RFProxy el cuál envía esta información al RFServer. El RFProxy se encarga de notificar al RFServer el estado de los *datapaths* a través de mensajes *Datapath-PortRegister* del protocolo RFP cuando un puerto se vuelve activo y con mensajes *DatapathDown* cuando se pierde una conexión con un *datapath*.
Por otra parte, el RFProxy recibe del RFServer las actualizaciones de la red y las operaciones de configuración para luego enviárselas a cada *datapath* correspondiente.
Como ya se ha comentado anteriormente, el RFProxy administra la instalación y eliminación de los flujos de los *datapaths*. Para ello, el RFProxy escucha los mensajes *RouteMod* del RFServer, los convierte a mensajes del protocolo OpenFlow del tipo *Modificar Estado* y los envía a los *datapaths*.

El RFProxy es responsable de redirigir el tráfico del plano de control entre las VMs y los *datapaths*. Un *switch* virtual llamado RouteFlow Virtual Switch (RFVS) y controlado con OpenFlow (como el Open vSwitch) se conecta con cada puerto de la VM. El RFProxy funciona como controlador para este *switch*, así como para los *datapaths* RouteFlow. El RFVS solo tiene una entrada en su tabla de flujos, definida para que coincida con todo el tráfico y luego lo redirija al controlador. De la misma manera, los *datapaths* tienen flujos que redirigen todo el tráfico del protocolo de enrutamiento al controlador. Siempre que el RFProxy recibe un paquete desde el RFVS, lo reenvía a través del correspondiente puerto del *datapath*. De manera similar, el tráfico recibido desde los *datapaths* es reenviado a través del RFVS. Las asociaciones entre los puertos del RFVS y los *datapaths* son recibidas desde el RFServer como mensajes *DataPlane-Map* del protocolo RFP. Los *datapath* que desempeñan el papel de RFVS se identifican porque su ID empieza por el número "72667673". En las figuras 2.3 y 2.4 del apartado 2.5, puede verse por un lado el mapeo entre las VMs y el RFVS, y por la otra, el mapeo entre los *datapaths* y el RFVS.

Para actuar cómo un *router*, las reglas de OpenFlow se definen para que hagan coincidencia con las direcciones IP y MAC destino y lleven a cabo la reescritura de las direcciones MAC basándose en la información del próximo salto. Para los dispositivos con OpenFlow 1.3 se utilizan varias tablas de flujos y de grupos para instalar las entradas de la

[8] equipo informático configurado para dotar a las máquinas de una red de área local conectadas a él de un acceso hacia una red exterior.
[9] *Address Resolution Protocol* es un protocolo de comunicaciones de la capa de enlace de datos, responsable de encontrar la dirección de hardware (Ethernet MAC) que corresponde a una determinada dirección IP.

FIB (*Forwarding Information Base*)[10].

Actualmente existen varias versiones del RFProxy, dónde cada una de ellas utiliza una plataforma de control OpenFlow distinta. Algunas versiones están implementadas para los controladores NOX y POX, escritos respectivamente en C++ y Python, así como también existe una versión para el controlador Ryu, también escrita en Python. Existen otras versiones de RFProxy para otros lenguajes de programación y NOSs, pero aún están bajo desarrollo.

2.2.3. Servidor RouteFlow

El RFServer es una aplicación autónoma responsable de la lógica central del sistema, que conoce todo el estado de la red y que a diferencia del RFClient, está escrito en Python. El RFServer administra la topología de la red, encargándose de mapear los *datapaths* con las VMs y de enviar los mensajes del RFP (RouteFlow Protocol) entre las instancias de RFClient y de RFProxy.

Cuando un RFClient le envía una actualización de ruta, el RFServer puede realizar alguna operación sobre ésa ruta y luego debe enviarla al RFProxy adecuado. El RFServer también maneja el conjunto de reglas por defecto de un *datapath* para que reenvíe el tráfico del protocolo de enrutamiento hacia el controlador.

El RFServer se configura con un fichero comma-separated values (CSV[11]), especificando el mapeo entre los puertos de las VMs y los puertos de los *datapaths*, los cuales lee de una tabla de la base de datos MongoDB. En la tabla 2.1 se muestra la estructura del fichero CSV junto con unos valores de ejemplo. La primera línea solo sirve para hacer más entendible el fichero. Los campos que especifica son: el identificador (ID) de la máquina virtual `vm_id`, el puerto a mapear `vm_port` de dicha máquina, el ID del controlador `ct_id` a utilizar, el ID del *datapath* `dp_id` y su puerto `dp_port` a mapear. En el ejemplo se muestra el mapeo del puerto 1 del RFClient con ID 12A0A0A0A0A0 con el puerto 1 del *datapath* con ID 99. El campo `ct_id` ofrece la posibilidad de soportar múltiples controladores para que cada uno gestione una parte de la red, pero en este caso el campo tiene valor 0 indicando que solo se va a utilizar un controlador.

```
vm_id,vm_port,ct_id,dp_id,dp_port
12A0A0A0A0A0,1,0,99,1
```

Tabla 2.1: Fichero de configuración del RFServer.

El estado de los *datapaths* y las VMs de la red se almacena en una tabla de MongoDB llamada “RFTable”. Cuando se activan los puertos, tanto si es en el *datapath* como en la VM, un mensaje de registro (*PortRegister*) es enviado al RFServer, conteniendo información de los puertos (p.ej. direcciones MAC de cada puerto) que luego será almacenada en la RFTable. Cuando se han recibido los mensajes de registro de ambos elementos, el RFServer envía un mensaje del protocolo RFP de tipo *DataPlaneMap* al RFProxy informándole del mapeo entre los puertos del *switch* virtual (encargado de enlazar las VMs con los *datapaths*) y los del *datapath*. Cuando se confirma que el reenvío entre la VM y el *datapath* está operativo, el RFServer establece el estado de esos puertos a “activo”. Cuando recibe mensajes *DatapathDown* del protocolo RFP (indicando que se ha caído un *datapath*), el RFServer da instrucciones al RFClient para que desactive todas las interfaces de la correspondiente VM a través de mensajes del RFP de tipo *Datapath Config*.

El RFServer controla el estado de los *datapaths* enviando mensajes *RouteMod* al RFProxy: el RFServer genera mensajes *RouteMod* cuando un *datapath* aparece por primera vez, añadiéndole los flujos para redirigir el tráfico del protocolo de control hacia el controlador. El RFServer es también responsable de transferir los mensajes *RouteMod* recibidos

[10] tabla que se utiliza en el enrutamiento y funciones similares para encontrar la interfaz apropiada a la que la interfaz de entrada debe enviar un paquete, que asigna dinámicamente las direcciones MAC de los puertos. Es el mecanismo esencial que separa los *switches* de los *hubs*.

[11] tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea.

desde el RFClient hacia la correcta instancia de RFProxy, así como asegurarse de que contienen la información correcta sobre los *datapaths*.

2.2.4. RFMonitor

El RFMonitor es una aplicación encargada de monitorizar los controladores de la red, comprobando que todos funcionan correctamente, y de escuchar los mensajes IPC del protocolo RouteFlow que le indican si hay nuevos controladores a monitorizar. Dispone de un "diccionario de controladores" que contiene una lista con los controladores que está monitorizando junto con los detalles de cada uno, tales como su dirección IP y su puerto, así como el rol que desempeña.

A partir de la versión 1.2 del protocolo OpenFlow, se añade la opción de poder configurar varios controladores para el mismo *switch* OpenFlow. Estos controladores pueden desempeñar tres roles: maestro, par o esclavo. Los roles maestro y par tienen acceso absoluto sobre los *switches* que gestionan y pueden recibir todos los mensajes asíncronos del *switch* (p.ej. mensajes *Packet-in*). El *switch* puede hacer conexiones seguras con todos los controladores con rol par, pero sólo puede acceder a un solo controlador maestro. Los controladores que tengan un rol esclavo, sólo tienen permisos de lectura sobre los *switches* y tan sólo pueden recibir los mensajes asíncronos que informan del estado de los puertos de los *switches* [8].

Cuando se configura un controlador, se envía un mensaje *ControllerRegister* del protocolo RouteFlow al RFMonitor, el cual generará una instancia de sí mismo (un Monitor) asociada a ése controlador, para que se encargue periódicamente de comprobar que sigue activo. Para comprobar que el controlador sigue activo, el RFMonitor crea un *socket* y intenta conectarse al dispositivo y puerto en el que está escuchando el controlador. En el caso de que el controlador no respondiera, el RFMonitor borraría de su instancia la entrada y los datos correspondientes al controlador, y cesaría de comprobar su estado.

En el caso de que un controlador maestro dejara de estar disponible, el RFMonitor dispone de una lista de posibles candidatos a controlador maestro, que es actualizada cada vez que se añade un nuevo controlador o aumenta la carga de algún controlador existente. La elección del nuevo controlador maestro se hace en base al número de dispositivos que tenga asociados cada controlador, promoviendo a los que tengan más. Cuando se ha elegido al nuevo controlador maestro, el RFMonitor se lo notifica al RFProxy mediante un mensaje *ElectMaster*, quien a su vez se lo notificará al resto de dispositivos.

2.3. Protocolo RouteFlow (RFP)

El protocolo RouteFlow (RFP) proporciona una serie de mensajes IPC que utilizan los componentes de RouteFlow para comunicarse entre ellos. Estos mensajes transmiten el estado de la red, el mapeo entre las VMs y los *datapaths*, mensajes de configuración, y también los detalles de las rutas aprendidas por las VMs que deban ser añadidas a los *datapaths*. Los tipos de mensajes que proporciona son:

- ***PortRegister*:** Se utilizan para indicar que hay un puerto listo para usarse en una VM.
- ***PortConfig*:** Se utilizan para modificar la configuración de los puertos de una VM.
- ***DatapathPortRegister*:** Se utilizan para indicar que hay un puerto listo para usarse en un *datapath*.
- ***DatapathDown*:** Indican que un *datapath* deja de poder comunicarse con el controlador.
- ***VirtualPlaneMap*:** Se utilizan para informar al RFProxy del mapeo entre los puertos del *switch* virtual (RFVS) y los de la VM.
- ***DataPlaneMap*:** Se utilizan para informar al RFProxy del mapeo entre los puertos del *switch* virtual (RFVS) y los del *datapath*.

- ***RouteMod*:** Contiene información sobre las rutas aprendidas por la VM. Esta información es la que permitirá añadir las rutas a los *datapaths*. Este tipo es un tipo especial de mensaje, ya que es el encargado de abstraer las características del protocolo OpenFlow, proporcionando una interfaz de alto nivel para la configuración de las tablas de flujo.
- ***ControllerRegister*:** Contiene información de la instancia de un controlador. Se utiliza para notificar al RFMonitor la existencia de un controlador, y para que posteriormente almacene dicha información en su "diccionario de controladores".
- ***ElectMaster*:** Se utiliza para informar al RFProxy, que se ha elegido un nuevo controlador maestro. Esto sucede cuando un controlador maestro deja de estar disponible y se ha elegido uno nuevo de entre los controladores disponibles.

El formato de los mensajes OpenFlow generados por el RFProxy como traducción a los mensajes RouteFlow, consiste en un conjunto de campos de coincidencia y acciones que se especifican en estructuras de tipo/longitud/valor (TLV)[12]. El uso de este tipo de codificación permite que en el futuro se puedan definir nuevos tipos de mensajes sin tener que modificar su formato. Aun así, puede que no siempre se pueda realizar la implementación de cualquier tipo de mensaje. Por ejemplo, puede que la implementación de un RFProxy registre un error en el que se indique que no es capaz de codificar las estructuras TLV de esos mensajes RouteFlow en los mensajes OpenFlow. También podría darse el caso de que el RFProxy soportara el tipo de mensaje, pero el *datapath* no fuera capaz de llevar a cabo la acción que se le especifica. En ese caso, se generarían mensajes de error OpenFlow que notificarían que las acciones no han podido ejecutarse.

Los mensajes OpenFlow para modificar flujos incluyen campos fijos que se utilizan para la configuración de los flujos (p.ej. el tiempo que se deben mantener dichos flujos en las tablas de reenvío del *datapath*). Esos campos fijos pueden ser codificados en los campos de metadatos de los mensajes IPC de RouteFlow utilizando estructuras TLV.

En RouteFlow existen dos canales por los que transmitir los mensajes IPC: uno de ellos se encarga de la comunicación entre el RFClient y el RFServer, mientras que el otro, maneja las comunicaciones entre el RFServer y el RFProxy. El formato de los mensajes es el mismo para los dos canales. Si los mensajes de uno de los canales requieren de unas opciones concretas, se especifican mediante estructuras TLV. De esta manera se reduce el número de operaciones que se deben realizar para reenviar cada ruta. RouteFlow proporciona un script llamado `rfdb.py` que permite monitorizar los mensajes IPC que se transmiten a través de esos dos canales.

Los mensajes *RouteMod* son un tipo de mensajes especiales que mediante el campo `mod_type` permiten a los componentes de RouteFlow enviar mensajes tanto para añadir, modificar o borrar flujos. Estos mensajes también permiten utilizar unos tipos de opciones, de campos de coincidencia y de acciones opcionales, que los componentes del sistema pueden no estar obligados a soportar. Si en un mensaje se identifica un tipo opcional desconocido, se permite crear la regla; si por el contrario se detecta un tipo de acción, opción o campo de coincidencia obligatorio desconocido, se detiene la producción de paquetes y se registra un error. Esto garantiza que los tipos obligatorios se utilizan de manera consistente y ofrece la posibilidad de añadir componentes en experimentación junto con los componentes más antiguos, garantizando el correcto funcionamiento.

La estructura que tiene cada tipo de mensaje IPC, se detalla[9] en la figura F.1 del apéndice F. Como se puede observar en dicha figura, los mensajes *RouteMod* tienen una estructura distinta a la de los demás mensajes:

- **`mod`:** entero de 8 bits, que especifica si es de tipo RMT_ADD, RMT_MODIFY o RMT_DEL, para añadir, modificar o eliminar entradas de la tabla de flujos.
- **`id`:** entero de 64 bits, que indica el ID del switch destino.
- **`matches`:** lista de TLVs que describen con qué campos debe coincidir esta entrada (p.ej. direcciones IP, puertos, etc.).
- **`actions`:** lista de TLVs que describen las acciones que se deben llevar a cabo.

[12] forma de codificar los datos, de manera que haya información que pueda tener presencia opcional y longitud variable. Esta codificación se basa en tres campos: tipo (especifica el tipo de dato), longitud (indica cuantos bytes ocupa el valor) y valor (valor a codificar).

- **`options`:** lista de TLVs describiendo parámetros adicionales. Actualmente se utiliza para incluir la prioridad, los *timeouts* y el ID del controlador.

2.3.1. Comunicación entre Procesos (IPC)

RouteFlow utiliza el formato de datos JSON Binario (BSON)[13] junto con la base de datos MongoDB para implementar las comunicaciones IPC. Para enviar mensajes en C++, RouteFlow utiliza la librería oficial de MongoDB para codificarlos. El formato BSON soporta varios tipos de datos, incluyendo los enteros de longitud variable y datos binarios. Los atributos que consistan en un solo octeto o un vector que contenga un octeto en cada posición, serán codificados directamente a binarios BSON. Para los valores con varios octetos, como por ejemplo las direcciones IP, primero se ordenan los bytes al formato *big-endian*[14] y luego se codifican. Por otra parte, las TLVs de las direcciones IP contienen dos valores: la dirección y el prefijo. Para codificarlos, primero se ordenan los bytes de cada valor en formato *big-endian* y luego se concatenan. Todo este proceso se invierte para decodificar los BSON.

2.4. Modos de operación

La separación del plano de control del de reenvío permite un mapeo y funcionamiento flexible entre los elementos virtuales y sus contrapartes físicas [10]. Debido a eso, RouteFlow está diseñado para soportar un mapeo de M *datapaths* con N *routers* virtuales; por ejemplo un mapeo 1:1 consistiría en asociar un *router* virtual a cada *datapath*. Debido a esto, RouteFlow soporta tres modos principales de operación:

1. **División lógica:** Representa un mapeo 1:1 entre los *switches* hardware y los motores de enrutamiento virtuales, que refleja el sustrato físico de la red (número de puertos de los *switch*, conectividad) en el plano de control virtual. Se pueden definir dos submodos de operación dependiendo de si los mensajes del protocolo de enrutamiento se envían a través de la infraestructura física (es decir, el enrutamiento tradicional) o se mantienen en el plano virtual. Éste último permite separar los problemas de detección de topología física y mantenimiento, de la distribución del estado de enrutamiento. Esto permite una optimización de los protocolos de enrutamiento, dónde el mantenimiento de la conectividad se puede realizar en el plano de datos (p.ej. la detección de reenvío bidireccional), mientras que las actualizaciones de ruta (p.ej. anuncio del estado de un enlace) pueden ser enviadas dentro del dominio virtual.

2. **Multiplexación:** Este mapeo 1:N del sustrato físico al virtual representa el enfoque común para la virtualización del router, donde múltiples planes de control están funcionando simultáneamente e instalan sus FIBs independientes en el mismo hardware.

3. **Agregación:** Consiste en un mapeo M:1 de recursos hardware con instancias virtuales. Esta agrupación de un conjunto de *switches*, permite simplificar la ingeniería del protocolo de red de tal manera que los dispositivos o dominios vecinos puedan tratar al conjunto agregado como si fuera un único elemento. De este modo, el enrutamiento intradominio (dentro de la misma red) puede ser definido de forma independiente por software, mientras que el interdominio (entre distintas redes) o interzona de enrutamiento tradicional (p.ej. BGP) puede converger hacia una única unidad de control para propósitos de simplificación y centralización de la gestión, así como para la escalabilidad de la señalización.

[13] formato de intercambio de datos usado principalmente para su almacenamiento y transferencia en la base de datos MongoDB. Es una representación binaria de estructuras de datos y mapas. Un objeto BSON consiste en una lista ordenada de elementos. Cada elemento consta de un campo nombre, un tipo y un valor.

[14] formato que define el orden en que los bytes de un objeto son transmitidos o almacenados. En este formato el byte más significativo, que es el que contiene el bit más significativo (bit que está en la posición más alta dentro un número binario), se almacena primero y los siguientes bytes se almacenan por orden de importancia, siendo el byte menos significativo el que ocupe el último lugar.

2.4.1. Modo Agregación en detalle

La agregación de múltiples *switches* OpenFlow para crear un único *router* virtual lógico, permite crear un modelo *Platform-as-a-Service*[15] para la administración de la red [1]. De esta forma, se consigue un panel de control con lógica centralizada, que pese a ser una red de *switches* conectados entre sí, interactúa con el resto de la red como si fuera un único dispositivo.

Uno de los objetivos más importantes de ésta abstracción es reducir la configuración. Para la agregación, tan sólo hace falta el típico fichero de configuración del RFServer, junto con un fichero adicional para detallar los *Inter-Switch Links* (*Enlaces Entre Switches*) de los *datapaths* que forman la agregación.

Esta estructura al ser interpretada por las demás redes como un único elemento, oculta los detalles internos de la red, dándole libertad a la red interna para soportar los protocolos de enrutamiento o conmutación que se deseen, independientemente de los protocolos requeridos por la red "exterior", de manera similar a lo que ocurre con las redes que utilizan MPLS. [11] Con este modelo, el *router* es el único encargado de configurar las rutas recibidas, así como de instalar los flujos en los *switches* OpenFlow. Otras ventajas que esta estructura ofrece son la de instalar las entradas de flujo sólo en esos *datapaths* que vayan a enrutar el tráfico, evitando así aumentar el tamaño de las tablas de flujos de los *switches*, y la de eliminar la necesidad de utilizar protocolos de enrutamiento distribuidos en la red central, gracias a que el servicio de enrutamiento está centralizado.

Para permitir que la red de *datapaths* interconectados pueda tener cualquier topología arbitraria, se debe cumplir que todos los *datapaths* de la estructura tengan un camino hacia los demás *datapaths*, ya sea un camino directo o pasando por otros *datapaths*.

El reenvío entre *datapaths* se basa en el camino más corto contando el número de saltos. Para las topologías arbitrarias se generan *Label-Switched Paths* (LSP) entre cada par de nodos. Los LSP son circuitos virtuales[16] creados en redes de conmutación de paquetes[17], en las que los paquetes son reenviados en función de unas etiquetas que se añaden a los paquetes.

Para gestionar de forma más eficiente los *switches* centrales, se pueden utilizar múltiples controladores, cada uno encargado de gestionar una parte de la red. De esta manera, los controladores pueden tomar decisiones de enrutamiento de forma cooperativa, ofrecer un control más sencillo de la red, así como ofrecer más resistencia en caso de que uno de los controladores fallara.

El comportamiento de una VM que gestione un *datapath* debe ser el mismo que la de una VM que gestione varios *datapaths*, dado que no se necesita ningún conocimiento más aparte del que proporciona el fichero de configuración de los *Inter-Switch Links* (ISL). La lógica que permite este hecho reside en el RFServer, el cual controla la configuración de los ISLs y conoce la topología subyacente. Para asegurarse de que se mantiene el correcto comportamiento del reenvío, en cuanto el RFServer recibe los mensajes *RouteMod* del RFClient, los distribuye entre los *datapaths*.

Configuración y manejo de puertos

Como ya se ha comentado anteriormente, para configurar los ISLs se utiliza un fichero de configuración adicional, que al igual que para el RFServer, será de tipo CSV. El fichero describe el mapeo a realizar de los extremos de los ISLs con los correspondientes puertos de los demás *datapaths*. Este fichero también especifica las direcciones MAC de los

[15] modelo de una red virtualizada en la que el control de la infraestructura está separado del control del servicio (paradigma SDN) y el plano de control se gestiona con la abstracción de un sólo *router*, a diferencia de otros modelos dónde el plano de control es gestionado por una red virtual formado por múltiples *routers* virtuales. Con este modelo se consigue que el *router* se centre únicamente en su servicio, en vez de tener que preocuparse también de gestionar la red virtual.

[16] forma de comunicación mediante conmutación de paquetes en la cual la información o datos son empaquetados en bloques que tienen un tamaño variable a los que se les denomina paquetes de datos. En los circuitos virtuales, al comienzo de la sesión se establece una ruta única entre las entidades terminales de datos o los host extremos. A partir de aquí, todos los paquetes enviados entre estas entidades seguirán la misma ruta.

[17] método de envío de datos en una red de computadoras. Un paquete es un grupo de información que consta de dos partes: los datos propiamente dichos y la información de control, que indica la ruta a seguir a lo largo de la red hasta el destino del paquete.

puertos a mapear, ya que a diferencia de lo que sucede con los puertos externos, estos no pueden recibir información MAC del RFClient. Para almacenar esta información se crea una tabla nueva en la base de datos MongoDB. En la tabla 2.2 se muestra la estructura del fichero de configuración de los ISLs, con un ejemplo que detalla la asociación de dos *datapaths* gestionados por una VM con ID 12A0A0A0A0A0. Se puede observar que el puerto 2 del *datapath* con ID 0x99 está conectado al puerto 2 del *datapath* con ID 0x9A. En el fichero también se detallan las respectivas direcciones MAC de los puertos.

```
vm_id,ct_id,dp_id,dp_port,eth_addr,rem_ct,rem_id,rem_port,rem_eth_addr
12A0A0A0A0A0,0,0x99,2,66:FD:B5:BE:39:77,0,0x9a,2,6E:71:60:A9:B9:8D
```

Tabla 2.2: Fichero de configuración de los *Inter-Switch Links*.

Los mensajes *Datapath Port Registration* de los ISLs se manejan de forma similar a los mensajes externos del mismo tipo: cuando un mensaje *Datapath Port Registration* se recibe, las dos tablas de configuración (ficheros CSV) son consultadas; si existe una entrada ISL, se utiliza una segunda tabla de MongoDB para los estados de los puertos en vez de la RFTable.

Para el caso en el que el RFProxy informa al RFServer de que un *datapath* ha dejado de responder, el comportamiento de la red no se ve modificado, por lo que en esta situación, el RFServer enviará mensajes al RFClient para reiniciar los puertos de la VM. Entonces, el RFClient reiniciará los puertos de la VM que tenía asociados con el *datapath*, mientras que los caminos entre los *datapaths* restantes se recalcularán para seguir manteniendo la estructura en funcionamiento.

Los *datapaths* y los ISLs pueden activarse en cualquier orden y no hay razón alguna para suponer que lo harán de tal manera que se asegure la conexión entre cada *datapath*. Para controlar este hecho, el RFServer designa al primer *datapath* en activarse como el "*datapath* origen". Así cuando se activan los ISLs, solo se habilitarán si hay un camino entre uno de los *datapaths* y el "*datapath* origen". Cuando se desactiva un *datapath*, se comprueba que todos los *datapaths* activos siguen teniendo un camino hacia el denominado "origen". Si alguno de estos *datapaths* ha dejado de tener un camino hacia el "*datapath* origen", se desactiva. Si se da el caso de que el *datapath* desactivado es el denominado "origen", se elige arbitrariamente otro "*datapath* origen" de entre los *datapaths* activos.

Los caminos entre *switches*

Para asegurar un correcto manejo de los paquetes, cuando el RFServer recibe un mensaje *RouteMod*, debe actualizar el comportamiento de reenvío de cada *datapath* de la estructura.

Para el caso en el que la estructura utiliza una topología de mallado completo, tan sólo se requiere reenviar los paquetes directamente al *datapath* de salida, quien luego los reenviará al puerto correcto. Cuando el RFServer recibe un mensaje *RouteMod*, crea mensajes *RouteMod* adicionales para cada puerto externo de la estructura, así como para cada puerto del ISL del nodo de salida. Algunos de los flujos agregados al nodo de salida reenvían los paquetes directamente hacia el puerto de salida, mientras que otros los reenvían al puerto de salida a través del puerto del ISL apropiado.

Para topologías más complejas, tal y como se apuntaba en apartados anteriores, se requiere de la utilización de *Label-Switched Paths* entre los *datapaths* para prevenir bucles. Cuando un paquete entra en la estructura, se añade a su cabecera una etiqueta especificando el camino que debe tomar a través de la estructura. La etiqueta se elimina antes de que el paquete abandone la estructura de *switches*, devolviéndolo a su estado original.

Las redes con control centralizado deben tener cuidado al modificar el comportamiento de reenvío de la red, ya que es imposible garantir que las actualizaciones de las tablas de flujo de cada *datapath* sucedan simultáneamente. Cuando se envían actualizaciones a los *datapaths*, algunos las recibirán antes y las empezarán a utilizar, mientras otros aún

estarán utilizando los flujos antiguos. Esto puede provocar comportamientos de reenvío de la red impredecibles, incluido bucles. Con el uso de LSPs, los nuevos flujos utilizarán nuevas etiquetas que no harán coincidencia con flujos anteriores, por lo que los paquetes serán descartados.

Los LSPs ofrecen un mayor beneficio en el caso de que el camino anterior siga disponible. Por ejemplo, cuando el cambio de la topología ha sido causado por un ISL que se ha activado, el LSP antiguo puede permanecer en uso mientras los nuevos flujos se están instalando, para que de esta manera no se pierda tráfico durante el cambio. Para conseguir esto, los flujos se instalan utilizando el siguiente procedimiento: primero, los flujos para los nuevos LSPs se cargan en los *datapaths*; seguidamente, todas las rutas que estén cargadas en ése momento en los *datapaths* se modifican para utilizar el nuevo LSP; finalmente, se eliminan los flujos para el LSP antiguo.

Los flujos con LSP se cargan en los *datapaths* tan pronto como son calculados. Esto significa que cuando una ruta es aprendida, los nuevos flujos solo necesitan ser instalados en la fuente de cada LSP. Las rutas son añadidas simplemente cargando los flujos para cada puerto externo, añadiendo la etiqueta asociada con el camino y reenviando por el puerto adecuado.

