Universidad de la República
Facultad de Ingeniería

SmartRoute: Arquitectura de red basada en SDN para optimización de ruteo

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

María Angelina Berrutti Cortela, Mariana Caballero
Ruy Lopez, Jeremías Laporte Susenna

en cumplimiento parcial de los requerimientos
para la obtención del título de
Ingeniero Electricista.

Tutores

Claudina Rattaro	Universidad de la República
Gabriel Gómez	Universidad de la República
Martín Randall	Universidad de la República

Tribunal

Eduardo Cota	Universidad de la República
Federico Rodríguez	Universidad de la República
Isabel Amigo	Universidad de la República

Montevideo
domingo 28 enero, 2024

SmartRoute: Arquitectura de red basada en SDN para optimización de ruteo, María Angelina Berrutti Cortela, Mariana Caballero Ruy Lopez, Jeremías Laporte Susenna.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 127 páginas.
Compilada el domingo 28 enero, 2024.
http://iie.fing.edu.uy/

Agradecimientos

Queremos expresar nuestro agradecimiento a tutores, amigos y familiares por su apoyo y orientación, que han sido fundamentales en nuestro camino hacia la culminación de este proyecto. También a la Universidad de la República por brindarnos la oportunidad de desarrollarnos académicamente.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

En este proyecto se logró la implementación de una arquitectura basada en Redes Definidas por Software (SDN, por su sigla en inglés) para la optimización del ruteo de paquetes en la red. Con el objetivo de implementar la arquitectura mencionada, se utilizó el controlador SDN OpenDayLight (ODL) y switches OpenFlow. El controlador se comunica con aplicaciones capaces de realizar un manejo de los flujos, mediciones de costos de cada ruta y elección de caminos.

En base a una arquitectura definida en trabajos previos, se diseñó una aplicación llamada *SmartRoute* que engloba las funcionalidades mencionadas anteriormente. *SmartRoute* está formada por un módulo de ingeniería de tráfico (*TEApp*) que se encarga de monitorear la red y tomar decisiones de ruteo, otro módulo que permite realizar medidas del parámetro *Round-Trip Time* (RTT) asociado a las rutas presentes en la red (*MeasureApp*), y un módulo que lleva a cabo el manejo de flujos y configuración de los switches considerando los datos obtenidos y decisiones tomadas en las demás etapas (*RouteApp*).

Se realizaron pruebas de funcionalidad de la aplicación en un ambiente simulado con Mininet y en una maqueta de laboratorio, validando la implementación de políticas de ruteo en la red, el mecanismo de medición de rutas y el algoritmo de ingeniería de tráfico diseñado.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 1

Introducción

En el presente capítulo se introducen los principales problemas que motivan el desarrollo de este proyecto junto con los objetivos específicos. En conjunto con lo anterior, se brinda una descripción de los trabajos que han sido tomados como antecedentes para la implementación y desarrollo del sistema.

1.1. Motivación

En la actualidad, la evolución tecnológica y el crecimiento exponencial del tráfico de datos plantean nuevos desafíos para las redes de comunicaciones. La creciente demanda de servicios en la nube y aplicaciones en línea motiva el estudio de alternativas para ofrecer ciertos niveles de calidad de servicio (Quality of Service, QoS) sobre Internet. La transmisión de datos en tiempo real, como videoconferencias, streaming de vídeo y juegos en línea, requiere una conexión confiable con baja latencia y ancho de banda garantizado. Sin embargo, la gestión tradicional de redes ha demostrado ciertas limitaciones en términos de flexibilidad, escalabilidad y eficiencia en la administración de recursos.

En Internet los flujos de datos se enrutan de acuerdo a las políticas del protocolo Border Gateway Protocol (BGP). Si bien BGP ha sido fundamental en el funcionamiento de Internet, presenta limitaciones considerables en términos de escalabilidad y eficiencia. Internet está formado por diferentes Sistemas Autónomos (AS) que son administrados por los proveedores de servicios de Internet (Internet Service Provider, ISP). En este escenario, el protocolo BGP determina las rutas basándose en la mínima cantidad de saltos entre los AS y en criterios económicos o políticos de los ISP. Está demostrado que las rutas propuestas por BGP no siempre son las mejores en términos de QoS, además de que en situaciones de falla los tiempos de propagación y restauración son muy altos, pudiendo alcanzar las decenas de minutos, como se plantea en [23], [24], [25].

Ante estas limitaciones, una alternativa interesante es el paradigma de las redes definidas por software (Software Defined Networks, SDN). SDN separa la capa de control de la capa de datos, permitiendo una gestión más centralizada y programable de la red. Esto ofrece la flexibilidad necesaria para adaptarse a las

demandas de QoS y eficiencia de tráfico, lo que puede abordar los desafíos actuales de Internet. Mediante su uso es posible buscar un nivel de QoS que sea conveniente para el cliente y ofrezca escalabilidad en su implementación. Los parámetros que se tomen como referencia de QoS pueden ser elegidos dependiendo de las necesidades de la red, la calidad y tipo de servicio que se desee brindar.

1.2. Redes definidas por software (SDN)

La arquitectura SDN surge a causa de que las redes tradicionales están presentando falencias para gestionar el gran volumen de datos en la red de Internet y los nuevos servicios y modalidades como lo son el streaming, servicios móviles, big data y servicios cloud [16].

Para hacer frente a los desafios mencionados, en esta arquitectura se plantea separar el plano de control y el de datos. Como se muestra en la Figura 1.1, esto se realiza ubicando el plano de datos en los equipos distribuidos e implementando un nuevo agente denominado controlador para gestionar el plano de control.