Para instalar LSPs en un *datapath*, se añade un flujo por cada puerto del ISL, que hará coincidencia con la etiqueta de camino y con la dirección destino Ethernet correcta del puerto de entrada. El flujo actualiza las direcciones origen y destino Ethernet, intercambia la etiqueta con una que tenga el valor correcto para que haga coincidencia en el próximo salto y reenvía al puerto de salida correcto. Hay una excepción para el penúltimo salto en la red interior: se realizan los mismos pasos, pero en vez de intercambiar la etiqueta, se elimina.

Para reducir el número de flujos en cada *datapath*, los LSPs comparten etiquetas siempre que es posible. Todos los LSPs con el mismo *datapath* como destino que pasen por solo un *datapath*, utilizarán las mismas etiquetas una vez hayan pasado por el *datapath* intermedio. Normalmente, para los LSPs se utilizan etiquetas MPLS, aunque en su defecto también se pueden implementar utilizando etiquetas VLAN.

Cálculo del camino

Los *Label-Switched Paths* (LSP) se calculan cada vez que hay una modificación en la topología de la estructura, ya sea causada porque se ha activado un ISL o porque se ha recibido un mensaje *DatapathDown*. Para el cálculo se utiliza el algoritmo Dijkstra[18] con pesos iguales. Cuando se activa un ISL entre dos *datapaths* (el ISL debe ser necesariamente el camino más corto entre los dos *datapaths*), se actualizan los LSPs. También se comprueba si los LSPs existentes de los dos *datapaths* pueden ser mejorados utilizando el nuevo enlace. Cualquier sustitución de LSP que se quiera hacer, debe esperar en una cola de espera. Luego, se extrae la primera posición de esa cola y se comprueba si se pueden ampliar sus extensiones[19]. Cualquier *datapath* con un ISL que le conecte con el último *datapath* del nuevo LSP, compara el nuevo LSP con el LSP que ya tenía con el mismo destino. Si el LSP existente puede ser mejorado, se genera un remplazo del LSP, que será añadido a la cola. De esta manera, cada LSP es visitado desde el más corto hasta el más largo, y si se encuentra un LSP que pueda ser modificado, es debido a que el nuevo LSP tiene un camino más corto al que ya existía.

Cuando se recibe un mensaje *DatapathDown*, primero de todo se guardan las longitudes de todos los LSPs que pasen a través de ese *datapath* y luego se eliminan los LSPs. A continuación, para cada LSP eliminado, se buscan todos los LSPs con mismo destino y longitud, y se comprueba si pueden ampliarse sus extensiones. Esto sigue un procedimiento similar a cuando un ISL es activado. Cualquier LSP más corto que el camino eliminado, no necesita ser comprobado dado que cualquier extensión que tenga, deberá ser más corta que la LSP anterior.

[18] algoritmo para la determinación del camino más corto desde un vértice origen al resto de los vértices en un grafo con pesos en cada arista. El algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene.

[19] información adicional que algunos protocolos de enrutamiento encapsulan en sus mensajes, que puede ser utilizada por otros protocolos. De esta manera, algunos protocolos de enrutamiento pueden aprovechar esta información para reducir el número de mensajes que deba transmitir.

Resistencia

Para asegurar que la estructura continúa operando correctamente después de que algún *datapath* deje de estar accesible, es suficiente con recalcular y recargar los LSPs, a no ser que parte de la red quede anexionada del resto. Sin embargo, si más adelante el *datapath* vuelve a estar accesible, se le deberán instalar de nuevo todas las rutas que tenga aprendidas en ese momento la VM.

Cuando un *datapath* pasa a estar inactivo, las correspondientes interfaces de la VM son desactivadas por el RFClient. Entonces, la VM elimina las rutas de su tabla de enrutamiento que iban a través de esas interfaces y el RFClient envía mensajes *RouteMod* reflejando estos cambios. Los LSPs son recalculados automáticamente y así el resto de la estructura puede continuar funcionando correctamente.

Si el *datapath* vuelve a estar activo, se le debe actualizar con todas las rutas que estén cargadas en ese momento en la estructura. Esto debe ser realizado por el RFServer, ya que el RFClient no tiene conocimiento de la topología subyacente. Para ello, el RFServer necesita guardar todas las rutas a medida que las añade a los *datapaths* y borrarlas a medida que son eliminadas. Cuando se activa un *datapath*, el RFServer reenvía todas las rutas que estén activas en ese mismo momento, sin necesidad de modificar el RFClient.

Una manera de ofrecer más resistencia a la red, consiste en configurar varios controladores en un mismo *switch*. Como que el *datapath* sólo puede ser gestionado por un sólo controlador a la vez, se designa a uno de ellos como el controlador primario. De esta manera, si el controlador principal dejara de estar accesible, uno de los otros controladores pasaría a sustituirlo. Adicionalmente, se podría utilizar el mecanismo de OpenFlow de notificación de fallo de enlace, en vez de esperar a que lo notifique el protocolo de enrutamiento.

2.5. Reenvío de paquetes

2.5.1. Reenvío de paquetes en el plano de datos

Como ya se ha explicado en apartados anteriores, para simular el comportamiento de reenvío que tendría un *router* convencional operando por encima de *switches* OpenFlow, RouteFlow programa esos *switches* con unas reglas en las tablas de flujos para que simulen el enrutamiento convencional.

Este hecho obliga el controlador a encontrar un equilibrio entre rendimiento y flexibilidad cada vez que instala reglas de flujo en los *switches*. Manejar los flujos de paquetes de forma autónoma en los *switches* OpenFlow, aumenta el rendimiento del plano de datos, pero por otra parte, el procesamiento de paquetes queda limitado a las acciones que pueda soportar el *switch*. Por el contrario, enviar los flujos de paquetes al controlador, añade una penalización al rendimiento pero le permite procesar arbitrariamente los paquetes. El reenvío de paquetes IP basándose en sus direcciones IP destino, se puede lograr con las características que están disponibles en todos los *switches* OpenFlow. Gracias a esto, los *switches* OpenFlow pueden ser capaces de reenviar paquetes IP sin necesidad de involucrar al controlador.

RouteFlow utiliza reglas de reenvío proactivas, reactivas e interpretadas. Las reglas proactivas y reactivas, son instaladas para tráfico IP dirigido a un *host* o a un *router* virtual de una red remota. Las reglas proactivas se instalan en los *switches* OpenFlow de antemano y se utilizan para todos los destinos accesibles de la red que hayan sido aprendidos por los *routers*. Por otra parte, las reglas reactivas se instalan en los *switches* OpenFlow como reacción a la llegada de un paquete para el que no se disponía de una regla de coincidencia, y se utilizan para los *hosts* que estén directamente conectados al *switch* y para los *routers* que estén a un salto. Mientras que las reglas reactivas tienen un detalle a nivel de dirección IP, las reglas proactivas son para toda la subred. La tercera categoría de reglas que hay son las interpretadas, que reenvían todos los paquetes del flujo al controlador (p.ej paquetes ICMP). El reenvío interpretado sólo se utiliza para el tráfico de control *out-of-band*[20].

[20]tráfico de control de una red que debido a su contenido de carácter sensible, se transmite de manera que no se utilicen los mismos canales de transmisión que los del tráfico de datos.

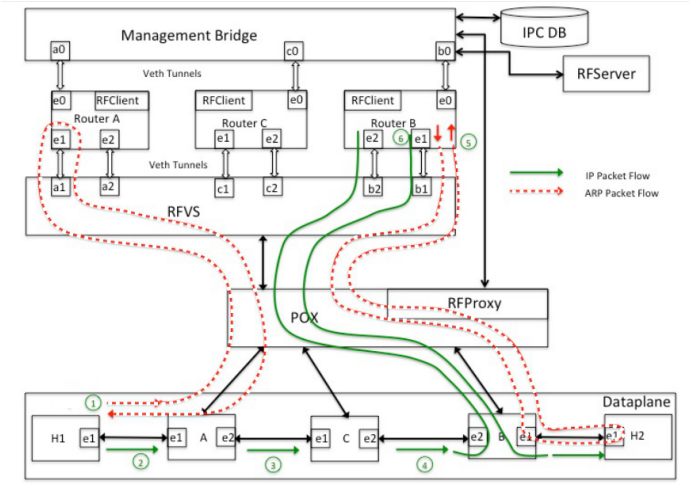


Figura 2.3: Reenvío de un paquete IP a través de una red gestionada con RouteFlow.

2.5.2. Reenvío de paquetes en el plano de control

Los *routers* virtuales que operan en el plano de control, necesitan intercambiar tráfico de control entre ellos mediante protocolos como OSPF, BGP, ICMP o ARP entre otros. RouteFlow utiliza flujos interpretados para reenviar estos paquetes hacia los *routers* virtuales correspondientes.

En RouteFlow, el tráfico de control de un flujo de paquetes debe ser reenviado a través del plano de control cada vez que hace un salto, incluso si el destino es otro *router* virtual. Esto no afecta al tráfico de control de protocolos como OSFP o ARP, ya que este tipo de mensajes suelen enviarse entre dispositivos vecinos y no suelen hacer largos recorridos por la red, pero por el contrario, otros protocolos del tráfico de control como ICMP o BGP que normalmente para llegar a su destino recorren largos tramos de la red, sí sufren una demora adicional en sus paquetes debido a que deberán ser transmitidos a través de varios *switches* OpenFlow (y como consecuencia hacer un viaje de ida y vuelta a sus *routers* asociados) antes de llegar a su destino.

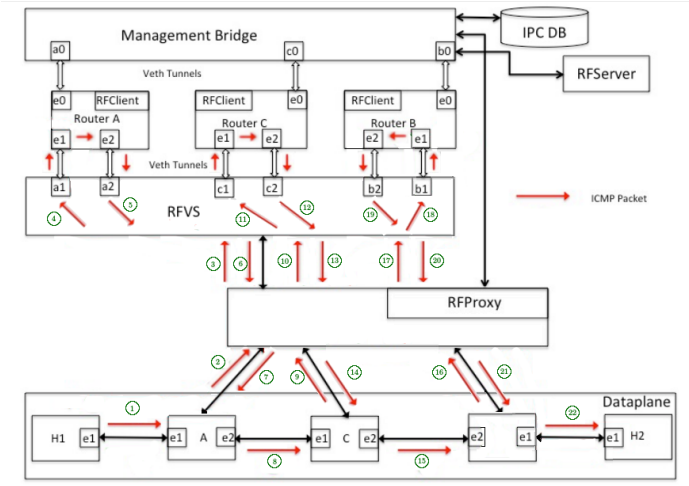


Figura 2.4: Reenvío de un paquete ICMP a través de una red gestionada con RouteFlow.

2.6. Prioridad de las entradas de flujo

Las tablas de flujos de OpenFlow empiezan a buscar coincidencias en las entradas con mayor prioridad, siguiendo con entradas cada vez menos prioritarias hasta encontrar una coincidencia; dicho de otra forma, se aplicará la primera regla que haga coincidencia con mayor prioridad. Esto difiere del enrutamiento convencional, en el que se utiliza la entrada con el prefijo más largo. RouteFlow implementa un mecanismo de cálculo de prioridad basado en la longitud de la máscara de subred, de manera que los flujos más específicos son utilizados antes que las reglas de flujo genéricas. Este mecanismo utiliza un multiplicador por cada bit de la máscara de subred en las reglas de flujo proactivas y reactivas, acompañado de un nivel de prioridad base que asigna a cada flujo en función del tipo de tráfico que transporte. Si se intentan hacer coincidencias de manera más granular con subredes del plano de datos, el factor de multiplicación se verá incrementado.

Los niveles de prioridad base que define el mecanismo son:

- **`PRIORITY_LOWEST` (0):** Es la prioridad más baja que se puede definir (valor 0). Se utiliza por ejemplo para definir la prioridad de una entrada de flujo *miss-table*.
- **`PRIORITY_LOW` (16400):** Esta prioridad se utiliza para las entradas de flujo que hagan referencia al tráfico de rutas.
- **`PRIORITY_HIGH` (32800):** Este nivel de prioridad se utiliza para las entradas de flujo del tráfico de control que deba ser reenviado al controlador.
- **`PRIORITY_HIGHEST` (49200):** Esta prioridad actualmente no se utiliza, pero puede ser usada para definir reglas *firewall*.

RouteFlow utiliza el nivel de prioridad predefinido `HIGH` para las reglas de flujo interpretadas utilizadas para los protocolos RIP, ICMP, ARP, IPv6, BGP, LDP y OSPF. Debido a que estos paquetes deben ser siempre enviados al controlador, la prioridad de estos flujos también debe ser superior respecto a otros, aunque por otra parte, estos paquetes van a demorarse a causa del procesamiento en el plano de control.

2.7. Tamaño de las tablas de flujos y requisitos de memoria

El tamaño de la tabla de enrutamiento de un *router* depende de la topología de la red y del rol que lleve a cabo el router en esa red (p.ej. *router* de acceso, *router* extremo, *router* del núcleo, etc.). Se ha observado como una sola ruta en un *router* virtual es traducida por RouteFlow en múltiples reglas en los *switches* OpenFlow. La razón de esta relación de uno a muchos, es la utilización del número del puerto de entrada y la dirección MAC del nodo destino como campos de coincidencia en las tablas de flujos.

Normalmente, los *routers* de acceso tienen una alta densidad de puertos, mientras que los *routers* del núcleo tienen grandes tablas de enrutamiento. Los *switches* OpenFlow actuales son capaces de manejar tan solo unos miles de flujos, mientras que los *routers* que se utilizan en el núcleo o en los extremos de la red de un proveedor de servicios deben manejar cientos de miles de rutas. Otro problema añadido es el hecho de añadir un nuevo *switch* a una red en funcionamiento, ya que puede incrementar repentinamente el número de mensajes *Flow Mod* de OpenFlow, y debido a eso congestionar el enlace entre el controlador y los *switches* OpenFlow.

Capítulo 3

Creación de una red virtual gestionada con RouteFlow

3.1. Introducción

En esta sección se procederá a la creación y posterior análisis de una red gestionada con RouteFlow. La red constará de cuatro *hosts* que estarán conectados entre sí a través de una red formada por cuatro *switches* virtuales en el plano de datos y cuatro *routers* virtuales en el plano de control. A continuación, en la figura 3.1 se puede observar la topología que tendrá dicha red:

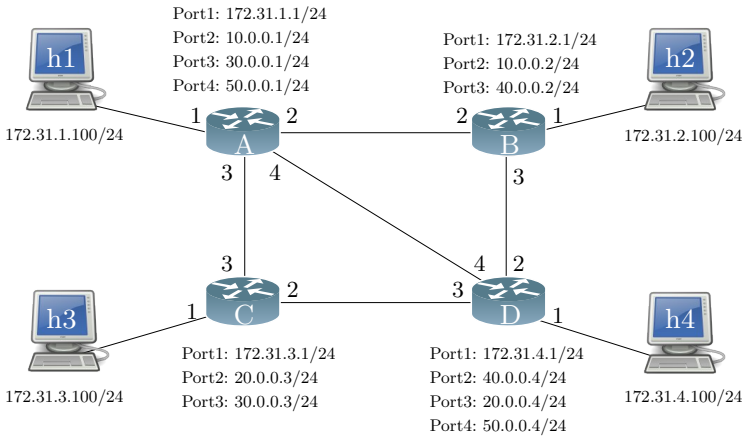


Figura 3.1: Topología de la red virtual.

Para la realización de la red y su posterior estudio, se utilizará un entorno virtualizado con *VirtualBox* que utilizará un sistema operativo Ubuntu 14.04 de 64 bits. En este entorno de trabajo se utilizarán distintas herramientas de virtualización, como por ejemplo LXC para crear los clientes RouteFlow y *mininet* para crear la red virtual que representará el plano de datos. Para la configuración de las interfaces se utilizará el motor de enrutamiento Quagga, y para gestionar los *switches* se utilizará un controlador llamado Ryu [7].

El proyecto RouteFlow está alojado en un repositorio de versiones llamado Git, dónde los creadores alojaron el código fuente del proyecto y este ha ido proliferando en varias "ramas" paralelas del mismo repositorio, dónde distintas personas de la comunidad *Open Source* han ido actualizando y manteniendo el proyecto. Para la realización de la red analizada en este documento, se ha optado por utilizar la "rama" del repositorio que más actualizaciones ha recibido y tiene más mantenimiento. Dicha rama recibe el nombre de "Vandervecken".

3.2. Preparación del entorno

3.2.1. Descarga de RouteFlow y herramientas de virtualización

Como ya se ha comentado anteriormente, RouteFlow está alojado en un repositorio de versiones llamado Git, por lo que para poder utilizar RouteFlow, se deberá de descargar primero dicha herramienta y con ella luego descargar RouteFlow.

Para su correcto funcionamiento, RouteFlow tiene unas dependencias que también deberán ser instaladas (entre ellas la base de datos MongoDB y el Open vSwitch). Todas las herramientas y dependencias, se pueden descargar mediante la ejecución del siguiente comando en un terminal Linux:

```
$ sudo apt-get install build-essential git libboost-dev \
  libboost-program-options-dev libboost-thread-dev \
  libboost-filesyste\vspace{5mm}m-dev iproute2 openvswitch-switch \
  mongodb python-pymongo
```

Una vez se complete la descarga, se puede proceder a la descarga de RouteFlow mediante Git. Para ello, primero se debe de navegar con un terminal al directorio del PC dónde se quiera instalar RouteFlow, y luego descargarlo mediante el siguiente comando de Git:

```
$ git clone -b vandervecken git://github.com/routeflow/RouteFlow.git
```

En este punto se habrá descargado el código fuente de RouteFlow en un directorio con el mismo que se habrá creado.

Para crear la red virtual del plano de datos, se va a utilizar una herramienta de virtualización llamada *mininet* [7]. Los creadores de esta herramienta, proporcionan una imagen[1] de una máquina virtual, que ya viene preparada para su uso. En la realización de este proyecto se ha utilizado la versión de *mininet* 2.0. Para montar esta máquina virtual, se utilizará la misma herramienta de virtualización que para crear el entorno de trabajo (*VirtualBox*). Esta máquina virtual hará de contenedor de la red del plano de datos. Dicha red estará formada por cuatro *switches* Open vSwitch y cuatro *hosts*, tal y como se ha podido observar en la figura 3.1.

Una vez montada la máquina virtual de *mininet* (que recibirá el nombre de “mininet”), se debe de conectar con el entorno de trabajo. Para ello se ejecuta el siguiente comando que creará una red interna de VirtualBox, llamada “intnet“:

```
$ vboxmanage dhcpserver add --netname intnet --ip 10.0.1.1 --netmask 255.255.255.0 \
--lowerip 10.0.1.100 --upperip 10.0.1.200 --enable
```

El comando anterior debe ejecutarse en un terminal del PC anfitrión, es decir fuera de VirtualBox (ni en la máquina virtual del entorno de trabajo, ni en la máquina virtual dónde se aloja *mininet*) y lo que hace es crear una red dónde el rango de direcciones IP irá desde la dirección IP 10.0.1.100 hasta la 10.0.1.200. Para este proyecto la dirección IP del entorno de trabajo será la 10.0.1.100 y la de la máquina virtual con *mininet* será la 10.0.1.102.

Seguidamente desde VirtualBox se deben configurar las interfaces de las máquinas virtuales (la del entorno de trabajo y la de *mininet*) para que estén conectadas a la red ”intnet“. Para hacerlo, primero se deben tener ambas máquinas

[1] archivo informático donde se almacena una copia o imagen exacta de un sistema de archivos o ficheros de un disco óptico, normalmente un disco compacto (CD) o un disco versátil digital (DVD). Algunos de los usos más comunes incluyen la distribución de sistemas operativos, por ejemplo: GNU/Linux, BSD y Live CD.

paradas, o de lo contrario VirtualBox no permitirá modificar sus parámetros de configuración. En el momento en que las máquinas virtuales ya no estén funcionando, se hace clic con el botón derecho del ratón sobre una de las máquinas virtuales y se accede a ”Configuración“.

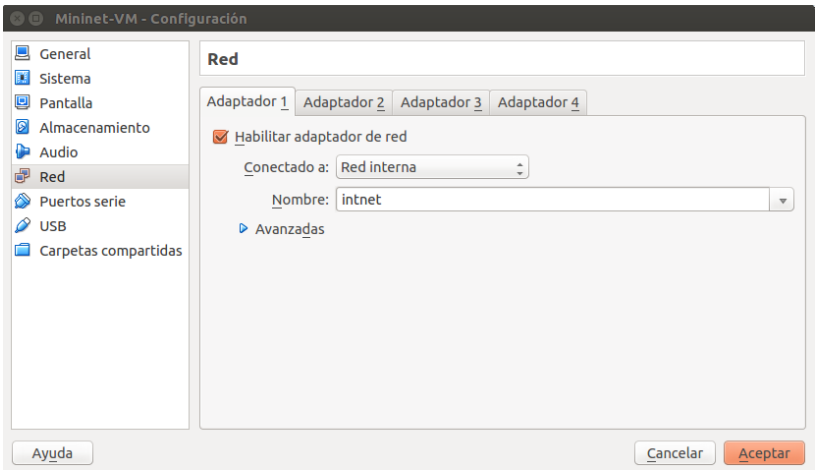


Figura 3.2: Configuración red interna de las máquinas virtuales.

Una vez hecho eso, se abre una ventana con las distintas secciones de configuración de la máquina virtual. El siguiente paso es acceder al apartado ”Red“ y seleccionar la opción ”Red interna“ del desplegable que aparece. Automáticamente habrá aparecido un nuevo desplegable con el nombre de la red creada anteriormente. De no ser así, se abre el desplegable y se selecciona el nombre correcto. Una vez esté configurado tal y como se muestra en la figura 3.2, se pulsa ”Aceptar“ y se realiza lo mismo con la otra máquina virtual.

Una vez terminado esto, las dos máquinas ya estarán conectadas entre sí y se deberán poner en marcha de nuevo.

3.2.2. Compilar componentes RouteFlow y generación de los elementos de la red

De nuevo en el entorno de trabajo, dentro del directorio RouteFlow existe una carpeta llamada `rftest` que contiene unos *scripts* para poder probar RouteFlow, así como algunos ficheros de configuración para generar los contenedores virtuales LXC. Estos ficheros de configuración de LXC son concretamente un script llamado `create`, y una carpeta llamada `config` que contiene los ficheros con la configuración de cada contenedor LXC.

El *script* `create` lo que hace es crear una carpeta en el directorio `/var/lib/` llamada `lxc` para que allí se almacenen todos los contenedores Linux. A continuación crea un contenedor genérico llamado `base`, cogiendo como plantilla el sistema de ficheros de Ubuntu. Este servirá de base para generar los demás contenedores. Seguidamente, en el directorio raíz de ese contenedor, instala las herramientas que vayan a ser necesarias (por ejemplo Quagga) para que el contenedor pueda desempeñar el papel de un dispositivo de red. Para finalizar, el `script` clona `base` para crear los nuevos contenedores a partir de los ficheros de configuración del directorio `config`. Estos contenedores representarán los distintos elementos de la red.

El siguiente paso a realizar, es compilar el código fuente y generar los contenedores virtuales. Para ello se deberá de ejecutar el *script* `build.sh` que hay en el directorio RouteFlow:

```
$ ./build.sh -ci ryu
```

En el comando anterior se le especifica al *script* que el controlador que se va a utilizar para gestionar los *switches* de la red será Ryu. Dentro de este *script* se utilizan varios ficheros de configuración y otros *scripts* que vienen en el directorio RouteFlow para compilar los componentes de RouteFlow (entre ellos el RFClient) e instalar las herramientas para la virtualización y enrutamiento (p.ej. LXC, Quagga, etc.). El *script* también se encarga de instalar las dependencias que falten para la configuración seleccionada (en este caso instalar el controlador Ryu).

Para la realización de estos pasos, en este proyecto se han necesitado de ciertas modificaciones en los *scripts* que se llaman en `build.sh`, incluso en este último. Estas modificaciones se detallan en el Apéndice E.

3.2.3. Creación de la red virtual

Generar los contenedores LXC

Una vez se tienen los componentes RouteFlow preparados para ser utilizados, se procede a crear la red virtual que va a ser administrada con RouteFlow. Para ello, lo primero que hay que hacer es generar los contenedores virtuales que representarán las distintas instancias del RFClient. Para generar dichos contenedores, se deben crear los ficheros de configuración correspondientes, con información como por ejemplo las tarjetas de red que vaya a tener cada contenedor o cómo van a ser montados dichos contenedores en el sistema de ficheros del PC. Estos ficheros, deben alojarse en el directorio `/rftest/config` de la carpeta RouteFlow. (Para más información consultar la documentación de LXC.)

Para generar los contenedores de la red de este proyecto, se han utilizado algunos de los ficheros de muestra que vienen incorporados con RouteFlow. Concretamente, los ficheros de la carpeta `config`: `rfvmA`, `rfvmB`, `rfvmC` y `rfvmD`. En el Apéndice D se pueden consultar dichos ficheros de configuración.

Una vez se tienen los ficheros de configuración de los contenedores LXC, se debe proceder a generarlos. Para ello, se utilizará el *script* `create`, que se nos proporciona en la misma carpeta `rftest`. Este lo que hará será generar los contenedores virtuales a partir de los ficheros de configuración que se hayan creado en la carpeta `config` e instalará en cada contenedor las herramientas que vayan a ser necesarias, como por ejemplo Quagga. Para el caso de estudio de este proyecto, no van a ser necesarias más herramientas de las que ya instala el *script*. Si fuera necesario instalar herramientas adicionales en los contenedores, se deberían modificar las líneas de código del *script* dónde se especifican las dependencias necesarias. Para ejecutar el *script*, tan solo hay que ejecutar el siguiente comando en un terminal Linux desde el directorio `rftest`:

```
# ./create
```

Creación del plano de datos

Como ya se ha explicado al principio de esta sección, para la creación de la red del plano de datos, se utilizará la herramienta llamada *mininet*. Esta herramienta necesita un fichero de configuración en el que se especifica la topología de la red y como están conectados entre sí los elementos de dicha red. Para el proyecto se utilizará un fichero en *python* como el de la figura F.2 del apéndice F.

Este fichero recibirá el nombre de `topo-4sw-4host.py`. Tal y como se apuntaba anteriormente, en este fichero se especifican los elementos que deberá tener la red, con qué nombre los identificará *mininet* y cómo estarán conectados entre sí los dispositivos. Además de este fichero, se utilizará un archivo adicional que contendrá unos comandos que deberán ser ejecutados por *mininet* para indicar a cada *host* su ruta de envío por defecto. Dicho fichero se llamará `ipconf` y contendrá lo siguiente:

```
1 h1 route add default gw 172.31.1.1
2 h2 route add default gw 172.31.2.1
3 h3 route add default gw 172.31.3.1
4 h4 route add default gw 172.31.4.1
```

Estos ficheros deberán estar en la máquina virtual "mininet". Si se han generado en el entorno de trabajo, se deberán copiar en los directorios que se especifican a continuación:

```
$ scp topo-4sw-4host.py mininet@10.0.1.102:/home/mininet/mininet/custom
$ scp ipconf mininet@10.0.1.102:/home/mininet
```

Con esto *mininet* ya estará configurada para iniciar la red virtual del proyecto.

3.3. Puesta en marcha de la red virtual

Entre los ficheros que provee RouteFlow hay un par de *scripts* que sirven de ejemplo para ejecutar las redes virtuales de prueba de RouteFlow. Para la ejecución de la red virtual de este proyecto se ha modificado uno de estos ficheros, concretamente el llamado `rftest2`, para adaptar el código al entorno de trabajo del proyecto y también aplicar alguna mejora. Por ejemplo, una de las modificaciones realizadas, ha sido adaptar el código para que utilizara correctamente ciertas herramientas que en el *script* de muestra estaban desactualizadas.

En el apéndice E pueden consultarse las modificaciones y mejoras realizadas, junto con el código del *script* de ejecución de la red virtual, al que se ha llamado `RFtfg`. Debido a que este fichero debe llamar a los distintos componentes de RouteFlow, se alojará en el mismo directorio que el *script rftest2*, ya que de esta manera se aprovechan los *paths* definidos en el *script* de ejemplo y así no hay necesidad de modificarlos.

Para iniciar la red virtual hay que ejecutar el fichero `RFtfg` descrito anteriormente:

```
# ./RFtfg
```

El fichero solo puede ejecutarse con permisos de administrador, ya que de lo contrario, algunos comandos de los que se utilizan en el *script* no podrían funcionar correctamente. A continuación se detallan las acciones que lleva a cabo el *script*:

1. Limpia los posibles restos de ejecuciones anteriores. Para ello, llama a una función definida en el mismo *script* que se encarga de detener los contenedores y procesos del *script* que pudieran estar aún activos de alguna otra ejecución.
2. Configura el *bridge* `lxcbr0` que será el encargado de conectar las interfaces de los contenedores Linux con la base de datos MongoDB.
3. Configura la base de datos MongoDB para que escuche al *bridge* `lxcbr0` y de esta manera pueda conectarse a la arquitectura de RouteFlow.
4. Inicia el RFServer con el fichero de configuración indicado. El fichero utilizado en este proyecto puede consultarse en el apéndice D.
5. Crea dentro de cada contenedor el directorio `/rootfs/opt/rfclient`, dónde copia el binario de RFClient. También crea un *script* llamado `run_rfclient.sh` que iniciará el servicio Quagga de cada contenedor y guardará en un *log* la salida del RFClient. Finalmente, inicia todos los contenedores Linux que harán de clientes RouteFlow.
6. Inicia el controlador Ryu y el módulo RFProxy para ése controlador, y espera a que los puertos estén operativos.
7. Genera el plano de control de la red. Para ello crea el *bridge* `dp0` que tendrá el papel de RFVS, y le añade los puertos necesarios para comunicarse con cada contenedor que haga de cliente. Finalmente le asigna al *bridge* un ID *datapath*, el controlador encargado de gestionarlo y le indica que el protocolo de OpenFlow que deberá utilizar es el 1.3.

8. Finalmente, muestra un mensaje de ayuda en el que se indica el comando que se debe de ejecutar en *mininet*.

Una vez se haya ejecutado el *script* y se hayan realizado los pasos anteriores, la red virtual aún no estará operativa, ya que aún le faltará un plano de datos sobre el que operar. Para ello, en la máquina virtual "mininet" se debe de ejecutar el comando `mn`, el cual iniciará la aplicación *mininet* con la topología de la red del plano de datos. El comando a ejecutar es el siguiente:

```
# mn --custom mininet/custom/topo-4sw-4host.py --topo=tfgnet" --switch ovs,protocols=OpenFlow13"
--controller=remote,ip=[host address],port=6633"--pre=ipconf"
```

En ese comando, se le indica a *mininet* qué topología y tipo de *switchs* debe utilizar para la virtualización, qué controlador utilizarán los *switchs*, y un fichero de preconfiguración `ipconf`. Una vez ejecutado el comando, aparece en el terminal la interfaz de *mininet*, en la que se pueden ejecutar distintos comandos de la aplicación. Para comprobar que la red virtual se ha generado correctamente y que el plano de control se comunica correctamente con el plano de datos, se ejecuta el comando de *mininet* llamado `pingall`, que como el nombre indica hará que cada *host* de la red envíe un mensaje ICMP de tipo *ping*[2] a los demás *hosts*. Si tras ejecutar el comando una o dos veces, no hay paquetes perdidos, significa que la red virtual se ha generado correctamente.

El resultado es la red virtual que se observa en la figura 3.3:

3.4. Estudio de la red virtual

En esta sección se pretende dar a conocer el funcionamiento de la red virtual gestionada por RouteFlow creada anteriormente. En el siguiente estudio, se utilizarán varias herramientas para poder ver desde distintos puntos de vista las comunicaciones que hay entre los elementos de la red. Algunas de las herramientas que se utilizarán venían proporcionadas junto a RouteFlow, mientras que otras como Wireshark son de terceros.

3.4.1. Tráfico generado en la red

Tal y como se ha explicado en apartados anteriores, la red virtual utiliza el motor de enrutamiento Quagga junto al protocolo OSPF. En este apartado, se observa el tráfico de paquetes que circula por cada una de las interfaces que tiene un RFClient. Para este estudio se ha escogido la máquina virtual `rfvmA`, la cual dispone de cuatro interfaces. Cada una de estas interfaces se ha monitorizado mediante la herramienta Wireshark.

Dichas interfaces se puede observar en la figura 3.1, dónde cada una de ellas conecta la máquina virtual con una red distinta. Esto permitirá ver dos tipos de tráfico de paquetes: uno que circulará por tres de las interfaces y en el que se podrá observar cómo los *routers* intercambian información mediante OSPF, y otro en el que solo se observará el tráfico con destino al *host* al que tiene conexión directa.

En el momento en el que las máquinas virtuales RFClient empiezan a funcionar, el protocolo OSPF de cada máquina empieza a mandar paquetes HELLO para informar a los demás *routers* de la red de su presencia. No es hasta que se inicia el plano de datos mediante *mininet*, que los demás *routers* pueden recibir dichos paquetes, ya que hasta entonces no existe un medio "físico" para transmitirlos. Una vez el plano de datos está funcional y los demás *routers* han recibido el paquete HELLO de otro *router*, se inicia el intercambio de información entre *routers* para conocer la topología de la red, y así poder calcular el camino más corto hacia los posibles destinos de la red. En la figura 3.4 se puede ver el intercambio de información que hay entre la máquina virtual `rfvmA` y la máquina `rfvmB` a través del puerto número '2' de `rfvmA`.

[2]utilidad diagnóstica en redes de computadoras que comprueba el estado de la comunicación del *host* emisor con uno o varios equipos remotos de una red IP por medio del envío de paquetes ICMP de solicitud (ICMP Echo Request) y de respuesta (ICMP Echo Reply). Mediante esta utilidad puede diagnosticarse el estado, velocidad y calidad de una red determinada.

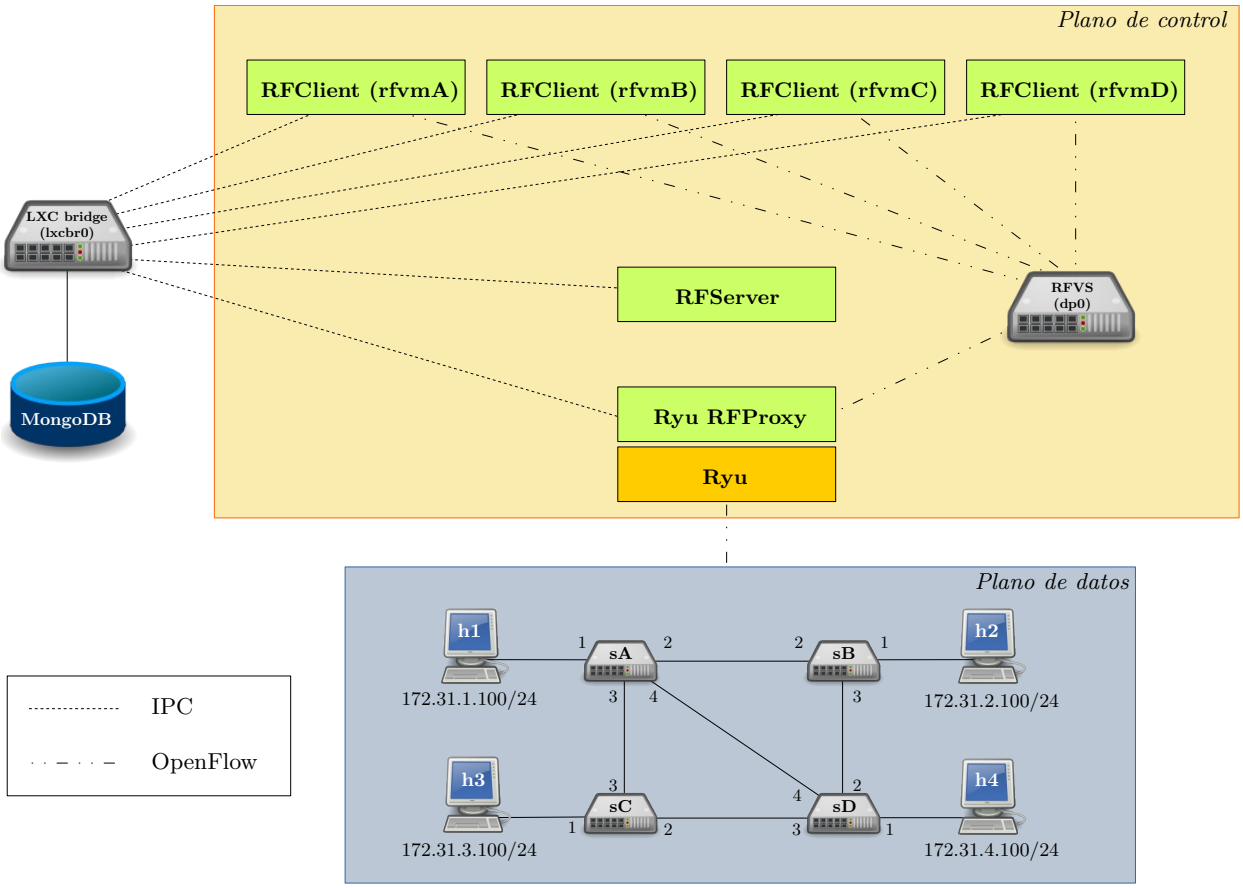


Figura 3.3: Arquitectura RouteFlow de la red virtual.

En esta imagen se puede observar como ambas máquinas utilizan el mismo puerto para conectarse a esa red. Esto se puede deducir al observar la dirección MAC de cada máquina, ya que han sido definidas de tal manera que aparezca la letra de la máquina virtual a la que pertenece esa interfaz y el número del puerto que utiliza (por ejemplo, la dirección MAC a2:a2:a2:a2:a2:a2 que aparece, corresponde a la máquina virtual `rfvmA` y a la interfaz que utiliza el puerto '2'). También se observan las direcciones IP de cada máquina, siendo la dirección 10.0.0.1 correspondiente a la máquina virtual `rfvmA` y la dirección 10.0.0.2 a la máquina `rfvmB`. La dirección IP 224.0.0.5 corresponde con la dirección *multicast* que utiliza el protocolo OSPF para que los *routers* se envíen mensajes de actualización entre ellos.

Como puede verse en las figuras 3.6 y 3.7, sucede lo mismo para las interfaces `rfvmA.3` y `rfvmA.4`. A diferencia que las anteriores, en la interfaz `rfvmA.1` no aparece tráfico de paquetes del protocolo OSPF, ya que en la red a la que está conectada la interfaz no existe ningún otro *router* que utilice ese protocolo de enrutamiento. Es por ello, que el protocolo OSPF etiqueta a esa red como un *stub*, lo que significa que los routers no pueden enviar paquetes OSPF a través de esa red. Esto puede observarse en la figura 3.5 donde se muestran algunos campos de un paquete OSPF capturado en la interfaz `rfvmA.2` y en donde se indica que la red 172.31.1.0 es un *stub*.

No.	Time	Source	Destination	Protocol	Length	Info
461	72.507134006	02:b2:b2:b2:b2:b2	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
462	72.507164006	02:a2:a2:a2:a2:a2	02:b2:b2:b2:b2:b2	ARP	42	10.0.0.1 is at 02:a2:a2:a2:a2:a2
465	72.523179006	10.0.0.2	10.0.0.1	OSPF	66	DB Description
466	72.523529006	10.0.0.1	10.0.0.2	OSPF	66	DB Description
467	72.523558006	10.0.0.1	10.0.0.2	OSPF	86	DB Description
468	72.585473006	10.0.0.2	10.0.0.1	OSPF	86	DB Description
469	72.585634006	10.0.0.1	10.0.0.2	OSPF	66	DB Description
470	72.585656006	10.0.0.1	10.0.0.2	OSPF	70	LS Request
471	72.615480006	10.0.0.2	10.0.0.1	OSPF	70	LS Request
472	72.615665006	10.0.0.1	224.0.0.5	OSPF	134	LS Update
473	72.615933006	10.0.0.2	224.0.0.5	OSPF	122	LS Update
474	72.616108006	10.0.0.1	224.0.0.5	OSPF	134	LS Update
475	72.663200006	10.0.0.2	224.0.0.5	OSPF	154	LS Update
476	72.849953006	10.0.0.1	224.0.0.5	OSPF	134	LS Update
477	72.963320006	10.0.0.2	224.0.0.5	OSPF	78	LS Acknowledge
478	72.966765006	10.0.0.1	224.0.0.5	OSPF	122	LS Update
479	72.966789006	10.0.0.1	224.0.0.5	OSPF	134	LS Update
480	72.985115006	10.0.0.1	224.0.0.5	OSPF	94	LS Update
495	73.057857006	10.0.0.2	224.0.0.5	OSPF	82	Hello Packet

Figura 3.4: Tráfico OSPF capturado en la interfaz `rfvmA.2`.

```
Advertising Router: 30.0.0.1 (30.0.0.1)
LS Sequence Number: 0x8000000e
LS Checksum: 0x7ca8
Length: 72
▶Flags: 0x00
Number of Links: 4
▶Type: Stub     ID: 172.31.1.0      Data: 255.255.255.0   Metric: 10
▶Type: Transit  ID: 10.0.0.2        Data: 10.0.0.1        Metric: 10
▶Type: Transit  ID: 30.0.0.3        Data: 30.0.0.1        Metric: 10
▶Type: Transit  ID: 50.0.0.4        Data: 50.0.0.1        Metric: 10
```

Figura 3.5: Paquete OSPF capturado en la interfaz `rfvmA.2`.

No.	Time	Source	Destination	Protocol	Length	Info
482	72.849741006	02:a3:a3:a3:a3:a3	Broadcast	ARP	42	Who has 30.0.0.3? Tell 30.0.0.1
483	72.906545006	02:c3:c3:c3:c3:c3	02:a3:a3:a3:a3:a3	ARP	42	30.0.0.3 is at 02:c3:c3:c3:c3:c3
484	72.906558006	30.0.0.1	30.0.0.3	OSPF	66	DB Description
485	72.913453006	30.0.0.3	30.0.0.1	OSPF	66	DB Description
486	72.913669006	30.0.0.1	30.0.0.3	OSPF	126	DB Description
487	72.942021006	30.0.0.3	30.0.0.1	OSPF	86	DB Description
488	72.942113006	30.0.0.1	30.0.0.3	OSPF	66	DB Description
489	72.942144006	30.0.0.1	30.0.0.3	OSPF	70	LS Request
490	72.966202006	30.0.0.3	30.0.0.1	OSPF	94	LS Request
491	72.966334006	30.0.0.1	224.0.0.5	OSPF	226	LS Update
492	72.966594006	30.0.0.3	224.0.0.5	OSPF	122	LS Update
493	72.966804006	30.0.0.1	224.0.0.5	OSPF	134	LS Update
494	72.984954006	30.0.0.3	224.0.0.5	OSPF	154	LS Update
499	73.396819006	30.0.0.1	224.0.0.5	OSPF	82	Hello Packet
500	73.397030006	30.0.0.1	224.0.0.5	OSPF	98	LS Acknowledge
503	73.840530006	30.0.0.3	224.0.0.5	OSPF	82	Hello Packet
504	73.840846006	30.0.0.3	224.0.0.5	OSPF	118	LS Acknowledge
505	73.897119006	30.0.0.3	224.0.0.5	OSPF	122	LS Update
513	74.034759006	30.0.0.3	224.0.0.5	OSPF	194	LS Update
514	74.119536006	30.0.0.3	224.0.0.5	OSPF	94	LS Update

Figura 3.6: Tráfico OSPF capturado en la interfaz `rfvmA.3`.

No.	Time	Source	Destination	Protocol	Length	Info
464	72.394100006	50.0.0.1	224.0.0.5	OSPF	78	Hello Packet
501	73.397341006	50.0.0.1	224.0.0.5	OSPF	78	Hello Packet
506	73.865044006	50.0.0.4	224.0.0.5	OSPF	78	Hello Packet
521	74.407052006	50.0.0.1	224.0.0.5	OSPF	82	Hello Packet
524	74.584831006	02:d4:d4:d4:d4:d4	Broadcast	ARP	42	Who has 50.0.0.1? Tell 50.0.0.4
525	74.584859006	02:a4:a4:a4:a4:a4	02:d4:d4:d4:d4:d4	ARP	42	50.0.0.1 is at 02:a4:a4:a4:a4:a4
526	74.609931006	50.0.0.4	50.0.0.1	OSPF	66	DB Description
527	74.624155006	50.0.0.1	50.0.0.4	OSPF	66	DB Description
528	74.625948006	50.0.0.1	50.0.0.4	OSPF	206	DB Description
529	74.670869006	50.0.0.4	50.0.0.1	OSPF	86	DB Description
530	74.671054006	50.0.0.1	50.0.0.4	OSPF	66	DB Description
531	74.671080006	50.0.0.1	50.0.0.4	OSPF	70	LS Request
532	74.739403006	50.0.0.4	50.0.0.1	OSPF	94	LS Request
533	74.739609006	50.0.0.4	224.0.0.5	OSPF	134	LS Update
534	74.758851006	50.0.0.1	224.0.0.5	OSPF	254	LS Update
538	74.864615006	50.0.0.4	224.0.0.5	OSPF	82	Hello Packet

Figura 3.7: Tráfico OSPF capturado en la interfaz `rfvmA.4`.

Cuando el comando `pingall` se ejecuta en *mininet*, pueden verse como circulan los distintos mensajes ICMP generados por cada una de las interfaces de la máquina virtual. Como se ve en la figura 3.8, por la interfaz `rfvmA.1` se envían las respuestas a las peticiones ICMP hechas por el *host* `h1`, así como las peticiones ICMP enviadas por los demás *hosts*.

No.	Time	Source	Destination	Protocol	Length	Info
826	95.004801006	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0xb8cf4f77
828	95.518274006	172.31.1.1	224.0.0.5	OSPF	78	Hello Packet
835	96.502269006	172.31.1.1	224.0.0.5	OSPF	78	Hello Packet
842	97.504223006	172.31.1.1	224.0.0.5	OSPF	78	Hello Packet
848	98.258851006	46:ab:3b:0a:fd:84	Broadcast	ARP	42	Who has 172.31.1.1? Tell 172.31.1.100
850	98.258883006	02:a1:a1:a1:a1:a1	46:ab:3b:0a:fd:84	ARP	42	172.31.1.1 is at 02:a1:a1:a1:a1:a1
858	98.326372006	172.31.2.100	172.31.1.100	ICMP	98	Echo (ping) reply id=0x0cdd, seq=1/256, ttl=63
859	98.361840006	172.31.3.100	172.31.1.100	ICMP	98	Echo (ping) reply id=0x0cde, seq=1/256, ttl=63
860	98.406603006	172.31.4.100	172.31.1.100	ICMP	98	Echo (ping) reply id=0x0cdf, seq=1/256, ttl=63
861	98.439751006	172.31.2.100	172.31.1.100	ICMP	98	Echo (ping) request id=0x0ce0, seq=1/256, ttl=63
862	98.506633006	172.31.1.1	224.0.0.5	OSPF	78	Hello Packet
865	98.578998006	172.31.3.100	172.31.1.100	ICMP	98	Echo (ping) request id=0x0ce3, seq=1/256, ttl=63
866	98.637743006	172.31.4.100	172.31.1.100	ICMP	98	Echo (ping) request id=0x0ce6, seq=1/256, ttl=63
870	99.507704006	172.31.1.1	224.0.0.5	OSPF	78	Hello Packet

Figura 3.8: Paquetes ICMP capturados en la interfaz `rfvmA.1`.

Si se capturan los paquetes que circulan por las demás interfaces de la máquina virtual *rfvmA*, se puede observar como por cada una de ellas se reciben las peticiones ICMP enviadas por las otros *hosts*. Estos paquetes son los que posteriormente la máquina `rfvmA` reenvía a su *host* en la figura 3.8.

No.	Time	Source	Destination	Protocol	Length	Info
805	92.741910006	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0x7753a120
810	93.146221006	10.0.0.2	224.0.0.5	OSPF	82	Hello Packet
813	93.483808006	10.0.0.1	224.0.0.5	OSPF	82	Hello Packet
818	94.156232006	10.0.0.2	224.0.0.5	OSPF	82	Hello Packet
821	94.512729006	10.0.0.1	224.0.0.5	OSPF	82	Hello Packet
827	95.197161006	10.0.0.2	224.0.0.5	OSPF	82	Hello Packet
829	95.515737006	10.0.0.1	224.0.0.5	OSPF	82	Hello Packet
833	96.157395006	10.0.0.2	224.0.0.5	OSPF	82	Hello Packet
836	96.502248006	10.0.0.1	224.0.0.5	OSPF	82	Hello Packet
840	97.164258006	10.0.0.2	224.0.0.5	OSPF	82	Hello Packet
843	97.504012006	10.0.0.1	224.0.0.5	OSPF	82	Hello Packet
849	98.151276006	10.0.0.2	224.0.0.5	OSPF	82	Hello Packet
851	98.326350006	172.31.2.100	172.31.1.100	ICMP	98	Echo (ping) reply id=0x0cdd, seq=1/256, ttl=64
852	98.439720006	172.31.2.100	172.31.1.100	ICMP	98	Echo (ping) request id=0x0ce0, seq=1/256, ttl=64
853	98.506424006	10.0.0.1	224.0.0.5	OSPF	82	Hello Packet
869	99.172237006	10.0.0.2	224.0.0.5	OSPF	82	Hello Packet
871	99.507285006	10.0.0.1	224.0.0.5	OSPF	82	Hello Packet
875	99.596655006	fe80::fc16:e5ff:fe7f:2	ff02::fb	MDNS	107	Standard query 0x0000 PTR _ipps._tcp.local, "QM" quest
879	100.17762206	10.0.0.2	224.0.0.5	OSPF	82	Hello Packet

Figura 3.9: Paquetes ICMP capturados en la interfaz `rfvmA.2`.

No.	Time	Source	Destination	Protocol	Length	Info
822	94.495213006	30.0.0.1	224.0.0.5	OSPF	82	Hello Packet
825	94.855969006	30.0.0.3	224.0.0.5	OSPF	82	Hello Packet
830	95.502472006	30.0.0.1	224.0.0.5	OSPF	82	Hello Packet
832	95.948977006	30.0.0.3	224.0.0.5	OSPF	82	Hello Packet
837	96.502207006	30.0.0.1	224.0.0.5	OSPF	82	Hello Packet
839	96.909588006	30.0.0.3	224.0.0.5	OSPF	82	Hello Packet
844	97.503650006	30.0.0.1	224.0.0.5	OSPF	82	Hello Packet
847	97.925898006	30.0.0.3	224.0.0.5	OSPF	82	Hello Packet
854	98.361820006	172.31.3.100	172.31.1.100	ICMP	98	Echo (ping) reply id=0x0cde, seq=1/256, ttl=64
855	98.503970006	30.0.0.1	224.0.0.5	OSPF	82	Hello Packet
863	98.578971006	172.31.3.100	172.31.1.100	ICMP	98	Echo (ping) request id=0x0ce3, seq=1/256, ttl=64
867	98.869996006	30.0.0.3	224.0.0.5	OSPF	82	Hello Packet

Figura 3.10: Paquetes ICMP capturados en la interfaz `rfvmA.3`.

No.	Time	Source	Destination	Protocol	Length	Info
824	95.009429006	50.0.0.4	224.0.0.5	OSPF	82	Hello Packet
831	95.518665006	50.0.0.1	224.0.0.5	OSPF	82	Hello Packet
834	96.040824006	50.0.0.4	224.0.0.5	OSPF	82	Hello Packet
838	96.502311006	50.0.0.1	224.0.0.5	OSPF	82	Hello Packet
841	97.075591006	50.0.0.4	224.0.0.5	OSPF	82	Hello Packet
845	97.504481006	50.0.0.1	224.0.0.5	OSPF	82	Hello Packet
846	98.033484006	50.0.0.4	224.0.0.5	OSPF	82	Hello Packet
856	98.406583006	172.31.4.100	172.31.1.100	ICMP	98	Echo (ping) reply id=0x0cdf, seq=1/256, ttl=64
857	98.506807006	50.0.0.1	224.0.0.5	OSPF	82	Hello Packet
864	98.637714006	172.31.4.100	172.31.1.100	ICMP	98	Echo (ping) request id=0x0ce6, seq=1/256, ttl=64
868	99.077187006	50.0.0.4	224.0.0.5	OSPF	82	Hello Packet

Figura 3.11: Paquetes ICMP capturados en la interfaz `rfvmA.4`.

3.4.2. Tablas de rutas

En la sección anterior se han observado tres instantes diferentes durante el proceso de puesta en marcha de la red. Estos tres instantes son el momento en el que se han iniciado las máquinas virtuales, el instante en el que el plano de datos ha estado funcional y el momento en el que se ha ejecutado el comando `pingall` de *mininet*.

Durante estos tres instantes las tablas de rutas de los clientes RouteFlow han ido cambiando a medida que los distintos elementos de la red se han ido comunicando entre sí. Siguiendo con el apartado anterior, las observaciones llevadas a cabo en esta sección, se realizarán sobre la máquina virtual *rfvmA*. Para hacer las observaciones, se abrirá una consola en esa máquina virtual desde la que poder ejecutar los comandos y acceder a los datos necesarios. Para ello se ejecuta el siguiente comando, que iniciará una consola en la máquina virtual *rfvmA*, en la que se pedirá entrar con un nombre y contraseña. El nombre y contraseña por defecto de los contenedores Linux son ubuntu/ubuntu.

```
# lxc-console -n rfvmA
```

En esta sección se observarán tanto la tabla de rutas como la tabla ARP del cliente *rfvmA* en cada uno de los instantes explicados anteriormente. Dichas tablas pueden visualizarse desde la consola que se ha abierto con el comando anterior, mediante los dos comandos que se muestran a continuación, siendo el primero para ver la tabla del protocolo ARP y el segundo para ver la tabla de rutas:

```
$ arp -n
$ route -n
```

Si se observan tanto la tabla de rutas como la tabla ARP del cliente RouteFlow `rfvmA` en el momento en el que este se inicia, se puede observar que solo contienen información acerca de las interfaces del cliente: en el caso de la tabla del protocolo ARP, solo conoce la dirección MAC del *bridge* `lxcbr0` (que es el *bridge* que LXC tiene configurado por defecto para que las máquinas virtuales estén conectadas entre sí, y que en este proyecto se utiliza también para gestionar la comunicación con MongoDB), mientras que en la tabla de rutas solo aparecen las redes a las que tiene

una interfaz conectada.

```
Dirección               TipoHW  DirecciónHW         Indic Máscara         Interfaz
192.168.10.1            ether   fe:59:f3:dd:fb:89   C                     eth0
```

Figura 3.12: Tabla ARP al momento de iniciarse `rfvmA`.

```
Tabla de rutas IP del núcleo
Destino         Pasarela        Genmask         Indic Métric Ref    Uso Interfaz
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth2
30.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth3
50.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth4
172.31.1.0      0.0.0.0         255.255.255.0   U     0      0        0 eth1
192.168.10.0    0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

Figura 3.13: Tabla de rutas al momento de iniciarse `rfvmA`.

En el instante en el que el plano de datos pasa a estar operativo, el protocolo OSPF empieza a intercambiar información de enrutamiento entre los clientes RouteFlow. Esto hace que los clientes vayan aprendiendo nuevas rutas y sus tablas se vean modificadas. Por ejemplo, se puede ver en la figura 3.15 como el cliente `rfvmA` ha aprendido cinco rutas nuevas y cómo puede acceder a ellas. De la misma manera, en la tabla ARP de la figura 3.14, se han guardado las direcciones MAC de los *routers* que le permiten llegar a las nuevas redes que ha aprendido.

```
Dirección               TipoHW  DirecciónHW         Indic Máscara         Interfaz
192.168.10.1            ether   fe:0a:62:c3:30:99   C                     eth0
10.0.0.2                ether   02:b2:b2:b2:b2:b2   C                     eth2
30.0.0.3                ether   02:c3:c3:c3:c3:c3   C                     eth3
50.0.0.4                ether   02:d4:d4:d4:d4:d4   C                     eth4
```

Figura 3.14: Tabla ARP de `rfvmA` cuando el plano de datos esta funcional.

```
Tabla de rutas IP del núcleo
Destino         Pasarela        Genmask         Indic Métric Ref    Uso Interfaz
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth2
20.0.0.0        30.0.0.3        255.255.255.0   UG    20     0        0 eth3
30.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth3
40.0.0.0        10.0.0.2        255.255.255.0   UG    20     0        0 eth2
50.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth4
172.31.1.0      0.0.0.0         255.255.255.0   U     0      0        0 eth1
172.31.2.0      10.0.0.2        255.255.255.0   UG    20     0        0 eth2
172.31.3.0      30.0.0.3        255.255.255.0   UG    20     0        0 eth3
172.31.4.0      50.0.0.4        255.255.255.0   UG    20     0        0 eth4
192.168.10.0    0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

Figura 3.15: Tabla de rutas de `rfvmA` cuando el plano de datos esta funcional

Cuando se ejecuta `pingall` en *mininet*, la tabla de rutas no se ve afectada ya que las rutas necesarias han sido aprendidas anteriormente, pero sí que se actualiza la tabla ARP, en la que se ha añadido la dirección MAC del *host* al que está conectado tal y como puede observarse en la figura 3.16.

```
Dirección                TipoHW  DirecciónHW         Indic Máscara         Interfaz
172.31.1.100             ether   16:2f:89:96:10:fc   C                     eth1
50.0.0.4                 ether   02:d4:d4:d4:d4:d4   C                     eth4
10.0.0.2                 ether   02:b2:b2:b2:b2:b2   C                     eth2
30.0.0.3                 ether   02:c3:c3:c3:c3:c3   C                     eth3
192.168.10.1             ether   fe:47:7f:3b:e0:1c   C                     eth0
```

Figura 3.16: Tabla ARP de `rfvmA` cuando se ha ejecutado `pingall`.

3.4.3. Mensajes IPC

Hasta este punto, las observaciones realizadas han servido para comprobar que la red se comporta como lo haría cualquier otra red tradicional con la misma topología. De aquí en adelante, se analizarán las comunicaciones internas entre los componentes de RouteFlow que se llevan a cabo para administrar la red. Esta comunicación utiliza mensajes IPC generados a través de MongoDB. En este apartado se observarán los mensajes enviados por cada uno de los canales de comunicación de RouteFlow: el canal RFClient-RFServer y el canal RFProxy-RFServer. Para monitorizar los mensajes que se transmiten por cada uno de los canales, RouteFlow proporciona un script llamado `rfdb.py`.

Si se ejecuta ese script en el momento de iniciarse las máquinas virtuales, en el canal RFProxy-RFServer se puede ver como se transmiten mensajes RouteFlow del tipo *DatapathPortRegister*, encargados de informar al RFServer de los puertos que tiene el *datapath* RFVS. Estos mensajes contienen el ID del controlador del *datapath*, el ID del propio *datapath* y el puerto del que se está informando. Debido a que cada mensaje solo puede informar de un puerto, se enviarán tantos mensajes como puertos tenga el RFVS. En la figura F.5 del apéndice F se puede ver un extracto de la salida del script `rfdb.py`, en el que se muestran algunos de estos mensajes.

Tal y como se puede observar en el extracto anterior, el ID del *datapath* empieza por el número ”72667673”, eso lo identifica como un RFVS. Una vez el RFVS ha notificado todos sus puertos, se empiezan a enviar mensajes RouteFlow del tipo *VirtualPlaneMap*, los cuales son utilizados para mapear los puertos de las máquinas virtuales con los del RFVS. Estos mensajes contienen el ID de la máquina virtual, el ID del RFVS y el número de un par de puertos a mapear. De la misma manera que para los mensajes anteriores, se deberá mandar un mensaje por cada par de puertos a mapear. En la figura F.4 del apéndice F, puede observarse un extracto de la salida del script dónde aparecen este tipo de mensajes.

Si para el mismo instante se observa el canal RFClient-RFServer, se puede ver como los mensajes que RouteFlow transmite son de tipo *PortRegister*, encargados de informar de las direcciones MAC que tienen los puertos de las máquinas virtuales. Estos mensajes contienen el ID de la máquina virtual, junto con el puerto y la dirección MAC de una de sus interfaces. Estos mensajes se envían para cada una de las interfaces. En la figura F.5 del apéndice F, puede verse el contenido de este tipo mensajes.

Una vez el plano de datos esta funcional, por el canal RFProxy-RFServer se mandan mensajes *DatapathPortRegister* para informar de todos los puertos que tienen los *datapath* del plano de datos. Posteriormente, se vuelven a mandar los mensajes *VirtualPlaneMap* para indicar el mapeo entre los puertos de las máquinas virtuales y los puertos del RFVS, y seguidamente se mandan mensajes RouteFlow de tipo *DataPlaneMap*, encargados de mapear los puertos de los *datapath* del plano de datos con los puertos del RFVS. Tras enviarse estos mensajes, se empiezan a enviar mensajes del tipo *RouteMod* que contienen información sobre las rutas que deben aprender los *datapath* y las prioridades que deben tener esas rutas en las tablas de flujo. En el extracto de la salida del script `rfdb.py` de la figura F.6 del apéndice F, pueden verse algunos de estos mensajes.

Por otra parte, a través del canal RFClient-RFServer se mandan mensajes del tipo *PortConfig* con los que se configuran las interfaces del cliente, pudiéndolas activar o desactivar. Estos mensajes se mandan para cada una de las interfaces de cada máquina virtual y contienen el ID de la máquina virtual, el número del puerto de la interfaz y el ID de la operación que se debe llevar a cabo. A estos mensajes les siguen los mensajes *RouteMod*, encargados de instalar las rutas necesarias en cada una de las máquinas. Esta secuencia de mensajes puede observarse en la figura F.7 del apéndice F.

3.4.4. Volcado de flujos

En esta sección se analizarán los flujos OpenFlow que se generan en el plano de datos de la red. Tomando de referencia los apartados anteriores, el punto de observación de la red seguirá siendo el cliente RouteFlow `rfvmA`. En este caso, al tratarse del plano de datos (en el que no hay *routers*), se hablará del switch asociado a la máquina virtual de ese cliente.

Para los análisis llevados a cabo en esta sección, desde el entorno de trabajo se ha abierto un terminal en la máquina virtual "mininet" utilizando el protocolo seguro *ssh*. Dado que el usuario de esa máquina se llama `mininet` y que su dirección IP es la 10.0.1.102, el comando utilizado ha sido el siguiente:

```
$ ssh mininet@10.0.1.102
```

Una vez ejecutado el comando, pedirá la contraseña de ese usuario, la cual no se ha modificado y es la que viene por defecto en la imagen virtual (`mininet` al igual que el nombre de usuario). Este análisis debe realizarse de forma remota, ya que la máquina virtual "mininet" no tiene interfaz gráfica y al estarse ejecutándose el programa *mininet*, no permite utilizar la línea de comandos de la máquina. Es por ello, que se ha abierto un terminal desde el entorno de trabajo, para poder así ejecutar y observar los datos necesarios.

Cuando el terminal en la máquina virtual "mininet" se haya abierto, se procederá a observar los flujos OpenFlow que estén instalados en la tabla de flujo del *switch* `s5` (correspondiente al cliente RouteFlow). Para que se muestren los flujos instalados en el *switch* se ejecutará el siguiente comando de Open vSwitch:

```
# ovs-ofctl dump-flows s5
```

Una vez ejecutado el comando anterior, aparecerán todos las reglas de flujo que tenga instaladas el *switch* en ese momento. A continuación se muestran algunas de las reglas que aparecen al ejecutar ese comando:

```
cookie=0x12007ef103, duration=187.94s, table=0, n_packets=0, n_bytes=0,
 priority=32800,tcp,dl_dst=02:a4:a4:a4:a4:a4,nw_dst=50.0.0.1,tp_src=179
 actions=CONTROLLER:65509
cookie=0x12007ef103, duration=191.06s, table=0, n_packets=0, n_bytes=0,
 priority=32800,tcp,dl_dst=02:a3:a3:a3:a3:a3,nw_dst=30.0.0.1,tp_src=179
 actions=CONTROLLER:65509
cookie=0x12007ef103, duration=195.653s, table=0, n_packets=0, n_bytes=0,
 priority=32800,tcp,dl_dst=02:a1:a1:a1:a1:a1,nw_dst=172.31.1.1,tp_src=179
 actions=CONTROLLER:65509
cookie=0x12007ef103, duration=194.18s, table=0, n_packets=0, n_bytes=0,
 priority=32800,tcp,dl_dst=02:a2:a2:a2:a2:a2,nw_dst=10.0.0.1,tp_src=179
 actions=CONTROLLER:65509
cookie=0x12007ef103, duration=182.598s, table=0, n_packets=0, n_bytes=0,
 priority=16640,ip,in_port=1,dl_dst=02:a1:a1:a1:a1:a1,nw_dst=40.0.0.0/24
 actions=set_field:02:a2:a2:a2:a2:a2->eth_src,set_field:02:b2:b2:b2:b2:b2->eth_dst,output:2
cookie=0x12007ef103, duration=184.007s, table=0, n_packets=0, n_bytes=0,
 priority=16640,ip,in_port=1,dl_dst=02:a1:a1:a1:a1:a1,nw_dst=172.31.4.0/24
 actions=set_field:02:a4:a4:a4:a4:a4->eth_src,set_field:02:d4:d4:d4:d4:d4->eth_dst,output:4
cookie=0x12007ef103, duration=184.918s, table=0, n_packets=0, n_bytes=0,
 priority=16640,ip,in_port=1,dl_dst=02:a1:a1:a1:a1:a1,nw_dst=172.31.2.0/24
 actions=set_field:02:a2:a2:a2:a2:a2->eth_src,set_field:02:b2:b2:b2:b2:b2->eth_dst,output:2
cookie=0x12007ef103, duration=181.085s, table=0, n_packets=0, n_bytes=0,
 priority=16640,ip,in_port=2,dl_dst=02:a2:a2:a2:a2:a2,nw_dst=172.31.3.0/24
 actions=set_field:02:a3:a3:a3:a3:a3->eth_src,set_field:02:c3:c3:c3:c3:c3->eth_dst,output:3
cookie=0x12007ef103, duration=181.098s, table=0, n_packets=0, n_bytes=0,
 priority=16640,ip,in_port=1,dl_dst=02:a1:a1:a1:a1:a1,nw_dst=172.31.3.0/24
 actions=set_field:02:a3:a3:a3:a3:a3->eth_src,set_field:02:c3:c3:c3:c3:c3->eth_dst,output:3
cookie=0x12007ef103, duration=181.383s, table=0, n_packets=0, n_bytes=0,
```

```
priority=16640,ip,in_port=1,dl_dst=02:a1:a1:a1:a1:a1,nw_dst=20.0.0.0/24
actions=set_field:02:a3:a3:a3:a3:a3->eth_src,set_field:02:c3:c3:c3:c3:c3->eth_dst,output:3
```

En el extracto anterior se han marcado en rojo dos de las reglas de flujo. En la primera de ellas se pueden observar varios campos de datos. Entre ellos destacan los siguientes: el tiempo que hace que está instalada esa regla en la tabla de flujo del *switch* (en este caso 195,653 segundos), el ID de la tabla de flujo en la que está instalada esa regla (en este caso la tabla 0), la prioridad de la regla (32800), el tipo de tráfico que espera (TCP), la dirección MAC destino (02:a1:a1:a1:a1:a1), la dirección IP destino (172.31.1.1), el puerto TCP origen (puerto 179) y las acciones que se llevaran a cabo (enviar los paquetes al controlador). En este caso, la regla define que todos los paquetes con destino el router del cliente `rfvmA` (con dirección IP 172.31.1.1), serán enviados al controlador.

La segunda regla de flujo marcada en rojo es ligeramente diferente a la anterior. En este caso se trata de una regla de flujo que indica por dónde se debe enviar un paquete IP que entre por el puerto 1 del *switch*, que tenga como destino la red IP 172.31.4.0/24. Concretamente, en la regla se especifica que un paquete IP que haya entrado por el puerto 1 del *switch* y tenga como dirección destino MAC 02:a1:a1:a1:a1:a1 (es decir que ha entrado en el *switch* por la interfaz `rfvmA.1`), y como dirección destino IP 172.31.4.0/24, será modificado para que su dirección MAC origen sea la 02:a4:a4:a4:a4:a4 (es decir que se saque por la interfaz `rfvmA.4`), su dirección MAC destino sea 02:d4:d4:d4:d4:d4 (la interfaz `rfvmD.4` del cliente `rfvmD`) y que el paquete se envíe por el puerto 4 del *switch* (correspondiente a la interfaz `rfvmA.4`).

3.4.5. RFWeb

Junto a RouteFlow se incorpora una aplicación llamada RFWeb, la cual consiste en una interfaz web que permite ver los mensajes RouteFlow que pasan por cada uno de los dos canales que tiene RouteFlow. También muestra una tabla en la que se muestra el mapeo entre los *datapaths* y las máquinas virtuales de la red, así como también permite ver una representación gráfica de la red virtual, en la que se muestran los distintos elementos de la arquitectura de RouteFlow.

Para poder utilizar esta interfaz web, tan solo hay que ejecutar el script `rfweb_server.py` que se encuentra en el directorio `rfweb` de RouteFlow, y posteriormente en un navegador web acceder a la dirección "localhost:8080/index.html".

```
$ python rfweb_server.py
```

En la figura 3.17 puede verse la interfaz web de RouteFlow mostrando los mensajes que se transmiten a través de los canales de RouteFlow para la red virtual de este proyecto:

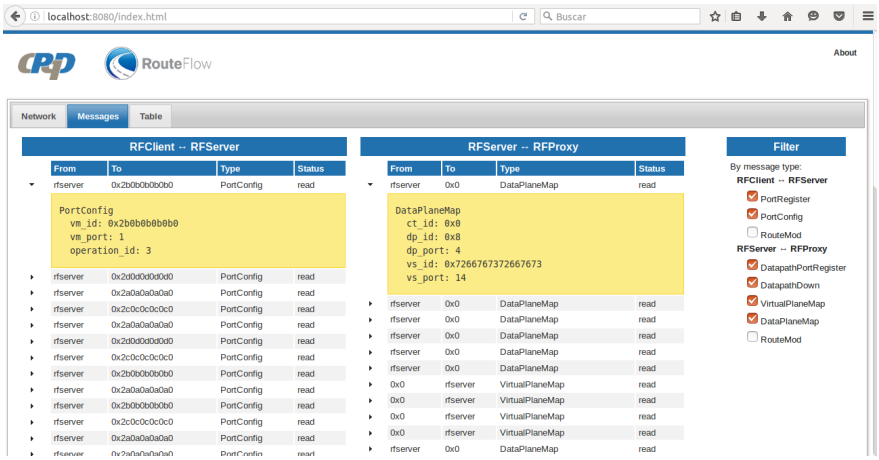


Figura 3.17: Interfaz web de RouteFlow.

Capítulo 4

Resultados

4.1. Presupuesto del proyecto

Para el desarrollo de este proyecto se ha utilizado en la mayoría de los casos software libre y gratuito, por lo que no ha reportado ningún gasto. En el único caso en el que se ha utilizado software privativo ha sido para la realización de los reportes que se han ido entregando a lo largo del proyecto. En ese caso, las herramientas ofimáticas utilizadas propiedad de Microsoft, han supuesto un coste de 80 euros.

También ha sido necesario un equipo informático para la creación de los entornos de trabajo y su posterior estudio. En este caso, se ha utilizado un portátil ASUS A53S con un coste de 699 euros, el cual tendrá un tiempo de amortización de 5 años. El sistema operativo utilizado ha sido un Ubuntu Linux 14.04, el cual tampoco ha supuesto coste alguno debido a su condición de software libre y gratuito.

Finalmente, se debe de tener en cuenta el tiempo de búsqueda e investigación que ha sido necesario para poder realizar el proyecto. Dado que una sola persona ha sido la responsable del proyecto y su sueldo ha sido elegido en base a los convenios laborales actuales, se estima que dado que la persona ha cobrado 8 euros/hora y ésta ha dedicado 32 horas semanales a realizar el proyecto durante 20 semanas, el coste total que ha supuesto es de 5120 euros.

De lo descrito anteriormente, tal y como se muestra en la tabla 4.1, el coste total que ha reportado este proyecto de investigación ha sido de 5899 euros.

Concepto	Coste
Microsoft Word	80€
Portátil ASUS A53S	699€
Sueldo Ingeniero	5120€
Total	**5899€**

Tabla 4.1: Coste realización del proyecto.

4.2. Conclusiones y futuro desarrollo

En este proyecto se explorado el paradigma del *Software Defined Networking* y en particular una implementación de transición suave entre las redes tradicionales y este nuevo paradigma denominada RouteFlow. Para ello, en primer lugar se detalla en qué consisten las redes basadas en software y su principal protocolo denominado OpenFlow. A continuación, el proyecto se ha centrado en explicar RouteFlow, tanto a nivel teórico, como a nivel práctico, incluyendo

su instalación y configuración y mostrando su funcionamiento en un caso práctico.

Durante la realización de este proyecto ha habido complicaciones no previstas al inicio de este. Una de ellas, ha sido la búsqueda de información acerca de la aplicación RouteFlow, que ha comportado una complejidad que no se esperaba. Dicha complejidad viene del hecho de que la información existente referente a RouteFlow no está lo suficientemente centralizada, por lo que ha sido necesario buscar a fondo para encontrar fuentes de información que aportaran detalles y puntos de vista diferentes. También cabe añadir que dada la falta de un control de versiones de la aplicación, se ha requerido de un estudio meticuloso de dichas fuentes de información, ya que cada una de ellas hace referencia a características de RouteFlow que pueden haber sufrido grandes modificaciones en las últimas versiones, dejando a dichas fuentes de información desactualizadas.

También ha quedado patente que RouteFlow es un proyecto aún en desarrollo que necesita mejorar algunos aspectos y que debe seguir evolucionando para adaptarse a las nuevas características que van ofreciendo las SDN. Desde sus inicios, RouteFlow ha ido evolucionando y se ha bifurcado en varios proyectos experimentales que añaden distintas características para mantener al día el proyecto. Una de estas ramas ha sido la utilizada para realizar este estudio, ya que el proyecto original de RouteFlow se ha quedado estancado en una versión muy temprana y utiliza tecnologías SDN muy primitivas (p.ej. OpenFlow 1.0) que actualmente han evolucionado mucho, ofreciendo mejoras y nuevas características.

Añadir también que sus creadores siguen trabajando en el proyecto y han publicado algunos artículos explicando cómo funciona RouteFlow, aun así todos sus artículos tienen el mismo punto de vista por lo que una publicación no aportan mucha información nueva respecto a la otra. Sí que es verdad que esta aplicación empieza a ser más conocida en el ámbito científico y han habido más personas que han realizado estudios sobre el rendimiento de RouteFlow, ofreciendo también nuevos puntos de vista desde los que explicar su funcionamiento. Aun así, no existía un punto de vista global que además aportara detalles a nivel de instalación y configuración. De ahí surge el objetivo de este proyecto: dar a conocer RouteFlow, explicar cómo se estructura y funciona, pero además explicar cómo se debe configurar y cómo funciona a nivel de usuario.

Otra complejidad que ha supuesto la realización de este proyecto, tiene que ver con lo expuesto anteriormente, y es que al ser un proyecto aún en desarrollo, mantenido por una comunidad Open Source no muy extensa y activa, hace que la aplicación no se actualice con la misma frecuencia en que lo hacen las herramientas que utiliza y ello afecta a la estabilidad de la aplicación, haciendo que no funcione correctamente. Esto ha hecho que para poder utilizar la aplicación en este proyecto se requirieran modificaciones en el código fuente.

Una vez resueltos los retos que conlleva el proyecto, se ha conseguido ofrecer una explicación en detalle de qué es RouteFlow y cómo funciona. Este documento reúne gran parte de la información que puede encontrarse en Internet acerca de RouteFlow, ofreciendo el mayor detalle que ha sido posible, sin llegar a niveles que pudieran despistar la atención del lector sobre el tema que se pretende dar a conocer en este documento.

Tras la finalización de este proyecto, se ha visto el potencial que puede tener RouteFlow en el ámbito de las SDN. Un sector en el que podría tomar un papel muy importante es en el de los operadores de red, ya que actualmente disponen de amplias redes desplegadas y una actualización completa de ellas para funcionar como SDN sería demasiado costosa. Es por ello, que RouteFlow ofrece la posibilidad de actualizar la red progresivamente sin repercutir en el rendimiento de la red, ya que permite que tanto los dispositivos tradicionales como los que utilizan OpenFlow puedan operar en la misma red. Además ofrece la posibilidad de virtualizar el plano de control, pudiendo ser ejecutado dentro de una sola máquina virtual, facilitando así la configuración de red y la resolución de problemas que puedan surgir. Esta centralización permite agregar varios *datapaths* en una misma entidad lógica de nivel 3 (p.ej un *router*), lo que ofrece una abstracción con la que están familiarizados los operadores de red. Finalmente, comentar que esperamos que el presente proyecto sea de utilidad para introducir a los lectores en estos nuevos paradigmas y tecnologías.

Bibliografía

[1] C. Lorier, *Cardigan: Development of a distributer router*. PhD thesis, The University of Waikato, 2013.

[2] K. L. I. S. S. Shenker and J. Rexford, "Routing as a service,"

[3] P. Zeng, K. Nguyen, Y. Shen, and S. Yamada, "On the resilience of software defined routing platform," in *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*, pp. 1–4, IEEE, 2014.

[4] J. Stringer and C. Owen, "Routemod: A flexible approach to route propagation," 2013.

[5] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk, "Revisiting routing control platforms with the eyes and muscles of software-defined networking," in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 13–18, ACM, 2012.

[6] P. Membrey, E. Plugge, and D. Hawkins, *The definitive guide to MongoDB: the noSQL database for cloud and desktop computing*. Apress, 2011.

[7] J. L. Muñoz, *Virtualization and SDN*. 2016.

[8] K. Kuroki, N. Matsumoto, and M. Hayashi, "Scalable openflow controller redundancy tackling local and global recoveries," in *The Fifth International Conference on Advances in Future Internet*, pp. 61–66, Citeseer, 2013.

[9] A. Vidal, "Routeflow wiki — github." `https://github.com/routeflow/RouteFlow/wiki`, 2013. [Internet; descargado 6-febrero-2016].

[10] F. Hu, *Network Innovation through OpenFlow and SDN: Principles and Design*. CRC Press, 2014.

[11] S. N. Rizvi, D. Raumer, F. Wohlfart, and G. Carle, "Towards carrier grade sdns," *Computer Networks*, vol. 92, pp. 218–226, 2015.

[12] C. Rothenberg, R. Chua, J. Bailey, M. Winter, C. Correa, S. de Lucena, M. Salvador, and T. Nadeau, "When open source meets network control planes," *Computer*, vol. 47, pp. 46–54, Nov 2014.

[13] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," white paper, Open Networking Foundation, Palo Alto, CA, USA, Apr. 2012.

[14] Open Networking Foundation, "OpenFlow Switch Specification," white paper, Open Networking Foundation, Palo Alto, CA, USA, June 2012. Version 1.3.0 (Wire Protocol 0x04).

Apéndice A

Software Defined Networking (SDN)

A.1. Introducción

El Software Defined Networking (del inglés, *Redes Definidas por Software*) es una arquitectura de red emergente, promovida por la Open Networking Foundation. Su aparición ha supuesto la apertura de nuevas formas de comunicación con dispositivos de red, que han sido históricamente propietario[1], permitiendo menor coste de operaciones y mayor agilidad y organización de las redes, así como también, ofreciendo nuevas oportunidades de innovación [12].

Tradicionalmente, los equipos de red han sido diseñados con el principio de "integración vertical" [7]. Este principio consiste en que las distintas capas o planos de abstracción con las que un dispositivo es diseñado, están integrados en el mismo dispositivo. Concretamente, el hardware de reenvío y la inteligencia que controla el envío de datos, están contenidos dentro del mismo dispositivo de red. De ahí parte la idea principal de las Software Defined Networks (SDN), que consiste en desacoplar el control de la red, de las funciones de envío de datos. En otras palabras, separar la inteligencia que controla las funciones de envío, de los dispositivos de red. De esta manera, en las SDN, el control de los equipos de reenvío (a partir de ahora llamados *datapaths*) es gestionado por una entidad llamada controlador, que será quien disponga de la "inteligencia" que gestione el comportamiento de los *datapaths*.

Los *datapath* usan una API[2] libre (conocida como Southbound API) con la que se comunican con el controlador. De este modo, cuando un paquete llega a uno de los puertos del *datapath*, solicita al controlador qué debe hacer. El controlador es el que decide el comportamiento de reenvío, como por ejemplo, enviar un paquete a través de los puertos. Para evitar cuellos de botella entre los *datapaths* y el controlador, lo que el controlador hace es instalar en las tablas de reenvío de los *datapaths*, lo que se llaman flujos. Estos flujos manejan el tráfico de datos, utilizando campos y metadatos[3] de los paquetes, tales como el puerto de entrada, las direcciones MAC e IP, los protocolos de transporte y demás, para definir unas reglas y aplicar unas acciones sobre los paquetes cuyos campos coincidan con los especificados en dichas reglas. Dicho de otra forma, los flujos son un conjunto de instrucciones que se aplican sobre los paquetes que cumplan unas reglas predefinidas.

El desacople del hardware y el software de red, permite la centralización de la parte de control (llamado plano de control), mientras se mantiene la función de envío de paquetes (llamado plano de datos) distribuida en los conmutadores de red físicos. Esta centralización, dota al controlador de una visión global de la red que permite abstraer la infraestructura subyacente para que las aplicaciones, los servicios de red y los motores de políticas puedan tratar a la red como una sola entidad lógica o virtual [13]. Las SDN además, simplifican los propios dispositivos de red, ya que debido a este desacople, ya no necesitan entender y procesar miles de normas de protocolo, sino que simplemente deben aceptar las instrucciones que reciben del controlador.

[1] dispositivos en que los fabricantes no permiten acceder a la documentación y por lo tanto se desconoce cómo escribir drivers para su hardware.

[2] conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos. Representa un método para conseguir abstracción en la programación, entre los niveles o capas inferiores y los superiores del software.

[3] información no relevante para el usuario final pero sí de suma importancia para el sistema que maneja los datos.

La centralización y abstracción que ofrecen las Software Defined Networks, permiten a los operadores y administradores de las redes, hacerse con el control de toda una red desde un único punto lógico, independientemente del fabricante de los dispositivos. Este hecho, simplifica enormemente el diseño y funcionamiento de la red, puesto que en vez de tener que configurar manualmente miles de dispositivos, tan sólo se debe configurar esta abstracción simplificada de la red. Esta abstracción, junto con el uso de APIs libres entre las capas de control y de aplicación de SDN, permiten aprovechar los servicios y capacidades de la red, sin estar ligado a los detalles de su implementación. Además, aprovechando la inteligencia centralizada del controlador, se puede modificar el comportamiento de la red en tiempo real y desplegar nuevas aplicaciones y servicios de red, en un tiempo mucho menor del que se suele requerir actualmente. La centralización del estado de la red en la capa de control, ofrece a los administradores la flexibilidad para configurar, administrar, proteger y optimizar recursos de la red, de forma dinámica, a través de programas SDN automatizados.

A.2. Arquitectura de una Software Defined Network (SDN)

La arquitectura de una Software Defined Network, se puede dividir en tres planos: el plano de datos, el plano de control y el plano de aplicación. A continuación se detallan los elementos que forman parte de cada plano y que se ven representados en la figura A.1:

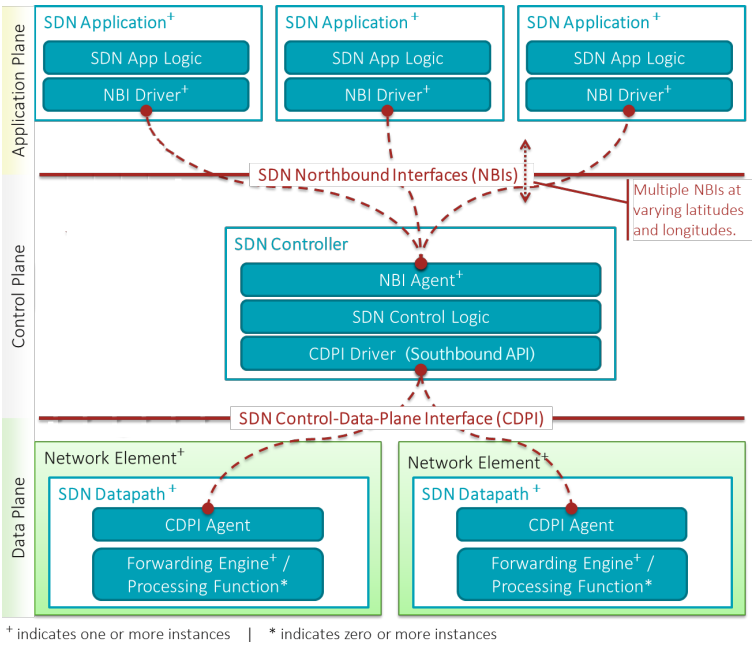


Figura A.1: Arquitectura de una Software Defined Network.

- Elementos que conectan los distintos planos:
  - **Interfaz del plano de control al plano de datos(CDPI):** La CDPI es la interfaz definida entre un controlador SDN y un *datapath* SDN, que por lo menos, proporciona:
    - Control de todas las operaciones de reenvío.
    - Anuncio de capacidades.
    - Informes de estadísticas.
    - Notificación de eventos.

Esta interfaz está implementada de manera libre e independientemente del fabricante, mediante el uso de APIs llamadas Southbound, tales como el protocolo OpenFlow, que permite la comunicación entre el controlador y los *datapaths*, especificando cómo deben manejar la información que comparten entre ellos.

- **Interfaces Northbound (NBI):** Las NBI son las interfaces que hay entre las aplicaciones SDN y los controladores SDN, que por lo general, proporcionan vistas abstractas de la red y permiten la expresión directa de la conducta y requisitos de la red. Esto puede ocurrir en cualquier nivel de abstracción (latitud) y a través de diferentes conjuntos de funcionalidad (longitud). Estas interfaces también se implementan de manera libre, mediante unas APIs llamadas Northbound, e independientemente del fabricante.

- Elementos del plano de datos:

  - ***Datapaths* SDN:** Los *datapaths* SDN son dispositivos de red lógicos, que expone visibilidad y control sobre sus capacidades de reenvío y procesamiento. Están formados por un agente CDPI, un conjunto de uno o más motores de reenvío de tráfico y cero o más funciones de procesamiento de tráfico. Estos motores y funciones, pueden incluir reenvíos simples entre las interfaces externas del *datapath* o entre las funciones de procesamiento de tráfico. Un elemento (físico) de la red puede contener uno o más *datapaths* SDN, aunque también se puede definir un solo *datapath* SDN a través de varios elementos físicos de la red. Esta definición no implica que no se permitan implementaciones tales como la lógica de mapeo físico, la gestión de recursos compartidos, la virtualización del *datapath* SDN, la interoperabilidad con redes que no sean SDN, así como la implementación de la funcionalidad de procesamiento de datos.

- Elementos del plano de control:

  - **Controlador SDN:** El controlador SDN es una entidad lógica centralizada, encargada de ofrecer una visión abstracta de la red a las aplicaciones SDN (así como estadísticas y eventos) y de traducir los requisitos de la aplicación SDN, para que los *datapaths* de niveles inferiores puedan interpretarlos. Un controlador SDN se compone de uno o más agentes NBI, la lógica de control SDN y el driver para la CDPI (Southbound API). La definición del controlador no impide la implementación de varios controladores, la conexión jerárquica de controladores, la comunicación entre las interfaces de los controladores, así como tampoco la virtualización de recursos de red.

- Elementos del plano de aplicación:

  - **Aplicación SDN:** Las aplicaciones SDN son programas que comunican sus requisitos y el comportamiento de la red deseado al controlador SDN, a través de una interfaz Northbound (NBI). Además, disponen de una visión abstracta de la red para la toma de decisiones. Una aplicación SDN consta de una aplicación lógica SDN y uno o más drivers NBI (Northbound API). Las aplicaciones SDN, pueden suponer ellas mismas otra capa de control abstracto de la red, ofreciendo NBIs de niveles superiores a través de los respectivos agentes NBI.

Apéndice B

OpenFlow

B.1. Introducción

OpenFlow es la primera interfaz de comunicaciones estándar definida entre las capas de control y de datos de una arquitectura SDN. OpenFlow permite el acceso directo y la manipulación del plano de datos de los dispositivos de red tales como los *switches*[1] o los *routers*[2], tanto físicos como virtuales. Normalmente el software OpenFlow se puede dividir en dos capas o niveles: por un lado un *Network Operating System* (Sistema operativo de red) encargado de la implementación OpenFlow y de la conexión con los *datapaths*, y por otra parte, las aplicaciones encargadas de controlar los eventos y definir el comportamiento de la red.

El protocolo OpenFlow se implementa en ambos lados de la interfaz que conecta los dispositivos que forman la infraestructura de la red, con el software de control SDN. OpenFlow utiliza el concepto de flujos para identificar el tráfico de red, basándose en reglas de coincidencia[3] predefinidas que pueden ser programadas estáticamente o dinámicamente por el controlador. Además, permite definir cómo debe fluir el tráfico a través de los dispositivos de red, basándose en parámetros tales como los patrones de uso, las aplicaciones y los recursos en la nube. Gracias a que OpenFlow permite programar la red en función de cada flujo de datos, las redes con arquitecturas SDN basadas en OpenFlow pueden responder en tiempo real a cambios sucedidos a nivel de aplicación, usuario o de sesión. Actualmente, el enrutamiento basado en IP de la redes tradicionales, no proporciona este nivel de control, ya que todos los flujos entre dos extremos deben seguir el mismo camino a través de la red, independientemente de sus necesidades.

El protocolo OpenFlow es un factor clave para las redes definidas por software y actualmente es el único protocolo SDN estandarizado que permite la manipulación directa del plano de datos de los dispositivos de red. Si bien inicialmente se aplicó sólo para redes basadas en Ethernet, la conmutación de paquetes de OpenFlow se puede extender a un conjunto de casos de uso mucho más amplio. Las SDN basadas en OpenFlow se pueden implementar en las redes existentes, tanto físicas como virtuales. Los dispositivos de red pueden soportar el reenvío basado en OpenFlow, así como el reenvío tradicional, lo que hace que sea muy fácil para empresas y operadoras introducir progresivamente tecnologías SDN basadas en OpenFlow, incluso en entornos de red de múltiples fabricantes.

Para la estandarización de OpenFlow se creó la Open Networking Foundation[4], la cual se hace responsable del protocolo, la configuración, las pruebas de interoperabilidad, y de otras actividades, ayudando a asegurar la interoperabilidad entre los dispositivos de red y el software de control de distintos proveedores. OpenFlow está siendo ampliamente adoptado por los proveedores de infraestructuras, que por lo general, lo implementan a través de una simple actuali-

[1] dispositivo digital lógico de interconexión de equipos que opera en la capa de enlace de datos del modelo OSI. Su función es interconectar dos o más segmentos de red. También pueden llamarse conmutadores.

[2] dispositivo que proporciona conectividad a nivel de red o nivel 3 en el modelo OSI. Su función principal consiste en enviar o encaminar paquetes de datos de una red a otra.

[3] criterios utilizados para comparar valores de distintos campos, en busca de coincidencias.

[4] `https://www.opennetworking.org/`

zación de firmware o software. Esto es debido a que la arquitectura SDN basada en OpenFlow puede integrarse sin problemas con una infraestructura ya existente de una empresa u operadora, además de que proporciona una migración sencilla para aquellos segmentos de la red que más necesiten la funcionalidad de la SDN.

B.2. Especificaciones del OpenFlow versión 1.3.0

En este apartado se profundiza en el protocolo OpenFlow: primero se especificarán los componentes de un *switch* OpenFlow y seguidamente se explicará en detalle el protocolo. Concretamente su versión 1.3.0, dado que es la que se va a utilizar en este proyecto. Esta versión contiene muchas características nuevas con respecto a la versión inicial 1.0.0. Entre otras, cabe destacar la posibilidad de utilizar MPLS, VLANs, IPv6, múltiples trayectos (*multipath*), puertos lógicos, múltiples tablas de flujos, tunelización (*tunneling*) y otras mejoras en *Quality-of-Service* (*QoS*).

B.2.1. Componentes de un *switch* OpenFlow

Tal y cómo se muestra en la figura B.1, un *switch* OpenFlow consta por una parte, de una o más tablas de flujos y una tabla de grupos, las cuales realizan operaciones de búsqueda y reenvío de paquetes, y por otra, de un canal OpenFlow que conecta el *switch* a un controlador externo [14]. Mediante este canal y el protocolo OpenFlow, el *switch* se comunica con el controlador que se encarga de gestionarlo. Con el protocolo OpenFlow, el controlador puede añadir, actualizar y borrar entradas de flujo de las tablas de flujos, tanto de forma reactiva (en respuesta a los paquetes), como de forma proactiva. Cada una de estas tablas de flujos del *switch* contiene un conjunto de entradas de flujo; a su vez, cada entrada de flujo consta de campos de coincidencia, contadores y un conjunto de instrucciones[5] para aplicar a los paquetes coincidentes.

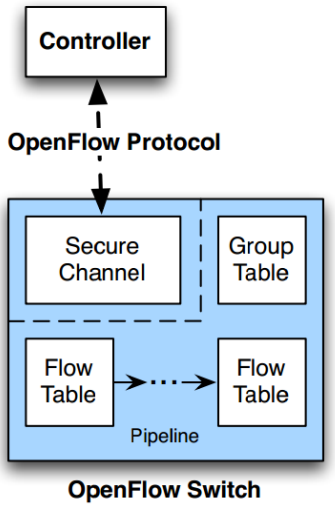


Figura B.1: Componentes principales de un *switch* OpenFlow.

El proceso de búsqueda de coincidencias consiste en comparar los campos de los paquetes con los campos de coincidencia de las entradas de flujo, y empieza en la primera tabla de flujos, con la posibilidad de seguir con las demás tablas de flujos. Las entradas de flujo coinciden con los paquetes en orden de prioridad, es decir, se utiliza la primera entrada coincidente de cada tabla. Si se encuentra una coincidencia con una entrada, se ejecutan las instrucciones asociadas con

[5] describen como OpenFlow procesa los paquetes que coinciden con una entrada de flujos. Pueden dirigir los paquetes a otra tabla de flujo, añadir acciones al Conjunto de Acciones a realizar o aplicar directamente una Lista de Acciones sobre los paquetes.

esa entrada de flujo en concreto. Por el contrario, si no se encuentran coincidencias en una tabla de flujos, se llevarán a cabo las acciones[6] configuradas en la entrada de flujo *table-miss*. Un ejemplo de las acciones que se pueden llevar a cabo en esta entrada serían: enviar el paquete que no ha coincidido con ninguna entrada de flujo al controlador, a través del canal OpenFlow, descartar el paquete o continuar la búsqueda de coincidencias en la siguiente tabla de flujos.

Las instrucciones asociadas con cada entrada de flujo pueden tanto contener acciones, como modificar el procesamiento de la *pipeline*[7]. Las acciones incluidas en las instrucciones, describen el reenvío y modificación de los paquetes, y el procesamiento de la tabla de grupos. Las instrucciones de procesamiento de la *pipeline* permiten a los paquetes ser enviados a las tablas siguientes para un procesamiento más amplio, además de permitir que la información se comunique entre tablas en forma de metadatos. El procesamiento de la *pipeline* se detiene cuando el conjunto de instrucciones asociadas con una entrada de flujo que haya hecho coincidencia, no especifica una tabla siguiente; en este punto, el paquete normalmente es modificado y reenviado.

Las entradas de flujo pueden reenviar paquetes a un puerto que suele ser físico, aunque también podrían ser un puerto lógico definido por un *switch* o un puerto reservado definido por el protocolo OpenFlow. Los puertos reservados pueden especificar acciones de reenvío genéricas, tales como el envío del paquete al controlador, la saturación (*flooding*) y el reenvío mediante métodos no-OpenFlow (como por ejemplo el procesamiento "normal“ del *switch*), mientras que los puertos lógicos definidos por un *switch* pueden especificar grupos de agregación de enlaces, túneles o interfaces de *loopback*[8].

Las acciones asociadas con las entradas de flujo también pueden dirigir paquetes a un grupo que especifique un procesamiento adicional. Los grupos representan un conjunto de acciones de reenvío complejas, tales como la agregación de links o el multicamino (*multipath*), entre otras. También permite que varias entradas de flujo dirijan los paquetes hacia un único identificador (p.ej. el reenvío del tráfico IP del siguiente salto[9] a un punto común). Esta abstracción permite modificar de manera eficiente las acciones de salida de las entradas de flujo que sean comunes.

La tabla de grupos contiene entradas de grupo; cada entrada de grupo contiene una lista de *buckets*[10] de acciones con semánticas específicas dependiendo del tipo de grupo. Las acciones de uno o más *buckets* de acciones son aplicadas a los paquetes enviados al grupo.

B.2.2. Puertos OpenFlow

Los puertos OpenFlow son las interfaces de red usadas para el paso de paquetes entre el punto de procesamiento OpenFlow y el resto de la red. Los *switches* OpenFlow se conectan entre sí mediante estos puertos.

Un *switch* OpenFlow dispone de un número de puertos OpenFlow para el procesamiento OpenFlow. Este conjunto de puertos no tiene porqué ser igual al número de puertos de que disponga el dispositivo hardware, ya que algunas interfaces puede que se deshabiliten para OpenFlow, o que por otra parte, se definan puertos OpenFlow adicionales.

Los paquetes OpenFlow se reciben en un puerto de entrada y son procesados por la *pipeline* de OpenFlow, que los reenviará a un puerto de salida. El puerto de entrada también representa una de las propiedades que tendrán los paquetes que vayan a través de la *pipeline* de OpenFlow, que indica por qué puerto el *switch* OpenFlow recibió el paquete. Esta propiedad se puede utilizar para encontrar coincidencias. La *pipeline* de OpenFlow puede decidir enviar un paquete a un puerto de salida, utilizando una acción que defina la forma en que el paquete salga de nuevo a la red.

[6] operaciones que envían un paquete dado a un puerto o lo modifican, por ejemplo decrementando su campo TTL.
[7] conjunto de tablas de flujos vinculadas que proporcionan coincidencias, reenvío y modificación de paquetes en un *switch* OpenFlow.
[8] la dirección de *loopback* es una dirección especial que los *hosts* utilizan para dirigir el tráfico hacia ellos mismos.
[9] con ”salto” nos referimos al envío de un paquete de un dispositivo a otro.
[10] tipo de buffer de datos o documento, en que los datos están divididos en regiones.

Un *switch* OpenFlow debe soportar tres tipos de puertos OpenFlow: los puertos físicos, los puertos lógicos y los puertos reservados.

Puertos físicos

Los puertos físicos OpenFlow son puertos definidos por el *switch* que corresponden con una interfaz hardware del *switch*. Por ejemplo, en un *switch* Ethernet, cada puerto físico se correspondería con una interfaz Ethernet.

En algunas implementaciones, el *switch* OpenFlow puede ser virtualizado sobre el hardware del *switch*. En estos casos, un puerto físico OpenFlow puede representar una porción virtual de la correspondiente interfaz hardware del *switch*.

Puertos lógicos

Los puertos lógicos OpenFlow son puertos definidos por el *switch* que no corresponden directamente a una interfaz hardware del *switch*. Los puertos lógicos son abstracciones de alto nivel que pueden ser definidas en el *switch* usando métodos no-OpenFlow (p.ej. grupos de agregación de links, túneles, interfaces de *loopback*).

Los puertos lógicos pueden incluir la encapsulación de paquetes y se pueden asignar a varios puertos físicos. El procesamiento hecho por el puerto lógico debe ser transparente para el procesamiento de OpenFlow, ya que OpenFlow los procesará como si fueran puertos físicos OpenFlow.

Las únicas diferencias entre los puertos físicos y los lógicos son que un paquete asociado con un puerto lógico, puede tener un campo extra de metadatos llamado *Tunnel-ID*, y que cuando un paquete recibido en un puerto lógico es enviado al controlador, tanto su puerto lógico, como el puerto físico asociado, son reportados al controlador.

Puertos Reservados

Los puertos reservados OpenFlow son puertos lógicos definidos por las especificaciones del *switch* OpenFlow. Estos puertos especifican acciones de reenvío genéricas tales como el envío al controlador, la saturación (flooding) o el reenvío utilizando métodos no-OpenFlow, como por ejemplo el procesamiento “normal” de paquetes en el *switch*.

A continuación se detallan los puertos reservados que debe soportar un *switch*:

- **ALL:** Representa a todos los puertos del *switch* que puedan ser utilizados para el reenvío de un paquete específico. Sólo se puede utilizar como puerto de salida. Si se utiliza éste puerto, una copia del paquete se envía a todos los puertos, excluyendo al puerto de entrada del paquete y a los puertos que tengan deshabilitada la opción de reenvío.
- **CONTROLLER:** Representa el canal de control que une el *switch* con el controlador OpenFlow. Puede ser utilizado como puerto de entrada o como puerto de salida. Cuando se utiliza como puerto de salida, el paquete se encapsula en un mensaje de paquetes mediante el protocolo OpenFlow. Por otro lado, cuando se utiliza como puerto de entrada, se identifica el paquete originado en el controlador.
- **TABLE:** Representa el inicio de la *pipeline* de OpenFlow. Éste puerto sólo es válido usarlo en una acción de salida de una Lista de Acciones de un mensaje *packet-out*, y sirve para enviar el paquete a la primera tabla de flujos para que pueda ser procesado por la *pipeline* de OpenFlow.
- **IN_PORT:** Representa el puerto de entrada del paquete. Sólo puede usarse como puerto de salida, es decir, para enviar un paquete de vuelta por el mismo puerto que ha entrado.
- **ANY:** Valor especial utilizado en algunos comandos OpenFlow cuando no se especifica ningún puerto (puerto comodín). No puede ser usado ni como puerto de entrada, ni de salida.

- **LOCAL *(opcional)*:** Representa a las pilas de red local y de gestión del *switch*. Puede ser usado tanto como puerto de entrada, como de salida. El puerto local permite a entidades remotas interactuar con el *switch* y sus servicios de red a través de la red OpenFlow, en lugar de a través de una red de control independiente. Con un conjunto adecuado de entradas de flujo predeterminadas, el puerto local puede ser usado para implementar una conexión controladora in-band.
- **NORMAL *(opcional)*:** Representa la *pipeline* no-OpenFlow tradicional del *switch*. Puede ser utilizada sólo como puerto de salida, y procesa el paquete usando la *pipeline* normal. Si el *switch* no puede enviar los paquetes desde la *pipeline* de OpenFlow hacia la *pipeline* normal, debe de estar indicado en las especificaciones del *switch* que esta acción no está soportada.
- **FLOOD *(opcional)*:** Representa la saturación (*flooding*) utilizando la *pipeline* normal del *switch*. Puede ser usada sólo como puerto de salida. En general, se envía el paquete a todos los puertos, menos al puerto de entrada y a los puertos que estén en estado bloqueado. El *switch* también puede usar el VLAN ID del paquete para seleccionar qué puertos saturar.

Existen dos tipos de *switches* compatibles con OpenFlow: los que soportan únicamente OpenFlow, y los híbridos (pueden soportar la conmutación "normal" y OpenFlow). Los *switches* que únicamente utilizan OpenFlow, no soportan ni los puertos **NORMAL**, ni los **FLOOD**, mientras que los *switches* híbridos soportan los dos tipos. El reenvío de paquetes al puerto **FLOOD** depende de la implementación y configuración del *switch*, mientras que el reenvío mediante un grupo de tipo *all* permite al controlador implementar más flexiblemente la saturación.

B.2.3. Tablas OpenFlow

En esta sección se detallan los componentes que forman las tablas de flujos y las tablas de grupos, además de explicar los mecanismos de búsqueda de coincidencias y el manejo de acciones.

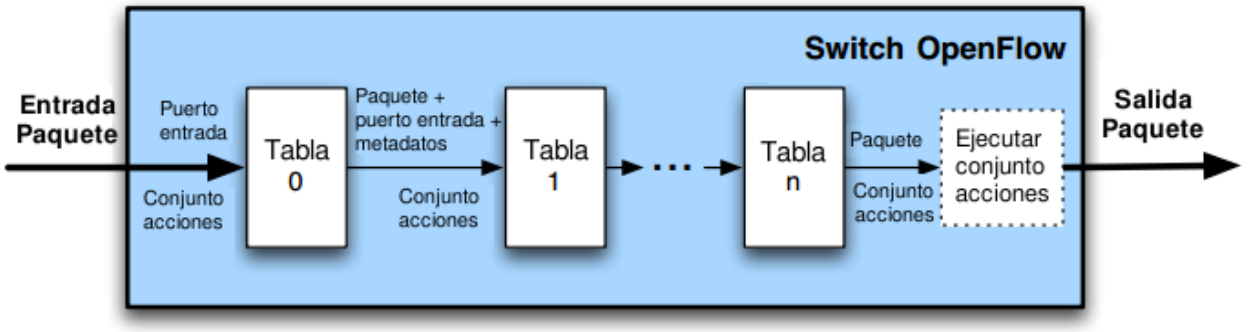
Procesamiento de la *pipeline*

Cómo se ha comentado en el apartado anterior, existen dos tipos de *switches*: los que soportan únicamente OpenFlow, y los híbridos. Los *switches* que sólo soportan OpenFlow, únicamente pueden realizar operaciones OpenFlow y los paquetes sólo pueden ser procesados por la *pipeline* de OpenFlow.

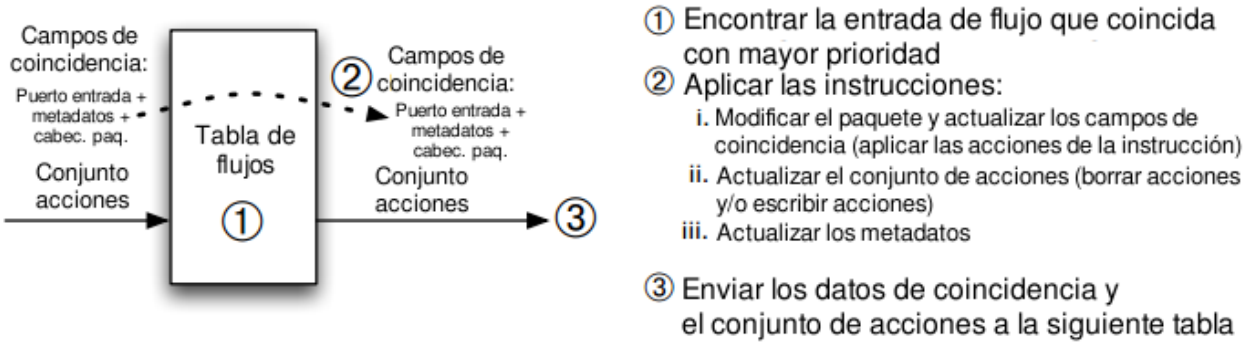
Los *switches* híbridos soportan tanto operaciones OpenFlow como operaciones normales de conmutación Ethernet (p.ej. conmutación tradicional Ethernet de nivel 2, aislamiento VLAN, enrutamiento de nivel 3, ACL y procesamiento de *QoS*). Éste tipo de *switches* deben poder enrutar el tráfico de datos, tanto a la *pipeline* de OpenFlow como a la *pipeline* normal. Por ejemplo, un *switch* puede usar el tag VLAN o el puerto de entrada para decidir si procesar el paquete con la *pipeline* normal o con la *pipeline* OpenFlow. Los *switches* híbridos también deben permitir pasar un paquete desde la *pipeline* de OpenFlow hacia la *pipeline* normal, mediante los puertos reservados **NORMAL** y **FLOOD**.

La *pipeline* OpenFlow de cada *switch* OpenFlow contiene varias tablas de flujos, cada una de ellas con múltiples entradas de flujo. El procesamiento de la *pipeline* OpenFlow define cómo los paquetes interactúan con las tablas de flujos. Un *switch* OpenFlow debe tener al menos una tabla de flujos, y opcionalmente puede tener más. En el caso de que el *switch* tuviera una sola tabla de flujos, el procesamiento de la *pipeline* se simplificaría en gran medida.

Las tablas de flujos de un *switch* OpenFlow están numeradas secuencialmente, empezando por el 0. El procesamiento de la *pipeline* empieza siempre en la primera tabla de flujos: el proceso de búsqueda de coincidencias de un paquete empieza con las entradas de flujo de la tabla 0. Las demás tablas se utilizarán en función del resultado obtenido en la primera tabla.



(a) Los paquetes se comparan en múltiples tablas de la *pipeline*



(b) Procesamiento de una tabla

Figura B.2: Flujo de paquetes en el procesamiento de la *pipeline*.

Cuando un paquete es procesado por una tabla de flujos, se buscan entradas de flujo que coincidan con los campos del paquete. Si se encuentra una entrada que coincida, se ejecutan las instrucciones que están incluidas en ella. Estas instrucciones pueden indicar que se envíe el paquete a otra tabla de flujo, donde el mismo proceso se repetirá otra vez. Una entrada de flujo sólo puede enviar paquetes a tablas de flujos con número superior a la que pertenece la entrada, en otras palabras, el procesamiento de la *pipeline* sólo se puede hacer hacia adelante no hacia atrás, por lo que por ejemplo, la última tabla no podrá dirigir un paquete hacia otra tabla. Si la entrada de flujo no direcciona un paquete hacia otra tabla, el procesamiento de la *pipeline* se detiene en esa tabla de flujos. Cuando el procesamiento de la *pipeline* se detiene, el paquete es procesado por su Conjunto de Acciones y normalmente reenviado.

Si un paquete no coincide con ninguna entrada de flujo de una tabla, se trata de una omisión de la tabla. El comportamiento en la omisión de una tabla depende de cómo esté configurada ésa tabla. Una entrada de flujo *table-miss* (omisión de tabla) en la tabla de flujos, puede especificar cómo procesar los paquetes sin coincidencias: desde descartar los paquetes o pasarlos a otra tabla, hasta enviarlos al controlador a través del canal de control mediante mensajes de paquetes.

Las entradas de flujo de las tablas, pueden ser eliminadas de dos formas: por una petición del controlador, o por el mecanismo de caducidad del *switch*. El mecanismo de caducidad de flujos del *switch* se basa en el estado y la configuración de las entradas de flujo, independientemente del controlador. Para ello, cada entrada de flujo tiene asociados dos campos: el *idle_timeout* y el *hard_timeout*. Cuando se le asigna un valor numérico (representando segundos) al campo *idle_timeout*, el *switch* eliminará la entrada de flujo asociada, si al transcurrir el tiempo indicado, ningún paquete ha coincidido con la entrada. Al campo *hard_timeout* también se le asigna un valor numérico que representa el tiempo máximo (expresado en segundos) que una entrada de flujo permanecerá en el *switch*, independientemente de las coincidencias que haya tenido con los paquetes. El *switch* debe eliminar la entrada de flujo cuando alguno de estos

dos campos expire.

Por otra parte el controlador también puede eliminar las entradas de flujo de una tabla enviando mensajes de *modificación de tabla* (`OFPFC_DELETE` o `OFPFC_DELETE_STRICT`). Cada vez que una entrada de flujo es eliminada, el *switch* debe comunicárselo al controlador mediante un mensaje de *flujo eliminado*, en el que se detalla la información del flujo y la razón por la que se ha eliminado (flujo expirado o eliminado por el controlador).

Componentes de una entrada de flujo

Las tablas de flujos están compuestas por entradas de flujo, y cada una de estas entradas contiene los siguientes componentes:

- **Campos coincidentes:** Valores que se comparan con los campos de un paquete para encontrar coincidencias. Estos valores son: el puerto de entrada, las cabeceras de los paquetes, y opcionalmente metadatos especificados por una tabla anterior.
- **Prioridad:** Valor numérico que determina qué entradas se analizan antes que las demás en busca de coincidencias.
- **Contadores:** Valores que se utilizan para contabilizar (bytes, paquetes, segundos...) y para actualizar los paquetes con coincidencias.
- **Instrucciones:** Modifican el Conjunto de Acciones o el procesamiento de la *pipeline*.
- **Timeouts:** Indica la cantidad de tiempo o tiempo de inactividad máximo que un flujo se mantendrá en un *switch* antes de que caduque.
- **Cookie:** Valor de datos opaco elegido por el controlador. Puede ser usado por el controlador para filtrar estadísticas de flujos, modificar y eliminar flujos, pero no para el procesamiento de paquetes.

Una entrada de flujo de la tabla de flujos se identifica por sus campos de coincidencia y de prioridad, ya que estos dos campos sólo pueden identificar una única entrada de flujo. La entrada de flujo que omita todos los campos y tenga prioridad 0, será la entrada de flujo *table-miss*.

Proceso de búsqueda de coincidencias

Tras recibir un paquete, un *switch* OpenFlow lleva a cabo las funciones que se detallan en la figura B.3. El *switch* empieza con una búsqueda de coincidencias en la primera tabla de flujos, y siguiendo con el procesamiento de la *pipeline*, puede realizar búsquedas en otras tablas de flujos.

Los campos de coincidencia se extraen de los paquetes. En función del tipo de paquete, se usarán unos campos u otros para consultar las tablas de flujos, los cuales suelen incluir varios campos de las cabeceras de los paquetes, como por ejemplo, la dirección de origen Ethernet o la dirección destino IPv4. Además de las cabeceras, se pueden utilizar el puerto de entrada del paquete y los campos de metadatos. Los metadatos pueden utilizarse para pasar información de una tabla a otra del *switch*. Los campos de coincidencia de los paquetes representan el estado actual de los mismos: si se han aplicado cambios en las cabeceras del paquete en tablas anteriores, quedará reflejado en los campos de coincidencia del paquete.

Un paquete coincide con una tabla de flujos si los valores de los campos coincidentes del paquete utilizados en la búsqueda, coinciden con una entrada de flujo de la tabla. Si una entrada de flujo de la tabla tiene un valor **ANY** (omitir campos), coincidirá con todos los posibles valores de la cabecera del paquete. Si el *switch* soporta máscaras de bits arbitrarios en campos de coincidencia específicos, se podrán especificar con mayor precisión las coincidencias.

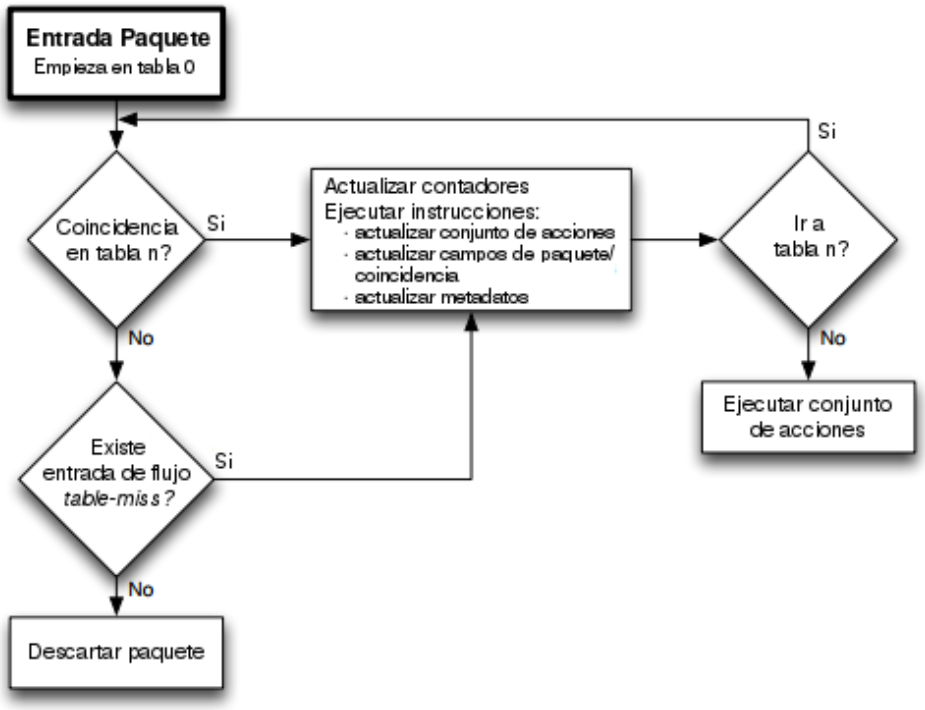


Figura B.3: Diagrama de flujo que detalla el flujo de paquetes a través de un *switch* OpenFlow.

El paquete se compara con la tabla de flujos, y sólo la entrada de flujo con mayor prioridad que coincida con el paquete será la seleccionada. Los contadores asociados con la entrada de flujo seleccionada deben ser actualizados y el conjunto de instrucciones incluidas en la entrada deben ser aplicadas. Si hay varias entradas de flujo que coincidan con la misma prioridad, la entrada de flujo seleccionada es explícitamente indefinida. Éste caso sólo puede surgir cuando un escritor del controlador nunca activa el bit de `OFPFF_CHECK_OVERLAP` en los mensajes de modificación de flujo y se añaden entradas que se superponen.

Los fragmentos IP deben volverse a juntar antes del procesamiento de la *pipeline*, si la configuración del *switch* contiene el *flag* `OPC_FRAG_REASM`.Esta versión de OpenFlow, no define el comportamiento esperado cuando un *switch* recibe un paquete con formato incorrecto o que esté dañado.

B.2.4. Entrada de flujo *table-miss*

Cada tabla de flujos debe soportar una entrada de flujo *table-miss* para procesar omisiones de la tabla. Esta entrada de flujo especifica cómo procesar a los paquetes que no hayan hecho ninguna coincidencia con las demás entradas de flujo de una tabla de flujos. Por ejemplo, pueden enviar estos paquetes al controlador, descartarlos o dirigirlos a la siguiente tabla.

Las entradas de flujo *table-miss* se identifican por omitir todos los campos de coincidencia y tener la prioridad más baja (0). De darse el caso de que una tabla no soporte la omisión de campos, si se deberá permitir hacerlo a la entrada de flujo *table-miss*.

Al igual que las demás entradas de flujo, una entrada de flujo *table-miss* no existe por defecto en una tabla de flujos: el controlador será el encargado de añadirlas o eliminarlas en cualquier momento. Si la entrada de flujo *table-miss* no existe, los paquetes que no hayan coincidido con ninguna de las demás entradas de flujo serán descartados. Este

comportamiento puede ser modificado en la configuración del *switch*.

B.2.5. Instrucciones

Como ya se ha explicado en secciones anteriores, cada entrada de flujo contiene un conjunto de instrucciones que se ejecutan cuando un paquete coincide con la entrada de flujo. Estas instrucciones dan como resultado cambios en el paquete, en el conjunto de acciones y/o en el procesamiento de la *pipeline*. El *switch* debe soportar varios tipos de instrucciones:

- *(Opcional)* **Medida *id-medida*:** Dirige el paquete a la medida especificada. Como resultado de la medición, el paquete puede ser descartado.
- *(Opcional)* **Aplicar-Acciones *acciones***: Aplica inmediatamente la o las acciones especificadas sin aplicar cambios en el Conjunto de Acciones. Esta instrucción puede ser utilizada para modificar un paquete entre dos tablas o para ejecutar múltiples acciones del mismo tipo. Las acciones se especifican como una Lista de Acciones.
- *(Opcional)* **Limpiar-Acciones *acciones*:** Borra inmediatamente todas las acciones del Conjunto de Acciones.
- **Escribir-Acciones *acciones*:** Combina la o las acciones especificadas con el Conjunto de Acciones actual. Si la acción ya existe en el Conjunto de Acciones, se sobrescribe, y de lo contrario se añade.
- *(Opcional)* **Escribir-Metadatos *metadatos/máscara*:** Escribe el valor enmascarado de metadatos en el campo de metadatos. La máscara especifica qué bits del registro de metadatos deberían ser modificados.
- **Ir-a-Tabla *id-tabla*:** Indica la siguiente tabla en el procesamiento de la *pipeline*. El *id* de la tabla que se especifica debe ser mayor que el de la tabla actual. Esta instrucción no puede ser incluida en las entradas de flujo de la última tabla de la *pipeline*.

El conjunto de instrucciones asociado a una entrada de flujo puede contener un máximo de una instrucción de cada tipo; las instrucciones se ejecutarán en el mismo orden en que aparecen en la lista anterior. Las únicas restricciones en el orden de ejecución son: que las instrucciones *Medida* se ejecuten antes que las instrucciones *Aplicar-Acciones*, que las *Limpiar-Acciones* se ejecuten antes que las *Escribir-Acciones*, y que las últimas en ejecutarse sean las instrucciones *Ir-a-Tabla*.

B.2.6. Acciones

A continuación se detallan varios tipos de acciones que un *switch* debe soportar, aunque el controlador también puede consultar si el *switch* soporta otro tipo de acciones "opcionales":

- **Salida:** La acción de *salida* (*output*) envía un paquete al puerto OpenFlow especificado. Para su correcto funcionamiento, los *switches* OpenFlow deben soportar el reenvío a los puertos físicos, a los puertos lógicos definidos por el *switch* y a los puertos reservados que se requieran.
- **Configurar-Cola *(Opcional)*:** La acción *Configurar-Cola* asigna el *id* de una cola a un paquete. Cuando un paquete es reenviado a un puerto mediante la acción de *salida*, el *id* de la cola determina qué cola de ése puerto se utilizará para planificar y reenviar el paquete. El comportamiento de reenvío viene dictado por la configuración de la cola y se usa para proveer de un soporte básico de *Quality-of-Service* (*QoS*).
- **Descartar:** No hay acciones explícitas para representar un descarte. En su lugar, se descartan los paquetes que en sus Conjuntos de Acciones no tengan acciones de *salida*. Este caso puede deberse a que el conjunto de instrucciones esté vacío, que los *buckets* de acciones estén vacíos al procesar la *pipeline*, o a que previamente se haya ejecutado una instrucción *Limpiar-Acciones*.
- **Grupo:** Se procesa el paquete a través del grupo especificado. La interpretación exacta depende del tipo de grupo.

- **Push-Tag/Pop-Tag *(Opcional)*:** Los *switches* pueden soportar la habilidad para añadir/extraer (*push*/*pop*) etiquetas (*tags*). Cuando hay varias acciones *push* añadidas en el Conjunto de Acciones del paquete, se aplican en el orden definido por las reglas del Conjunto de Acciones: primero MPLS (*Multiprotocol Label Switching*), luego PBB (*Provider Backbone Bridging*) y después VLAN (*Virtual Local Area Network*).
- **Configurar-Campo *(Opcional)*:** Las distintas acciones *Configurar-Campo* modifican los valores de los respectivos campos de cabecera del paquete, y se identifican por su tipo de campo. Aunque no es estrictamente necesario, la reescritura de varios campos de cabecera utilizando acciones *Configurar-Campo*, aumenta en gran medida la utilidad de una implementación OpenFlow. Este tipo de acciones deben aplicarse siempre a la cabecera más externa posible (p.ej. una acción "Define el ID VLAN" siempre establece el ID de la etiqueta VLAN más externa), a menos que el tipo de campo especifique lo contrario.
- **Cambiar-TTL *(Opcional)*:** Las distintas acciones *Cambiar-TTL* modifican los valores de los TTL[11] IPv4, los Hop Limit IPv6 (*Límite de saltos IPv6*) o los TTL MPLS del paquete. Este tipo de acciones, al igual que las del tipo anterior, deben aplicarse siempre en la cabecera más externa posible.

Conjunto de Acciones

Cada paquete tiene asociado un Conjunto de Acciones, que por defecto está vacío. Una entrada de flujo puede modificar el Conjunto de Acciones utilizando una instrucción *Escribir-Acciones* o una instrucción *Limpiar-Acciones* que esté asociada a una coincidencia concreta. El Conjunto de Acciones se conserva a través de las tablas de flujos. Cuando el conjunto de instrucciones de una entrada de flujo no contiene una instrucción *Ir-a-Tabla*, el procesamiento de la *pipeline* se detiene, y se ejecutan las acciones del Conjunto de Acciones.

Un Conjunto de Acciones contiene como máximo una acción de cada tipo. Las acciones *Configurar-Campo* son identificadas por sus tipos de campo, por lo que el Conjunto de Acciones contiene un máximo de una acción *Configurar-Campo* por cada tipo de campo. Cuando se requiere la ejecución de varias acciones del mismo tipo (p.ej. añadir o extraer múltiples etiquetas MPLS), se debe de utilizar la instrucción *Aplicar-Acciones* para aplicarlas al momento.

Las acciones de un Conjunto de Acciones se aplican en el orden que se especifica a continuación, sin tener en cuenta en qué orden se añadieron al conjunto. Si un Conjunto de Acciones contiene una acción de tipo *grupo*, las acciones contenidas en el *bucket* de acciones del grupo también serán aplicadas en el orden que se detalla a continuación. El *switch* debe soportar el orden arbitrario de ejecución de acciones a través de la Lista de Acciones de la instrucción *Aplicar-Acciones*.

– **copiar TTL hacia dentro:** aplica acciones de *copia del TTL hacia dentro* en el paquete.

– **pop:** aplica todas las acciones de extracción (*pop*) de etiquetas en el paquete.

– **push-MPLS:** aplica todas las acciones *push* de etiquetas MPLS en el paquete.

– **push-PBB:** aplica todas las acciones *push* de etiquetas PBB en el paquete.

– **push-VLAN:** aplica todas las acciones *push* de etiquetas VLAN en el paquete.

– **copiar TTL hacia fuera:** aplica acciones de *copia del TTL hacia fuera* en el paquete.

– **decrementar TTL:** aplica la acción de *decrementar el TTL* en el paquete.

– **configurar:** aplica todas las acciones *Configurar-Campo* en el paquete.

– **qos:** aplica todas las acciones de *QoS*, tales como *Configurar-Cola* en el paquete.

[11] el TTL (Time-To-Live) es un concepto usado en redes de computadores para indicar por cuántos nodos puede pasar un paquete antes de ser descartado por la red o devuelto a su origen.

– **group:** si una acción de tipo *grupo* es especificada, aplica las acciones del *bucket* de acciones del grupo, en el orden especificado en esta lista.

– **salida:** si no se especifica ninguna acción de tipo *grupo*, envía el paquete al puerto especificado en la acción de *salida*.

La última acción en ejecutarse de un Conjunto de Acciones es la acción *salida*. Si en el conjunto se especifican tanto una acción de *grupo* como una acción de *salida*, ésta última pasa a ser ignorada y la acción de *grupo* toma precedencia. Si no se especifica ninguna acción de *salida*, ni tampoco una de *grupo*, el paquete es descartado.

Lista de Acciones

La instrucción *Aplicar-Acciones* y el mensaje *Packet-out* incluyen una Lista de Acciones. Las acciones especificadas en una Lista de Acciones son ejecutadas en el orden especificado en la lista y son aplicadas inmediatamente sobre el paquete.

La ejecución de una Lista de Acciones empieza con la primera acción de la lista y se ejecutan en orden sobre el paquete. El efecto de estas acciones es acumulativo, es decir, si una Lista de Acciones contiene dos acciones *push VLAN*, sobre el paquete se añadirán dos cabeceras VLAN. Si la lista contiene una acción de *salida*, una copia del paquete será enviada en su estado actual al puerto deseado. Si la lista contiene acciones de *grupo*, una copia del paquete en su estado actual será procesado por los *buckets* del grupo. Tras la ejecución de una Lista de Acciones en una instrucción de tipo *Aplicar-Acciones*, se continúa con la ejecución de la *pipeline* utilizando el paquete modificado. El Conjunto de Acciones no se ve modificado por la ejecución de la Lista de Acciones.

B.2.7. Tablas de grupos y de medidas

Tabla de grupos

Una tabla de grupos consiste en un conjunto de entradas de grupo. Tal y como se explicaba al principio de esta sección, los grupos representan un conjunto de acciones de reenvío complejas. La habilidad de una entrada de flujo de apuntar a un grupo, permite a OpenFlow representar métodos adicionales de reenvío. Cada entrada de grupo es identificada por el identificador de su grupo, y contiene:

– **identificador de grupo:** un entero sin signo de 32 bits que identifica de forma exclusiva a un grupo en el *switch* OpenFlow.

– **tipo de grupo:** se utiliza para determinar la semántica del grupo.

– **contadores:** se actualizan cuando un paquete es procesado por un grupo.

– ***bucket* de acciones:** una lista ordenada de *buckets* de acciones, donde cada *bucket* de acciones contiene un conjunto de acciones a ejecutar y sus parámetros asociados. Las acciones de un *bucket* son siempre aplicadas como un Conjunto de Acciones.

Un *bucket* normalmente contiene acciones que modifican el paquete y una acción de *salida* que lo envía a un puerto. Además, también puede incluir una acción de tipo *grupo* que invoque a otro grupo, siempre y cuando el *switch* soporte el encadenamiento de grupos; en ese caso el procesamiento del paquete continuaría en el grupo invocado.

Los tipos de grupos que debe soportar un *switch* son:

- **All:** Ejecuta todos los *buckets* en el grupo. Este grupo se utiliza para el reenvío *multicast*[12] y *broadcast*[13]. El paquete es clonado para cada *bucket*, de esta manera, cada *bucket* del grupo procesará un paquete. Si un *bucket*

[12] consiste en el envío de información a un grupo seleccionado de destinos simultáneamente.
[13] forma de transmisión de información donde un nodo emisor envía información a todos los nodos receptores de su red de manera simultánea.

dirige un paquete explícitamente hacia el puerto de entrada, el paquete clonado es descartado. Si el escritor del controlador quiere enviar un paquete por el puerto de entrada, el grupo deberá incluir un *bucket* extra que incluya una acción de *salida* al puerto reservado `OFPP_IN_PORT`.

- **Exclusivo *(opcional)*:** Ejecuta un *bucket* del grupo. Los paquetes son procesados por un solo *bucket* del grupo, basándose en un algoritmo de selección (p.ej. comprobación aleatoria de tuplas configuradas por el usuario o *Simple Round Robin*[14]), que reparta la carga por igual. Toda la configuración del algoritmo de selección es externo a OpenFlow. En el caso de que un puerto especificado en un *bucket* se venga abajo, el *switch* debe restringir el conjunto de *buckets*, dejando sólo los que contengan envíos a puertos activos, para así evitar el descarte constante de paquetes con destino a un puerto fuera de servicio, y de esta manera reducir el impacto que pueda tener en la red la caída de un enlace o un *switch*.
- **Indirecto:** Ejecuta el *bucket* definido en este grupo. Este grupo soporta solo un único *bucket*. Permite a múltiples entradas de flujo o grupos apuntar a un identificador de grupo común, aportando una convergencia más rápida y eficiente. Este tipo de grupo es idéntico al de tipo *all*, pero con un solo *bucket*.
- **Rápida recuperación *(opcional)*:** Ejecuta el primer *bucket* activo. Cada *bucket* de acciones está asociado a un puerto y/o grupo específicos que controlan la disponibilidad del *bucket*. Los *buckets* son evaluados en el orden definido por el grupo, y se selecciona el primer *bucket* que esté asociado a un puerto/grupo activo. Este tipo de grupo permite al *switch* modificar el reenvío, sin un viaje de ida y vuelta al controlador. Si no hay *buckets* activos, los paquetes son descartados.

Tabla de medidas

Una tabla de medidas consiste en entradas de medidas, dónde las medidas son definidas por flujo. Las medidas por flujo permiten a OpenFlow implementar varias operaciones simples de *QoS*, como por ejemplo limitar la velocidad, y pueden combinarse con las colas de cada puerto para implementar estructuras complejas de *QoS*, tales como el *Diff-Serv*[15].

Una medida, como su nombre indica, mide la tasa de paquetes asignados a la misma, y permite controlar la velocidad de los paquetes. Las medidas están asociadas a las entradas de flujo (en contraposición a las colas, que están asociadas a los puertos). Cualquier entrada de flujo puede especificar una medida en su conjunto de instrucciones. La medida mide y controla la tasa de paquetes del conjunto de todas las entradas de flujo que tiene asociadas. En la misma tabla se pueden usar múltiples medidas, pero estas deben ser disjuntas entre sí; también se pueden utilizar múltiples medidas en el mismo conjunto de paquetes, usándolas en tablas de flujo sucesivas.

Cada entrada de medida contiene:

- **Identificador de medida:** un entero sin signo de 32 bits único que identifica la medida.
- **Bandas de medidas:** una lista desordenada de bandas de medidas, donde cada banda de medidas especifica la tasa de paquetes y la manera de procesarlos.
- **Contadores:** actualizados cuando los paquetes son procesados por una medida.

B.2.8. Protocolo OpenFlow

El protocolo OpenFlow soporta tres tipos de mensajes, *controlado-a-switch*, *asíncronos* y *simétricos*, cada uno con múltiples subtipos. El intercambio de mensajes *controlador-a-switch* es iniciado por el controlador, y son utilizados para manejar o inspeccionar directamente el estado del *switch*. El intercambio de mensajes *asíncronos* es iniciado por

[14] método para seleccionar todos los elementos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento.

[15] los Servicios Diferenciados (DiffServ) proporcionan un método que intenta garantizar la calidad de servicio en redes de gran tamaño, como puede ser Internet.

el *switch* y utilizado para actualizar la información del controlador sobre eventos de red y modificaciones en el estado del *switch*. El intercambio de mensajes *simétricos* puede ser iniciado tanto por el *switch* como por el controlador, y enviado sin ser solicitado. A continuación se explica con más detalle cada uno de estos tipos.

Mensaje *controlador-a-switch*

Este tipo de mensajes son enviados por el controlador y pueden (o no) requerir una respuesta por parte del *switch*. Los tipos de mensajes que pueden haber son:

- **Características:** El controlador puede solicitar las capacidades de un *switch* mediante el envío de una solicitud de características; el *switch* debe responder con una respuesta que especifique las capacidades del *switch*. Esto normalmente se realiza tras el establecimiento del canal OpenFlow.
- **Configuración:** El controlador es capaz de configurar y consultar los parámetros de configuración del *switch*. El *switch* sólo responde a una consulta hecha desde controlador.
- **Modificar-Estado:** Los mensajes *Modificar-Estado* son enviados por el controlador para administrar el estado de los *switches*. Su propósito principal es añadir, borrar y modificar entradas de flujo/grupo en las tablas OpenFlow y configurar las propiedades de los puertos de los *switches*.
- **Leer-Estado:** Los mensajes *Leer-Estado* son utilizados por el controlador para recoger información del *switch*, como por ejemplo la configuración actual, estadísticas o las capacidades del *switch*.
- **Packet-out:** Este tipo de mensajes son utilizados por el controlador para sacar paquetes por un puerto especificado del *switch* y para enviar los paquetes recibidos a través de mensajes *Packet-in*. Los mensajes *Packet-out* deben contener un paquete completo o el ID de un *buffer* que haga referencia a un paquete almacenado en el *switch*. El mensaje también debe contener una Lista de Acciones a aplicar; si la Lista de Acciones está vacía, el paquete se descarta.
- **Barrera:** Los mensajes de petición/respuesta *Barrera* son utilizados por el controlador para asegurar que las dependencias de los mensajes se han cumplido o para recibir notificaciones de operaciones completadas.
- **Petición-Rol:** Los mensajes *Petición-Rol* son utilizados por el controlador para configurar o consultar el rol de sus canales OpenFlow. Este mensaje es de gran utilidad cuando el *switch* se conecta a múltiples controladores.
- **Configuración-Asíncrona:** Este tipo de mensajes son utilizados por el controlador para establecer un filtro adicional a los mensajes *asíncronos* que desea recibir en su canal OpenFlow. Esto es sobre todo útil cuando el *switch* se conecta a varios controladores, y normalmente se realiza tras el establecimiento del canal OpenFlow.

Mensajes *Asíncronos*

Los mensajes *asíncronos* son enviados desde el *switch* sin que el controlador haya hecho una petición. Los *switches* envían mensajes *asíncronos* al controlador para indicar la llegada de un paquete, un cambio de estado del *switch*, o un error. A continuación se explican los cuatro tipos principales de mensajes *asíncronos*:

- **Packet-in:** Transfiere el control de un paquete al controlador. Para todos los paquetes que sean reenviados al puerto reservado **CONTROLLER** mediante una entrada de flujo o una entrada de flujo *table-miss*, un evento *packet-in* será siempre enviado al controlador. Otros tipos de procesamiento, tales como la comprobación del TTL, también pueden generar eventos *packet-in* para enviar paquetes al controlador.

  Los eventos *packet-in* pueden ser configurados para almacenar paquetes en el *buffer*. Para los *packet-in* generados por una acción de *salida* de una entrada de flujo o del *bucket* de un grupo, el almacenamiento de paquetes en el *buffer* puede ser especificado individualmente en la misma acción de *salida*, y para los demás *packet-in* se puede configurar en la configuración del *switch*.

- **Flujo-Borrado:** Informa al controlador sobre la eliminación de una entrada de flujo de una tabla de flujos. Los mensajes *Flujo-Borrado* sólo son enviados por entradas de flujo con el *flag* `OFPFF_SEND_FLOW_REM` activado. Estos mensajes son generados como resultado de una petición por parte del controlador para borrar un flujo o porque ha expirado un flujo del *switch* debido a que el *timeout* del flujo ha expirado.
- **Estado-Puerto:** Informa al controlador si ha habido un cambio en un puerto. Los cambios que el *switch* debe notificar al controlador mediante mensajes *Estado-Puerto* pueden por ejemplo, ser cambios que haya hecho el usuario en la configuración del puerto, o cambios en el estado del puerto, como por ejemplo que el enlace se venga abajo.
- **Error:** El *switch* es capaz de notificar a los controladores acerca de un problema mediante mensajes de error.

Mensajes *Simétricos*

Los mensajes *simétricos* se envían en cualquier dirección (controlador-*switch*, *switch*-controlador) y sin ser solicitados. Los hay de tres tipos:

- **Hello:** Los mensajes *Hello* se intercambian entre el *switch* y el controlador al inicio de una conexión.
- **Echo:** Los mensajes petición/respuesta de *echo* pueden ser enviados tanto por el *switch* como por el controlador, y deben devolver una respuesta *echo*. La mayoría de veces se utilizan para verificar que la conexión controlador-*switch* sigue activa, aunque también pueden ser utilizados para medir la latencia y ancho de banda de la conexión.
- **Experimenter:** Este tipo de mensajes proporcionan a los *switches* OpenFlow una forma estándar para ofrecer funcionalidades adicionales en el espacio destinado a "tipo de mensaje" OpenFlow. Se trata de un área de ensayo para características de futuras versiones de OpenFlow.

Conexiones del Canal OpenFlow

El Canal OpenFlow es la interfaz que conecta cada *switch* OpenFlow con el controlador. A través de esta interfaz, el controlador configura y administra el *switch*, recibe eventos desde el *switch*, y envía paquetes hacia fuera del *switch*. El Canal OpenFlow normalmente es encriptado mediante TLS[16], pero se puede utilizar sobre TCP[17].

Normalmente un controlador OpenFlow administra varios Canales OpenFlow, cada uno conectado a un *switch* OpenFlow distinto. Por su parte, un *switch* OpenFlow debe tener un Canal OpenFlow que lo conecte con un solo controlador, pudiendo tener más Canales OpenFlow, cada uno conectado a un controlador distinto.

El *switch* debe poder establecer una conexión con un controlador en la dirección IP que haya sido configurada por el usuario, utilizando el puerto especificado. Si el *switch* ya conoce la dirección IP del controlador, el *switch* inicia una conexión estándar TLS o TCP con el controlador. El tráfico de y desde el canal OpenFlow no se procesa a través de la *pipeline* de OpenFlow, por lo que el *switch* debe identificar ése tráfico entrante como local antes de compararlo con las tablas de flujo.

Cuando se establece una conexión OpenFlow, cada extremo de la conexión debe enviar inmediatamente un mensaje `OFPT_HELLO` con el campo *version* configurado con la versión del protocolo OpenFlow más alta que soporte el emisor. Tras la recepción de este mensaje, el destinatario calculará la versión del protocolo OpenFlow a utilizar, comparando el mensaje enviado y el recibido, y eligiendo siempre de los dos mensajes, la versión que tenga el menor número (versión más antigua).

[16] *Transport Layer Security* es un protocolo criptográfico que proporciona comunicaciones seguras por una red, asegurando la autenticación y privacidad de la información.

[17] *Transmission Control Protocol* es uno de los protocolos fundamentales en Internet. Se utiliza para crear "conexiones" entre programas dentro de una red, a través de las cuales puede enviarse un flujo de datos. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron.

Si la versión negociada es soportada por el receptor, entonces la conexión procede. De lo contrario, el receptor debe responder con un mensaje `OFPT_ERROR` que contenga en el campo `type` el tipo de error `OFPET_HELLO_FAILED`, en el campo `code` el valor `OFPHFC_INCOMPATIBLE` y opcionalmente una cadena ASCII explicando la situación en los datos, y finalmente terminar la conexión.

Después de negociar la versión del protocolo, el controlador envía un mensaje `OFPT_FEATURES_REQUEST`, al que el *switch* responde con un mensaje `OFPT_FEATURES_REPLY`. El mensaje de respuesta contiene el número de tablas que soporta el *switch* y su ID *datapath*; además, puede contener una lista con las capacidades del switch:

- Estadísticas de flujo.
- Estadísticas de tablas.
- Estadísticas de puertos.
- Estadísticas de grupos.
- Capacidad para volver a unir fragmentos IP.
- Estadísticas de colas.
- Capacidad de bloquear los puertos de *looping*.

El intercambio de estos mensajes de características es lo que se llama OpenFlow *Handshake* (*apretón de manos*). Después de que el controlador haya recibido el mensaje `OFPT_FEATURES_REPLY`, normalmente le solicita al *switch* sus parámetros de configuración. El controlador establece y solicita los parámetros de configuración del *switch* mediante los mensajes `OFPT_SET_CONFIG` y `OFPT_GET_CONFIG_REQUEST`, respectivamente. Por su parte, el *switch* responde a una solicitud de parámetros de configuración con un mensaje `OFPT_GET_CONFIG_REPLY` (pero no responde a peticiones para establecer los parámetros de configuración).

Los mensajes de configuración contienen la información dentro de un conjunto de *flags* de configuración. Estos *flags* indican si los fragmentos IP deben ser tratados normalmente, descartados o reensamblados. La manipulación "normal" de los fragmentos significa que se intentarán hacer pasar a través de las tablas OpenFlow.

B.2.9. Mensajes de modificación de las Tablas de Flujos

Los tipos de mensajes que pueden tener los mensajes de modificación de las tablas de flujos, son:

- (*Añadir*) **`OFPFC_ADD`:** Añade un nuevo flujo a la tabla de flujos. Si el *flag* `OFPFF_CHECK_OVERLAP` está activado, el *switch* debe primero de todo, comprobar que no haya ninguna entrada de flujo superpuesta en la tabla. Dos entradas de flujo se superponen si un paquete puede coincidir con las dos entradas y ambas tienen la misma prioridad. Si existe un conflicto por superposición entre una entrada de flujo de la tabla y la petición de añadir, el *switch* debe denegar la petición de añadir y responder con un mensaje *ofp_error_msg* con tipo `OFPET_FLOW_MOD_FAILED` y código `OFPFMFC_OVERLAP`.

  Para las peticiones en las que no haya superposición o no haya comprobación de superposición, el *switch* debe insertar la entrada de flujo en la tabla. Si una entrada de flujo con los mismos campos de coincidencia y prioridad ya existe en la tabla, se sobrescribe con la nueva entrada, incluida su duración. Si el *flag* `OFPFF_RESET_COUNTS` está activado, los contadores de la entrada de flujo deben ser limpiados, o bien copiados de la entrada de flujo sobrescrita. Al eliminarse la entrada de flujo antigua, no se generan mensajes de *Flujo-Borrado* debido a que forma parte del proceso de una petición para añadir; si el controlador quiere recibir un mensaje de *Flujo-Borrado*, debe enviar explícitamente una petición de borrado de la antigua entrada de flujo, y luego enviar la petición para añadir la nueva entrada.

- *(Modificar)* **`OFPFC_MODIFY` o `OFPFC_MODIFY_STRICT`:** Estos mensajes se utilizan para enviar peticiones para modificar entradas de flujo de las tablas. Si en la tabla existe una entrada que coincida, el campo de las instrucciones es actualizado con los valores de la petición, donde los campos de la *cookie*, el *idle_timeout*, el *hard_timeout*, los *flags*, los contadores y la duración, se dejan sin modificar.

Si el *flag* `OFPFF_RESET_COUNTS` está activado, los contadores de la entrada de flujo deben limpiarse. Si una petición de modificación no ha encontrado ninguna entrada en la tabla que coincida, no se genera ningún error y tampoco se realiza ninguna modificación en la tabla de flujos.

- *(Borrar)* **`OFPFC_DELETE` o `OFPFC_STRICT`:** Estos mensajes son peticiones para borrar una entrada de flujo de una tabla. Si en la tabla se encuentra una entrada de flujo coincidente, se borra, y si la entrada tiene el *flag* `OFPFF_SEND_FLOW_REM` activado, debe generar un mensaje conforme el flujo ha sido eliminado. Si no se encuentra ninguna entrada de flujo que coincida, no se genera ningún error y tampoco se realiza ninguna modificación en la tabla de flujos.

Los comandos para modificar y borrar una entrada de flujo tienen una versión estricta y otra que no:

- En las versiones estrictas sólo las entradas que coincidan idénticamente, es decir, que todos los campos de la entrada de flujo coincidan con los de la petición, serán las que se modifiquen; si no son idénticas, no se realizará ninguna modificación.

  Por ejemplo, si un mensaje para borrar entradas es enviado y no tiene ningún campo de coincidencia añadido, el comando `OFPFC_DELETE` borrará todas las entradas de flujo de las tablas, mientras que el comando `OFPFC_DELETE_STRICT` sólo borrará la entrada de flujo que afecte a todos los paquetes y que tenga la misma prioridad que se especifique en el mensaje.

- En las versiones no estrictas, los comandos modifican una entrada de flujo si coincide con la descripción del comando. Una coincidencia puede suceder cuando una entrada de flujo coincide exactamente o es más específica que la descripción del comando; en estos comandos, los campos de coincidencia no especificados se consideran comodines, el campo de máscaras está activo, y otros campos de modificación de flujo cómo la prioridad son ignorados.

  Por ejemplo, si un comando `OFPFC_DELETE` dice de borrar todas las entradas de flujo con destino el puerto 80, entonces una entrada que tenga como comodín todos los campos de coincidencia no será borrado. Sin embargo, un comando `OFPFC_DELETE` que tenga como comodín todos sus campos de coincidencia sí que borrará las entradas que lleven tráfico al puerto 80.

Los comandos de modificación y borrado también pueden ser filtrados por el valor de la *cookie*, si el campo `cookie_mask` contiene un valor que no sea 0.

B.2.10. Cabecera OpenFlow

Como ya se ha explicado anteriormente, el protocolo OpenFlow se implementa usando mensajes OpenFlow transmitidos a través del canal OpenFlow. Cada tipo de mensaje es descrito por una estructura específica, la cual siempre empieza con la *Cabecera OpenFlow*. Cada estructura define el orden en que la información se incluye en el mensaje y puede contener otras estructuras, valores, enumeraciones o máscaras de bits.

A continuación se detalla la *Cabecera OpenFlow* con la que empiezan todos los mensajes OpenFlow:

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

El campo `version` especifica la versión del protocolo OpenFlow que el *switch* está utilizando. El bit más significativo del campo `version` está reservado y debe tener valor 0, mientras que los otros siete bits se utilizan para indicar el número de la versión del protocolo. La versión que se describe en este apartado del proyecto es la 1.3, que corresponde con el valor 0x04.

El campo `length` indica la longitud total del mensaje, por lo que no es necesaria ninguna estructura adicional para diferenciar una trama de la siguiente. El campo `type` indica el tipo de mensaje y puede tener los siguientes valores:

```
enum ofp_type {
/* Immutable messages. */
OFPT_HELLO = 0, /* Symmetric message */
OFPT_ERROR = 1, /* Symmetric message */
OFPT_ECHO_REQUEST = 2, /* Symmetric message */
OFPT_ECHO_REPLY = 3, /* Symmetric message */
OFPT_EXPERIMENTER = 4, /* Symmetric message */
/* Switch configuration messages. */
OFPT_FEATURES_REQUEST = 5, /* Controller/switch message */
OFPT_FEATURES_REPLY = 6, /* Controller/switch message */
OFPT_GET_CONFIG_REQUEST = 7, /* Controller/switch message */
OFPT_GET_CONFIG_REPLY = 8, /* Controller/switch message */
OFPT_SET_CONFIG = 9, /* Controller/switch message */
/* Asynchronous messages. */
OFPT_PACKET_IN = 10, /* Async message */
OFPT_FLOW_REMOVED = 11, /* Async message */
OFPT_PORT_STATUS = 12, /* Async message */
/* Controller command messages. */
OFPT_PACKET_OUT = 13, /* Controller/switch message */
OFPT_FLOW_MOD = 14, /* Controller/switch message */
OFPT_GROUP_MOD = 15, /* Controller/switch message */
OFPT_PORT_MOD = 16, /* Controller/switch message */
OFPT_TABLE_MOD = 17, /* Controller/switch message */
/* Multipart messages. */
OFPT_MULTIPART_REQUEST = 18, /* Controller/switch message */
OFPT_MULTIPART_REPLY = 19, /* Controller/switch message */
/* Barrier messages. */
OFPT_BARRIER_REQUEST = 20, /* Controller/switch message */
OFPT_BARRIER_REPLY = 21, /* Controller/switch message */
34 c 2012 The Open Networking Foundation
OpenFlow Switch Specification Version 1.3.0
/* Queue Configuration messages. */
OFPT_QUEUE_GET_CONFIG_REQUEST = 22, /* Controller/switch message */
OFPT_QUEUE_GET_CONFIG_REPLY = 23, /* Controller/switch message */
/* Controller role change request messages. */
OFPT_ROLE_REQUEST = 24, /* Controller/switch message */
OFPT_ROLE_REPLY = 25, /* Controller/switch message */
/* Asynchronous message configuration. */
OFPT_GET_ASYNC_REQUEST = 26, /* Controller/switch message */
OFPT_GET_ASYNC_REPLY = 27, /* Controller/switch message */
OFPT_SET_ASYNC = 28, /* Controller/switch message */
/* Meters and rate limiters configuration messages. */
OFPT_METER_MOD = 29, /* Controller/switch message */
};
```

Apéndice C

Red real gestionada por RouteFlow: Proyecto Cardigan

En la actualidad existe un proyecto que ha puesto a prueba RouteFlow en una red real en producción. Este proyecto se llama Cardigan y consiste en una red SDN que conecta la Red Avanzada de Educación e Investigación de Nueva Zelanda (REANNZ, de sus siglas en inglés), la cual consiste en una Red de Educación e Investigación local de ámbito nacional, con la Wellington Internet Exchange (WIX), un punto de interconexión neutral basado en Ethernet dónde los participantes pueden encontrar a otras personas que quieran intercambiar tráfico IP y que funciona con conmutadores facilitados por la empresa de telecomunicaciones CityLink, situados en Wellington, Nueva Zelanda.

Para poder utilizar RouteFlow en el proyecto Cardigan, fueron necesarias algunas modificaciones de la versión existente al inicio del proyecto para que se ajustara a sus necesidades. Una de esas modificaciones fue añadir el protocolo IPv6, que para aquel entonces aún no incorporaba RouteFlow. Esta y otras modificaciones que se llevaron a cabo en RouteFlow, fueron añadidas al repositorio de versiones Git en una rama de desarrollo paralela a la del proyecto RouteFlow original. Estos cambios que aportó el proyecto Cardigan contribuyeron a la evolución de RouteFlow.

Este proyecto se inició en Enero de 2013 y fue uno de los primeros en desplegar una red SDN en un entorno de producción de todo el mundo. Consistió inicialmente en dos *datapaths* conectados entre sí, uno en la REANNZ y el otro en la WIX, que utilizaban el protocolo OpenFlow 1.2. Cada uno de ellos tenía un puerto conectado a sus respectivas redes, mientras que para conectarse entre ellos, los dos *datapaths* utilizaban un *Inter-Switch Link* (ISL). Las rutas se aprendían desde la WIX y la REANNZ, distribuyéndolas posteriormente a los *datapaths*, dando como resultado aproximadamente 1000 flujos al mismo tiempo en cada *datapath*.

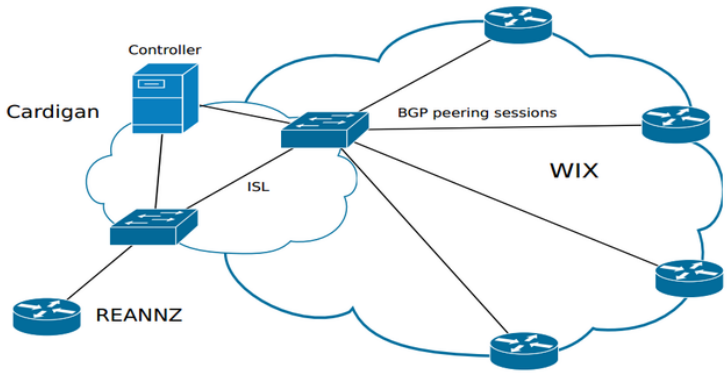


Figura C.1: Arquitectura red del proyecto Cardigan.

Con el tiempo, esta topología se mejoró incorporando varios *datapath* Open vSwitch gestionados por una misma máquina virtual RouteFlow, ofreciendo así una plataforma de enrutamiento centralizada. Esta agregación de múltiples *datapaths* para que funcionaran como una única entidad lógica de nivel 3, significó una reducción en la configuración necesaria, mejoró la consistencia de las políticas y simplificó la estructura de la red. Esta abstracción de la red permite que sea totalmente configurable además de que representa una interfaz de red con la que los operadores de red están familiarizados.

Actualmente el proyecto Cardigan sigue operativo y ha evolucionado al mismo tiempo en que lo hacía su tecnología subyacente, por lo que desde su inicio ha ido incorporando nuevas tecnologías, las cuales le han permitido ofrecer nuevos servicios de red e implementar más protocolos como por ejemplo el MPLS.

Apéndice D

Ficheros de configuración

D.1. Ficheros de configuración contenedores LXC

A modo de consulta en este apéndice se muestran los ficheros de configuración utilizados para la creación de la red gestionada con RouteFlow.

Fichero de configuración del cliente RouteFlow `rfvmA`

```
lxc.utsname = rfvmA
lxc.network.type = veth
lxc.network.flags = up
lxc.network.hwaddr = 02:a0:a0:a0:a0:a0
lxc.network.link=lxcbr0

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rfvmA.1
lxc.network.hwaddr = 02:a1:a1:a1:a1:a1

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rfvmA.2
lxc.network.hwaddr = 02:a2:a2:a2:a2:a2

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rfvmA.3
lxc.network.hwaddr = 02:a3:a3:a3:a3:a3

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rfvmA.4
lxc.network.hwaddr = 02:a4:a4:a4:a4:a4

lxc.devttydir = lxc
lxc.tty = 4
lxc.pts = 1024
lxc.rootfs = /var/lib/lxc/rfvmA/rootfs
lxc.mount  = /var/lib/lxc/rfvmA/fstab
lxc.arch = amd64
lxc.cap.drop = sys_module mac_admin
lxc.pivotdir = lxc_putold

# uncomment the next line to run the container unconfined:
#lxc.aa_profile = unconfined
```

```
38
39 lxc.cgroup.devices.deny = a
40 # Allow any mknod (but not using the node)
41 lxc.cgroup.devices.allow = c *:* m
42 lxc.cgroup.devices.allow = b *:* m
43 # /dev/null and zero
44 lxc.cgroup.devices.allow = c 1:3 rwm
45 lxc.cgroup.devices.allow = c 1:5 rwm
46 # consoles
47 lxc.cgroup.devices.allow = c 5:1 rwm
48 lxc.cgroup.devices.allow = c 5:0 rwm
49 #lxc.cgroup.devices.allow = c 4:0 rwm
50 #lxc.cgroup.devices.allow = c 4:1 rwm
51 # /dev/{,u}random
52 lxc.cgroup.devices.allow = c 1:9 rwm
53 lxc.cgroup.devices.allow = c 1:8 rwm
54 lxc.cgroup.devices.allow = c 136:* rwm
55 lxc.cgroup.devices.allow = c 5:2 rwm
56 # rtc
57 lxc.cgroup.devices.allow = c 254:0 rwm
58 #fuse
59 lxc.cgroup.devices.allow = c 10:229 rwm
60 #tun
61 lxc.cgroup.devices.allow = c 10:200 rwm
62 #full
63 lxc.cgroup.devices.allow = c 1:7 rwm
64 #hpet
65 lxc.cgroup.devices.allow = c 10:228 rwm
66 #kvm
67 lxc.cgroup.devices.allow = c 10:232 rwm
```

Fichero de configuración del cliente RouteFlow `rfvmB`

```
1  lxc.utsname = rfvmB
2  lxc.network.type = veth
3  lxc.network.flags = up
4  lxc.network.hwaddr = 02:b0:b0:b0:b0:b0
5  lxc.network.link=lxcbr0
6
7  lxc.network.type = veth
8  lxc.network.flags = up
9  lxc.network.veth.pair = rfvmB.1
10 lxc.network.hwaddr = 02:b1:b1:b1:b1:b1
11
12 lxc.network.type = veth
13 lxc.network.flags = up
14 lxc.network.veth.pair = rfvmB.2
15 lxc.network.hwaddr = 02:b2:b2:b2:b2:b2
16
17 lxc.network.type = veth
18 lxc.network.flags = up
19 lxc.network.veth.pair = rfvmB.3
20 lxc.network.hwaddr = 02:b3:b3:b3:b3:b3
21
22 lxc.devttydir = lxc
23 lxc.tty = 4
24 lxc.pts = 1024
25 lxc.rootfs = /var/lib/lxc/rfvmB/rootfs
26 lxc.mount  = /var/lib/lxc/rfvmB/fstab
27 lxc.arch = amd64
28 lxc.cap.drop = sys_module mac_admin
29 lxc.pivotdir = lxc_putold
30
```

```
# uncomment the next line to run the container unconfined:
#lxc.aa_profile = unconfined

lxc.cgroup.devices.deny = a
# Allow any mknod (but not using the node)
lxc.cgroup.devices.allow = c *:* m
lxc.cgroup.devices.allow = b *:* m
# /dev/null and zero
lxc.cgroup.devices.allow = c 1:3 rwm
lxc.cgroup.devices.allow = c 1:5 rwm
# consoles
lxc.cgroup.devices.allow = c 5:1 rwm
lxc.cgroup.devices.allow = c 5:0 rwm
#lxc.cgroup.devices.allow = c 4:0 rwm
#lxc.cgroup.devices.allow = c 4:1 rwm
# /dev/{,u}random
lxc.cgroup.devices.allow = c 1:9 rwm
lxc.cgroup.devices.allow = c 1:8 rwm
lxc.cgroup.devices.allow = c 136:* rwm
lxc.cgroup.devices.allow = c 5:2 rwm
# rtc
lxc.cgroup.devices.allow = c 254:0 rwm
#fuse
lxc.cgroup.devices.allow = c 10:229 rwm
#tun
lxc.cgroup.devices.allow = c 10:200 rwm
#full
lxc.cgroup.devices.allow = c 1:7 rwm
#hpet
lxc.cgroup.devices.allow = c 10:228 rwm
#kvm
lxc.cgroup.devices.allow = c 10:232 rwm
```

Fichero de configuración del cliente RouteFlow `rfvmC`

```
lxc.utsname = rfvmC
lxc.network.type = veth
lxc.network.flags = up
lxc.network.hwaddr = 02:c0:c0:c0:c0:c0
lxc.network.link=lxcbr0

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rfvmC.1
lxc.network.hwaddr = 02:c1:c1:c1:c1:c1

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rfvmC.2
lxc.network.hwaddr = 02:c2:c2:c2:c2:c2

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rfvmC.3
lxc.network.hwaddr = 02:c3:c3:c3:c3:c3

lxc.devttydir = lxc
lxc.tty = 4
lxc.pts = 1024
lxc.rootfs = /var/lib/lxc/rfvmC/rootfs
lxc.mount  = /var/lib/lxc/rfvmC/fstab
lxc.arch = amd64
lxc.cap.drop = sys_module mac_admin
```

```
lxc.pivotdir = lxc_putold

# uncomment the next line to run the container unconfined:
#lxc.aa_profile = unconfined

lxc.cgroup.devices.deny = a
# Allow any mknod (but not using the node)
lxc.cgroup.devices.allow = c *:* m
lxc.cgroup.devices.allow = b *:* m
# /dev/null and zero
lxc.cgroup.devices.allow = c 1:3 rwm
lxc.cgroup.devices.allow = c 1:5 rwm
# consoles
lxc.cgroup.devices.allow = c 5:1 rwm
lxc.cgroup.devices.allow = c 5:0 rwm
#lxc.cgroup.devices.allow = c 4:0 rwm
#lxc.cgroup.devices.allow = c 4:1 rwm
# /dev/{,u}random
lxc.cgroup.devices.allow = c 1:9 rwm
lxc.cgroup.devices.allow = c 1:8 rwm
lxc.cgroup.devices.allow = c 136:* rwm
lxc.cgroup.devices.allow = c 5:2 rwm
# rtc
lxc.cgroup.devices.allow = c 254:0 rwm
#fuse
lxc.cgroup.devices.allow = c 10:229 rwm
#tun
lxc.cgroup.devices.allow = c 10:200 rwm
#full
lxc.cgroup.devices.allow = c 1:7 rwm
#hpet
lxc.cgroup.devices.allow = c 10:228 rwm
#kvm
lxc.cgroup.devices.allow = c 10:232 rwm
```

Fichero de configuración del cliente RouteFlow `b2`

```
lxc.utsname = rfvmD
lxc.network.type = veth
lxc.network.flags = up
lxc.network.hwaddr = 02:d0:d0:d0:d0:d0
lxc.network.link=lxcbr0

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rfvmD.1
lxc.network.hwaddr = 02:d1:d1:d1:d1:d1

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rfvmD.2
lxc.network.hwaddr = 02:d2:d2:d2:d2:d2

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rfvmD.3
lxc.network.hwaddr = 02:d3:d3:d3:d3:d3

lxc.network.type = veth
lxc.network.flags = up
lxc.network.veth.pair = rfvmD.4
lxc.network.hwaddr = 02:d4:d4:d4:d4:d4

```

```
lxc.devttydir = lxc
lxc.tty = 4
lxc.pts = 1024
lxc.rootfs = /var/lib/lxc/rfvmD/rootfs
lxc.mount  = /var/lib/lxc/rfvmD/fstab
lxc.arch = amd64
lxc.cap.drop = sys_module mac_admin
lxc.pivotdir = lxc_putold

# uncomment the next line to run the container unconfined:
#lxc.aa_profile = unconfined

lxc.cgroup.devices.deny = a
# Allow any mknod (but not using the node)
lxc.cgroup.devices.allow = c *:* m
lxc.cgroup.devices.allow = b *:* m
# /dev/null and zero
lxc.cgroup.devices.allow = c 1:3 rwm
lxc.cgroup.devices.allow = c 1:5 rwm
# consoles
lxc.cgroup.devices.allow = c 5:1 rwm
lxc.cgroup.devices.allow = c 5:0 rwm
#lxc.cgroup.devices.allow = c 4:0 rwm
#lxc.cgroup.devices.allow = c 4:1 rwm
# /dev/{,u}random
lxc.cgroup.devices.allow = c 1:9 rwm
lxc.cgroup.devices.allow = c 1:8 rwm
lxc.cgroup.devices.allow = c 136:* rwm
lxc.cgroup.devices.allow = c 5:2 rwm
# rtc
lxc.cgroup.devices.allow = c 254:0 rwm
#fuse
lxc.cgroup.devices.allow = c 10:229 rwm
#tun
lxc.cgroup.devices.allow = c 10:200 rwm
#full
lxc.cgroup.devices.allow = c 1:7 rwm
#hpet
lxc.cgroup.devices.allow = c 10:228 rwm
#kvm
lxc.cgroup.devices.allow = c 10:232 rwm
```

D.2. Fichero de configuración del RFServer

A continuación se detalla el código utilizado para la configuración del RFServer de RouteFlow. En él se detalla el mapeo entre los puertos de los RFClients y sus *datapaths*. Los IDs de las máquinas virtuales se han asignado para que sea intuitivo a qué máquina pertenecen. Por ejemplo el ID 2A0A0A0A0A0 corresponde a la `rfvmA`.

```
vm_id,vm_port,ct_id,dp_id,dp_port
2D0D0D0D0D0,1,0,8,1
2D0D0D0D0D0,2,0,8,2
2D0D0D0D0D0,3,0,8,3
2D0D0D0D0D0,4,0,8,4
2C0C0C0C0C0,1,0,7,1
2C0C0C0C0C0,2,0,7,2
2C0C0C0C0C0,3,0,7,3
2B0B0B0B0B0,1,0,6,1
2B0B0B0B0B0,2,0,6,2
2B0B0B0B0B0,3,0,6,3
2A0A0A0A0A0,1,0,5,1
2A0A0A0A0A0,2,0,5,2
2A0A0A0A0A0,3,0,5,3
```

```
15 2A0A0A0A0A0,4,0,5,4
```

Apéndice E

Modificaciones de los *scripts*

E.1. Modificaciones realizadas para la instalación de RouteFlow

En esta sección se explican los pasos y modificaciones que se han realizado para la instalación de RouteFlow. Algunos de los pasos enunciados a continuación, ya han sido explicados anteriormente en este proyecto, por lo que en este apartado no van a ser explicados en detalle.

E.1.1. Descarga de RouteFlow

Para descargar RouteFlow se requiere de la herramienta Git, y para su correcto funcionamiento un conjunto de dependencias:

```
$ sudo apt-get install build-essential git libboost-dev \
  libboost-program-options-dev libboost-thread-dev \
  libboost-filesystem-dev iproute2 openvswitch-switch \
  mongodb python-pymongo
```

Una vez se han descargado las herramientas, se procede a descargar RouteFlow:

```
$ git clone -b vandervecken git://github.com/routeflow/RouteFlow.git
```

E.1.2. Compilar componentes RouteFlow y generación de los elementos de la red

El siguiente paso tras la descarga de RouteFlow, consiste en compilar el código fuente que se ha obtenido. Para ello simplemente hay que ejecutar un *script* que se proporciona en el directorio RouteFlow recién generado, que se llama `build.sh`. Este *script* ejecuta y utiliza otros *scripts* y ficheros de configuración. Estos archivos adicionales han sido modificados para que RouteFlow se instalara correctamente. A continuación se detallan las modificaciones realizadas:

Modificaciones en el *script* `configure.ac`

Uno de los archivos que utiliza el *script* `build.sh`, es uno llamado `configure.ac`. Este archivo es el fichero de configuración de la herramienta *autoconf*, utilizada por `build.sh` para auto-generar los *scripts* que vaya a necesitar. En una de las lineas de código de este fichero, se especifican las librerías que van a ser necesarias. Se ha observado que si se ejecuta `build.sh` con la configuración por defecto, suceden problemas con la conexión SSL del cliente de MongoDB. Para solventar el problema, se han añadido unas librerías adicionales en el fichero de configuración `configure.ac`. A continuación se muestra un extracto del fichero, dónde se marcan en rojo las modificaciones realizadas:

```
AC_INIT([RFClient], m4_esyscmd_s([echo "0.1 ($(git describe --long --tags
--dirty --always))"]),
        [routeflow-discuss@googlegroups.com], [],
        [http://routeflow.github.io/RouteFlow/])
AC_CONFIG_AUX_DIR(config)
AC_CONFIG_SRCDIR(rfclient/RFClient.cc)

AC_PROG_CC
AC_PROG_CC_C_O
AC_PROG_CPP
AC_PROG_CXXCPP
AC_PROG_CXX
AC_PROG_RANLIB
AC_C_BIGENDIAN
AC_TYPE_SIZE_T
AC_LANG([C++])
AC_HEADER_STDC

AM_INIT_AUTOMAKE

AC_CHECK_HEADERS([boost/scoped_array.hpp], [], [AC_MSG_ERROR([Cannot find Boost
system headers.])])
AC_CHECK_HEADERS([boost/thread.hpp], [], [AC_MSG_ERROR([Cannot find Boost
threading headers.])])

LIBS="-lboost_thread -lboost_system -lboost_filesystem
-lboost_program_options -lpthread -lssl -lcrypto"

PKG_CHECK_MODULES(LIBNL3, libnl-route-3.0 >= 3.1, [have_libnl3=yes], [have_libnl3=no])

if (test "${have_libnl3}" = "yes"); then
        CPPFLAGS+=" $LIBNL3_CFLAGS"
        LIBS+=" $LIBNL3_LIBS"
fi

                                                    –continua–
```

Modificaciones en el *script* `build.sh`

Por el mismo motivo que para el *script* anterior, se debe de modificar el *script* `build.sh` para añadir una dependencia adicional. De la misma manera que para el caso anterior, se muestran los cambios realizados en un extracto del código, marcados de color rojo:

```
#!/bin/sh

INSTALL_VMS=0
FETCH_ONLY=0
UPDATE=0
QUIET=1

OVS_VERSION="deb"
MONGO_VERSION="deb"
ZMQ_VERSION="3.2.4"
USE_MONGO=1

RFDIR=`pwd`
CONTROLLERS=""
DO="echo"
SUPER="$DO sudo"
APT_OPTS="-y"
PIP_OPTS=""

ROUTEFLOW_GIT="https://github.com/routeflow/RouteFlow.git"
```

```
21 DEPENDENCIES="build-essential autoconf pkg-config git-core libboost-dev libboost-dev \
22     libboost-program-options-dev libboost-thread-dev \
23     libboost-filesystem-dev libboost-system-dev libnl-3-dev libnl-route-3-dev \
24     python-dev python-pip python-bson libssl-dev"
25
26                                                    --continua--
```

A continuación se actualizan los repositorios de Linux para evitar posibles problemas, y seguidamente se ejecuta el script `build.sh`:

```
# apt-get update
$ ./build.sh -ci ryu
```

Debido a un error que aparece de Open vSwitch, se debe de ejecutar dos veces el *script*. (*Debido a que no es el objetivo de este proyecto y no se disponía del tiempo necesario, no se ha investigado la causa del error.*)

En la preparación de la red de este proyecto, también se han observado problemas al utilizar la herramienta LXC en un sistema operativo Ubuntu 14.04. La solución encontrada para este problema ha consistido en instalar el paquete *selinux* en el PC. Este paquete es incompatible con la herramienta LXC y su dependencia `apparmor`, por lo que al instalar *selinux*, se desinstalará automáticamente LXC. Una vez instalado el paquete, se procederá reinstalar LXC, por lo que `selinux` se desinstalará. (*Debido a que no es el objetivo de este proyecto y no se disponía del tiempo necesario, no se ha investigado la causa del error.*)

Los siguientes comandos se pueden utilizar para realizar la tarea descrita anteriormente:

```
# apt-get install selinux
# apt-get install lxc
```

Una vez se ha realizado el paso anterior, se requiere reiniciar el ordenador.

E.2. Modificaciones en el *script* de ejecución

Para ejecutar la simulación de la red virtual de este proyecto, se ha utilizado como ejemplo el fichero `rftest2` del directorio `rftest`. En este caso el fichero utilizado ha recibido el nombre de `RFtfg`.

Las modificaciones realizadas han sido:

- El parámetro `SCRIPT_NAME` se ha substituido por "`RFtfg`".
- El parámetro `IPC` se ha substituido por "`mongodb`" para que se utilice la base de datos MongoDB para mandar los mensajes IPC.
- Se han substituido todos los comandos `lxc-shutdown` por `lxc-stop`, ya que en las últimas versiones de LXC el primero de los comandos ha dejado de existir, por lo que si no se realiza esta modificación, luego no se podrán detener los contenedores Linux.
- Mejorar el *script* añadiendo un control que no permita seguir la ejecución, si una máquina virtual no se ha iniciado correctamente.
- Indicar que el fichero de configuración del RFServer es `RFtfgconfig.csv`.
- Modificar el mensaje de ayuda que aparece cuando el script ha terminado de ejecutarse, para que indique el comando correcto a ejecutar en la máquina virtual de "mininet".

El fichero `RFtfg` contiene el código siguiente:

```
#!/bin/bash

if [ "$EUID" != "0" ]; then
  echo "You must be root to run this script."
  exit 1
fi

SCRIPT_NAME="RFtfg"
LXCDIR=/var/lib/lxc
IPC="mongodb"
MONGODB_CONF=/etc/mongodb.conf
MONGODB_PORT=27017
MONGODB_ADDR=192.168.10.1
CONTROLLER_PORT=6633
if [ -d ./build ]; then
    RF_HOME=$PWD
else
    RF_HOME=..

export PATH=$PATH:/usr/local/bin:/usr/local/sbin
export PYTHONPATH=$PYTHONPATH:$RF_HOME

cd $RF_HOME

wait_port_listen() {
    port=$1
    while ! `nc -z localhost $port` ; do
        echo -n .
        sleep 1
    done
}

echo_bold() {
    echo -e "\033[1m${1}\033[0m"
}

kill_process_tree() {
    top=$1
    pid=$2

    children=`ps -o pid --no-headers --ppid ${pid}`

    for child in $children
    do
        kill_process_tree 0 $child
    done

    if [ $top -eq 0 ]; then
        kill -9 $pid &> /dev/null
    fi
}

reset() {
    init=$1;
    if [ $init -eq 1 ]; then
        echo_bold "-> Starting $SCRIPT_NAME";
    else
        echo_bold "-> Stopping child processes...";
        kill_process_tree 1 $$
    fi

    ovs-vsctl del-br dp0 &> /dev/null;
    ovs-vsctl emer-reset &> /dev/null;

    echo_bold "-> Stopping and resetting LXC VMs...";
```

```
    for vm in "rfvmA" "rfvmB" "rfvmC" "rfvmD"
    do
        lxc-stop -n "$vm";
        while true
        do
            if lxc-info -q -n "$vm" | grep -q "STOPPED"; then
                break;
            fi
            echo -n .
            sleep 1
        done
    done

    echo_bold "-> Deleting (previous) run data...";
    mongo db --eval "
        db.getCollection('rftable').drop();
        db.getCollection('rfconfig').drop();
        db.getCollection('rfstats').drop();
        db.getCollection('rfclient<->rfserver').drop();
        db.getCollection('rfserver<->rfproxy').drop();
    "

    rm -rf /var/lib/lxc/rfvmA/rootfs/opt/rfclient;
    rm -rf /var/lib/lxc/rfvmB/rootfs/opt/rfclient;
    rm -rf /var/lib/lxc/rfvmC/rootfs/opt/rfclient;
    rm -rf /var/lib/lxc/rfvmD/rootfs/opt/rfclient;
}
reset 1
trap "reset 0; exit 0" INT

echo_bold "-> Setting up the management bridge (lxcbr0)..."
ifconfig lxcbr0 $MONGODB_ADDR up

if [ $IPC = "mongodb" ]; then
        echo_bold "-> Setting up MongoDB..."
        sed -i "/bind_ip/c\bind_ip = 127.0.0.1,$MONGODB_ADDR" $MONGODB_CONF
        service mongodb restart
        wait_port_listen $MONGODB_PORT
fi

echo_bold "-> Starting RFServer..."
nice -n 20 ./rfserver/rfserver.py rftest/RFtfgconfig.csv &

echo_bold "-> Configuring the virtual machines..."

for vm in rfvmA rfvmB rfvmC rfvmD; do
    mkdir "/var/lib/lxc/$vm/rootfs/opt/rfclient"
    cp rfclient/rfclient "/var/lib/lxc/$vm/rootfs/opt/rfclient/rfclient"

    # We sleep for a few seconds to wait for the interfaces to go up
    cat <<EOF > "/var/lib/lxc/$vm/rootfs/root/run_rfclient.sh"
#!/bin/sh
sleep 3
/etc/init.d/quagga start
/opt/rfclient/rfclient 2>&1 >/var/log/rfclient.log
EOF
    chmod +x "/var/lib/lxc/$vm/rootfs/root/run_rfclient.sh"

    echo_bold "-> Starting the virtual machine $vm..."
    lxc-start -n "$vm" -d
    if lxc-info -q -n "$vm" | grep -q "STOPPED"; then
        echo_bold "-> A virtual machine has not started!";
        reset 0;
        exit 0;
    fi
```

```
131 done
132
133 echo_bold "-> Starting the controller and RFPRoxy..."
134 ryu-manager --use-stderr --ofp-tcp-listen-port=$CONTROLLER_PORT ryu-rfproxy/rfproxy.p$
135 wait_port_listen $CONTROLLER_PORT
136
137 echo_bold "-> Starting the control plane network (dp0 VS)..."
138 ovs-vsctl add-br dp0
139 for i in 1 2 3; do
140     for vm in rfvmA rfvmB rfvmC rfvmD; do
141         ovs-vsctl add-port dp0 "$vm.$i"
142     done
143 done
144 ovs-vsctl add-port dp0 rfvmA.4
145 ovs-vsctl add-port dp0 rfvmD.4
146 ovs-vsctl set Bridge dp0 other-config:datapath-id=7266767372667673
147 ovs-vsctl set-controller dp0 tcp:127.0.0.1:$CONTROLLER_PORT
148 ovs-vsctl set Bridge dp0 protocols=OpenFlow13
149 ovs-ofctl -O OpenFlow13 add-flow dp0 actions=CONTROLLER:65509
150
151 echo_bold "---"
152 echo_bold "This test is up and running."
153 echo_bold "Start Mininet:"
154 echo_bold "  $ sudo mn --custom mininet/custom/topo-4sw-4host.py --topo=tfgnet"
155 echo_bold "    --switch ovs,protocols=OpenFlow13"
156 echo_bold "    --controller=remote,ip=[host address],port=6633"
157 echo_bold "    --pre=ipconf"
158 echo_bold "Replace [host address] with the address of this host's interface "
159 echo_bold "connected to the Mininet VM."
160 echo_bold "Then try pinging everything:"
161 echo_bold "  mininet> pingall"
162 echo_bold "You can stop this test by pressing CTRL+C."
163 echo_bold "---"
164 wait
165
166 exit 0
```

Apéndice F

Figuras adicionales sobre RouteFlow

```
PortRegister
   i64 vm_id
   i32 vm_port
   mac hwaddress

PortConfig
   i64 vm_id
   i32 vm_port
   i32 operation_id

DatapathPortRegister
   i64 ct_id
   i64 dp_id
   i32 dp_port

DatapathDown
   i64 ct_id
   i64 dp_id

VirtualPlaneMap
   i64 vm_id
   i32 vm_port
   i64 vs_id
   i32 vs_port

DataPlaneMap
   i64 ct_id
   i64 dp_id
   i32 dp_port
   i64 vs_id
   i32 vs_port

RouteMod
   i8 mod
   i64 id
   match[] matches
   action[] actions
```

```
   option[] options

ControllerRegister
   ip ct_addr
   i32 ct_port
   string ct_role

ElectMaster
   ip ct_addr
   i32 ct_port
```

Figura F.1: Estructura de los mensajes IPC de RouteFlow.

```
from mininet.topo import Topo

class tfgnet(Topo):

    def __init__( self, enable_all = True ):
        "Create custom topo."

        Topo.__init__( self )

        h1 = self.addHost("h1",
                          ip="172.31.1.100/24",
                          defaultRoute="gw 172.31.1.1")

        h2 = self.addHost("h2",
                          ip="172.31.2.100/24",
                          defaultRoute="gw 172.31.2.1")

        h3 = self.addHost("h3",
                          ip="172.31.3.100/24",
                          defaultRoute="gw 172.31.3.1")

        h4 = self.addHost("h4",
                          ip="172.31.4.100/24",
                          defaultRoute="gw 172.31.4.1")

        sA = self.addSwitch("s5")
        sB = self.addSwitch("s6")
        sC = self.addSwitch("s7")
        sD = self.addSwitch("s8")

        self.addLink(h1, sA)
        self.addLink(h2, sB)
        self.addLink(h3, sC)
        self.addLink(h4, sD)
        self.addLink(sA, sB)
        self.addLink(sB, sD)
        self.addLink(sD, sC)
        self.addLink(sC, sA)
        self.addLink(sA, sD)


topos = { 'tfgnet': ( lambda: tfgnet() ) }
```

Figura F.2: Fichero de configuración de topología de “mininet”.

```
DatapathPortRegister
  ct_id: 0x0
  dp_id: 0x7266767372667673
  dp_port: 1

DatapathPortRegister
  ct_id: 0x0
  dp_id: 0x7266767372667673
  dp_port: 2

DatapathPortRegister
  ct_id: 0x0
  dp_id: 0x7266767372667673
  dp_port: 3

DatapathPortRegister
  ct_id: 0x0
  dp_id: 0x7266767372667673
  dp_port: 4

DatapathPortRegister
  ct_id: 0x0
  dp_id: 0x7266767372667673
  dp_port: 5

DatapathPortRegister
  ct_id: 0x0
  dp_id: 0x7266767372667673
  dp_port: 6
```

Figura F.3: Extracto de la salida del script `rfdb.py` con mensajes *DatapathPortRegister*.

```
VirtualPlaneMap
  vm_id: 0x2b0b0b0b0b0b
  vm_port: 1
  vs_id: 0x7266767372667673
  vs_port: 2

VirtualPlaneMap
  vm_id: 0x2b0b0b0b0b0b
  vm_port: 2
  vs_id: 0x7266767372667673
  vs_port: 6

VirtualPlaneMap
  vm_id: 0x2b0b0b0b0b0b
  vm_port: 3
  vs_id: 0x7266767372667673
  vs_port: 10

VirtualPlaneMap
  vm_id: 0x2a0a0a0a0a0a
  vm_port: 1
  vs_id: 0x7266767372667673
  vs_port: 1

VirtualPlaneMap
  vm_id: 0x2a0a0a0a0a0a
```

```
  vm_port: 2
  vs_id: 0x7266767372667673
  vs_port: 5
```

Figura F.4: Extracto de la salida del script `rfdb.py` con mensajes *VirtualPlaneMap*.

```
PortRegister
  vm_id: 0x2a0a0a0a0a0a0
  vm_port: 1
  hwaddress: 02:a1:a1:a1:a1:a1dicho

PortRegister
  vm_id: 0x2a0a0a0a0a0a0
  vm_port: 2
  hwaddress: 02:a2:a2:a2:a2:a2

PortRegister
  vm_id: 0x2a0a0a0a0a0a0
  vm_port: 3
  hwaddress: 02:a3:a3:a3:a3:a3

PortRegister
  vm_id: 0x2a0a0a0a0a0a0
  vm_port: 4
  hwaddress: 02:a4:a4:a4:a4:a4

PortRegister
  vm_id: 0x2b0b0b0b0b0b0
  vm_port: 1
  hwaddress: 02:b1:b1:b1:b1:b1

PortRegister
  vm_id: 0x2b0b0b0b0b0b0
  vm_port: 2
  hwaddress: 02:b2:b2:b2:b2:b2
```

Figura F.5: Extracto de la salida del script `rfdb.py` con mensajes *PortRegister*.

```
VirtualPlaneMap
  vm_id: 0x2c0c0c0c0c0c0
  vm_port: 2
  vs_id: 0x7266767372667673
  vs_port: 7

VirtualPlaneMap
  vm_id: 0x2c0c0c0c0c0c0
  vm_port: 1
  vs_id: 0x7266767372667673
  vs_port: 3

DataPlaneMap
  ct_id: 0x0
  dp_id: 0x7
  dp_port: 3
  vs_id: 0x7266767372667673
  vs_port: 11

DataPlaneMap
```

```
  ct_id: 0x0
  dp_id: 0x7
  dp_port: 2
  vs_id: 0x7266767372667673
  vs_port: 7

DataPlaneMap
  ct_id: 0x0
  dp_id: 0x7
  dp_port: 1
  vs_id: 0x7266767372667673
  vs_port: 3

RouteMod
  mod: 2
  id: 0x7
  vm_port: 0
  table: 0
  group: 0
  meter: 0
  flags: 0
  matches:
    RFMT_ETHERNET : 02:c3:c3:c3:c3:c3
    RFMT_IPV4 : ('30.0.0.3', '255.255.255.255')
    RFMT_NW_PROTO : 1
  actions:
    RFAT_OUTPUT : 4294967293
  options:
    RFOT_PRIORITY : 32800
    RFOT_CT_ID : 0
  bands:
```

Figura F.6: Extracto de la salida del script `rfdb.py` con algunos mensajes IPC de RouteFlow(1).

```
PortConfig
  vm_id: 0x2a0a0a0a0a0
  vm_port: 1
  operation_id: 2

PortConfig
  vm_id: 0x2a0a0a0a0a0
  vm_port: 2
  operation_id: 2
informar del mapeo
PortConfig
  vm_id: 0x2a0a0a0a0a0
  vm_port: 3
  operation_id: 2

PortConfig
  vm_id: 0x2a0a0a0a0a0
  vm_port: 4
  operation_id: 2

RouteMod
  mod: 0
  id: 0x2d0d0d0d0d0
  vm_port: 4
  table: 0
  group: 0
  meter: 0
  flags: 0
```

```
  matches:
    RFMT_IPV6 : ('ff02::2', 'ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff')
  actions:
    RFAT_SET_ETH_SRC : 02:d4:d4:d4:d4:d4
    RFAT_SET_ETH_DST : 33:33:00:00:00:02
  options:
    RFOT_PRIORITY : 17680
  bands:

RouteMod
  mod: 2
  id: 0x2d0d0d0d0d0
  vm_port: 3
  table: 0
  group: 0
  meter: 0
  flags: 0
  matches:
    RFMT_ETHERNET : 02:d3:d3:d3:d3:d3
    RFMT_IPV4 : ('20.0.0.4', '255.255.255.255')
    RFMT_NW_PROTO : 1
  actions:
  options:
    RFOT_PRIORITY : 32800
  bands:
```

Figura F.7: Extracto de la salida del script `rfdb.py` con algunos mensajes IPC de RouteFlow(2).

Glosario

A

acciones OpenFlow

operaciones que envían un paquete dado a un puerto o lo modifican, por ejemplo decrementando su campo TTL. 47

Address Resolution Protocol

Address Resolution Protocol es un protocolo de comunicaciones de la capa de enlace de datos, responsable de encontrar la dirección de hardware (Ethernet MAC) que corresponde a una determinada dirección IP. 14

API

conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos. Representa un método para conseguir abstracción en la programación, entre los niveles o capas inferiores y los superiores del software. 42

aplicaciones de la nube

aplicaciones que están alojadas en servidores de Internet. 12

B

big-endian

formato que define el orden en que los bytes de un objeto son transmitidos o almacenados. En este formato el byte más significativo, que es el que contiene el bit más significativo (bit que está en la posición más alta dentro un número binario), se almacena primero y los siguientes bytes se almacenan por orden de importancia, siendo el byte menos significativo el que ocupe el último lugar. 18

bridge

dispositivo de interconexión de redes que opera en la capa 2 del modelo OSI y de funcionalidad muy parecida a la de un *switch*. La diferencia más importante entre un *bridge* y un *switch* es que los primeros normalmente tienen un número pequeño de interfaces (de dos a cuatro), mientras que los *switches* pueden llegar a tener docenas. 13

broadcast

forma de transmisión de información donde un nodo emisor envía información a todos los nodos receptores de su red de manera simultánea. 55

BSON

formato de intercambio de datos usado principalmente para su almacenamiento y transferencia en la base de datos MongoDB. Es una representación binaria de estructuras de datos y mapas. Un objeto BSON consiste en una lista ordenada de elementos. Cada elemento consta de un campo nombre, un tipo y un valor. 18

bucket

tipo de buffer de datos o documento, en que los datos están divididos en regiones. 47

C

circuito virtual

forma de comunicación mediante conmutación de paquetes en la cual la información o datos son empaquetados en bloques que tienen un tamaño variable a los que se les denomina paquetes de datos. En los circuitos virtuales, al comienzo de la sesión se establece una ruta única entre las entidades terminales de datos o los host extremos. A partir de aquí, todos los paquetes enviados entre estas entidades seguirán la misma ruta. 19

comma-separated values

tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea. 15

conmutación de paquetes

método de envío de datos en una red de computadoras. Un paquete es un grupo de información que consta de dos partes: los datos propiamente dichos y la información de control, que indica la ruta a seguir a lo largo de la red hasta el destino del paquete. 19

D

DiffServ

los Servicios Diferenciados (DiffServ) proporcionan un método que intenta garantizar la calidad de servicio en redes de gran tamaño, como puede ser Internet. 56

Dijkstra

algoritmo para la determinación del camino más corto desde un vértice origen al resto de los vértices en un grafo con pesos en cada arista. El algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene. 21

dispositivo propietario

dispositivos en que los fabricantes no permiten acceder a la documentación y por lo tanto se desconoce cómo escribir drivers para su hardware. 42

E

estructuras tipo/longitud/valor

forma de codificar los datos, de manera que haya información que pueda tener presencia opcional y longitud variable. Esta codificación se basa en tres campos: tipo (especifica el tipo de dato), longitud (indica cuantos bytes ocupa el valor) y valor (valor a codificar). 17

extensiones (de protocolo)

información adicional que algunos protocolos de enrutamiento encapsulan en sus mensajes, que puede ser utilizada por otros protocolos. De esta manera, algunos protocolos de enrutamiento pueden aprovechar esta información para reducir el número de mensajes que deba transmitir. 21

F

Forwarding Information Base

tabla que se utiliza en el enrutamiento y funciones similares para encontrar la interfaz apropiada a la que la interfaz de entrada debe enviar un paquete, que asigna dinámicamente las direcciones MAC de los puertos. Es el mecanismo esencial que separa los *switches* de los *hubs*. 15

Forwarding Path Manager

interfaz propuesta por el grupo Open Source Routing (OSR), que tiene por objetivo permitir a los motores de enrutamiento exponer la información de encaminamiento de manera independiente de la plataforma mediante un administrador. 13

G

gateways

equipo informático configurado para dotar a las máquinas de una red de área local conectadas a él de un acceso hacia una red exterior. 14

H

host

término usado en informática para referirse a las computadoras conectadas a una red. 12

I

imagen (de máquina virtual)

archivo informático donde se almacena una copia o imagen exacta de un sistema de archivos o ficheros de un disco óptico, normalmente un disco compacto (CD) o un disco versátil digital (DVD). Algunos de los usos más comunes incluyen la distribución de sistemas operativos, por ejemplo: GNU/Linux, BSD y Live CD. 26

instrucciones OpenFlow

describen como OpenFlow procesa los paquetes que coinciden con una entrada de flujos. Pueden dirigir los paquetes a otra tabla de flujo, añadir acciones al Conjunto de Acciones a realizar o aplicar directamente una Lista de Acciones sobre los paquetes. 46

K

kernel

software que constituye una parte fundamental del sistema operativo. Es el principal responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o dicho de otra manera, es el encargado de gestionar recursos, a través de servicios de llamada al sistema. 13

L

loopback

la dirección de *loopback* es una dirección especial que los *hosts* utilizan para dirigir el tráfico hacia ellos mismos. 47

M

metadatos

información no relevante para el usuario final pero sí de suma importancia para el sistema que maneja los datos. 42

multicast

consiste en el envío de información a un grupo seleccionado de destinos simultáneamente. 55

P

ping

utilidad diagnóstica en redes de computadoras que comprueba el estado de la comunicación del *host* emisor con uno o varios equipos remotos de una red IP por medio del envío de paquetes ICMP de solicitud (ICMP Echo Request) y de respuesta (ICMP Echo Reply). Mediante esta utilidad puede diagnosticarse el estado, velocidad y calidad de una red determinada. 30

pipeline

conjunto de tablas de flujos vinculadas que proporcionan coincidencias, reenvío y modificación de paquetes en un *switch* OpenFlow. 47

Platform-as-a-Service

modelo de una red virtualizada en la que el control de la infraestructura está separado del control del servicio (paradigma SDN) y el plano de control se gestiona con la abstracción de un sólo *router*, a diferencia de otros modelos dónde el plano de control es gestionado por una red virtual formado por múltiples *routers* virtuales. Con este modelo se consigue que el *router* se centre únicamente en su servicio, en vez de tener que preocuparse también de gestionar la red virtual. 19

protocolo Netlink

los *sockets* Netlink son interfaces del kernel de Linux que se utilizan para comunicaciones entre procesos (IPC) entre los procesos del kernel y los del espacio de usuario, y también entre los propios procesos del espacio de usuario, de manera similar a los *sockets* de dominio Unix. 13

R

reglas de coincidencia

criterios utilizados para comparar valores de distintos campos, en busca de coincidencias. 45

router

dispositivo que proporciona conectividad a nivel de red o nivel 3 en el modelo OSI. Su función principal consiste en enviar o encaminar paquetes de datos de una red a otra. 45

Routing-as-a-Service

concepto que consiste en ofrecer como servicio el cálculo personalizado de rutas, ofreciendo a los usuarios la posibilidad de elegir el tipo de control que tienen sobre el enrutamiento de su tráfico. 11

S

salto entre dispositivos

con ”salto” nos referimos al envío de un paquete de un dispositivo a otro. 47

Simple Round Robin

método para seleccionar todos los elementos en un grupo de manera equitativa y en un orden racional, normalmente comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento. 56

switch

dispositivo digital lógico de interconexión de equipos que opera en la capa de enlace de datos del modelo OSI. Su función es interconectar dos o más segmentos de red. También pueden llamarse conmutadores. 45

T

Time-To-Live

el TTL (Time-To-Live) es un concepto usado en redes de computadores para indicar por cuántos nodos puede pasar un paquete antes de ser descartado por la red o devuelto a su origen. 54

tráfico de control *out-of-band*

tráfico de control de una red que debido a su contenido de carácter sensible, se transmite de manera que no se utilicen los mismos canales de transmisión que los del tráfico de datos. 22

Transmission Control Protocol

Transmission Control Protocol es uno de los protocolos fundamentales en Internet. Se utiliza para crear "conexiones" entre programas dentro de una red, a través de las cuales puede enviarse un flujo de datos. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. 58

Transport Layer Security

Transport Layer Security es un protocolo criptográfico que proporciona comunicaciones seguras por una red, asegurando la autenticación y privacidad de la información. 58