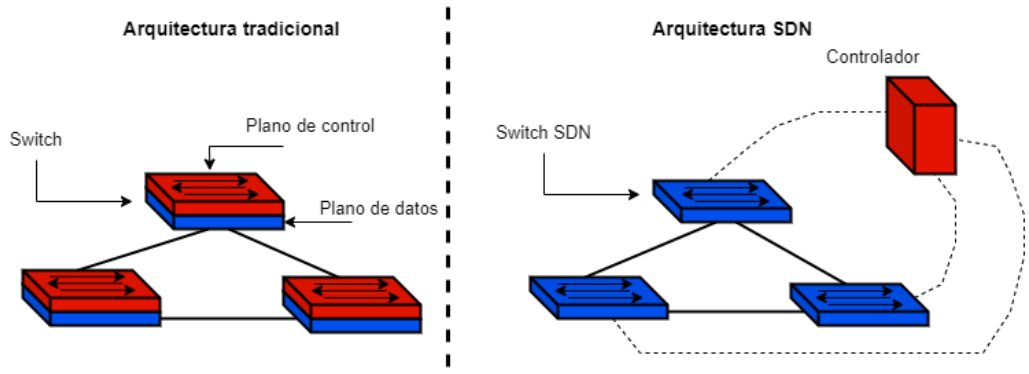


Figura 1.1: Arquitectura tradicional VS Arquitectura SDN.

La arquitectura SDN cuenta con las siguientes ventajas respecto a la tradicional [1]:

- Reducción de costos: Al realizar la separación de planos el controlador es el que centraliza el procesamiento reduciendo así el costo del hardware distribuido necesario. Además, las redes de gran porte son mas fáciles de diseñar, administrar y manejar, por lo que se requiere menor personal para la gestión y el mantenimiento.
- Utilización eficiente de los recursos de red: El controlador es capaz de visualizar los recursos que necesitan las aplicaciones y los que tiene disponibles en la red, lo que resulta en un manejo eficiente de las comunicaciones.
- Reducción del tiempo de caída de servicios: Como se mencionó anteriormente, el controlador SDN es capaz de visualizar los equipos de red que administra. Esto resulta en que ante una falla en uno de los equipos pueda tomar la acción de utilizar otro camino, salteando así el elemento defectuoso.

- Integración con servicios en la nube: La arquitectura SDN facilita la conectividad con servicios en la nube. Las organizaciones pueden gestionar el tráfico de red de forma centralizada logrando priorizar y optimizar los accesos a servicios basados en la nube como aplicaciones, almacenamientos y respaldos.

- Naturaleza de recurso libre: La arquitectura SDN se concibe con el objetivo de generar estándares libres y neutrales frente a los fabricantes. A diferencia de las redes tradicionales en donde para distintos fabricantes se tenían diferentes formas de administración, la estandarización de la administración genera independencia del fabricante lo que resulta en una reducción de los tiempos de implementación.

1.2.1. Redes sobrepuestas en el ámbito de SDN

Una red sobrepuesta es una capa de red virtual, que se ubica sobre una infraestructura de red física para proporcionar funcionalidades específicas. Proveen una forma de estructurar la comunicación entre dispositivos en una red (como Internet), aunque los mismos no estén conectados entre sí físicamente, pues utilizan enlaces virtuales. Las decisiones de enrutamiento dentro de las redes sobrepuestas bajo la arquitectura SDN no son realizadas por los routers y switches, sino por un software externo que conoce toda la topología y conexiones. Esto permite optimizar el ruteo de datos y reducir la latencia, ya que las rutas pueden ser más directas y específicas en comparación con las ya existentes. Estas redes poseen una escalabilidad mayor que las redes tradicionales, puesto que se pueden agregar nuevos nodos simplemente mediante la instalación de software y la configuración de conexiones virtuales sin cambiar la infraestructura física.

Ruteo en redes sobrepuestas

En el contexto de una red sobrepuesta bajo el paradigma SDN, el ruteo se refiere a cómo se determina el camino que deben seguir los paquetes de datos en el tipo de red mencionada. El controlador SDN desempeña un rol clave en esta función, debido a que como se mencionó anteriormente posee una vista completa de toda la red y puede tomar decisiones de enrutamiento basadas en políticas definidas por el administrador de la red. En el presente proyecto se tienen en cuenta dos áreas fundamentales en el ruteo, como son las siguientes:

- Políticas de ruteo: El administrador de la red puede definir políticas de ruteo que especifican cómo deben enrutarse los paquetes en la red sobrepuesta. Estas políticas pueden basarse en parámetros de calidad de servicio como lo es el tiempo de ida y vuelta (RTT).

- Actualizaciones Dinámicas: Una de las ventajas que posee el enfoque SDN es la capacidad de realizar actualizaciones dinámicas en la configuración de enrutamientos. Respecto a las redes tradicionales esto permite una adaptación rápida a cambios en la red o en las necesidades del tráfico de datos.

Para abordar los dos puntos anteriores se utilizan fuertemente [21] y [26] las cuales dan herramientas para la realización de una red sobrepuesta bajo el paradigma SDN y presentan métodos para medición y decisión de las rutas a elegir.

1.3. Antecedentes

En esta sección se presentará una breve descripción de trabajos anteriores enfocados al ruteo de paquetes en redes sobrepuestas y sus diferentes aspectos, como la medición y elección de rutas de acuerdo con la calidad de servicio deseada. Se han presentado algoritmos que fueron tomados como punto de partida para la implementación y desarrollo del sistema.

En trabajos como [17, 18], se propone una arquitectura de red basada en SDN para el ruteo en redes sobrepuestas en base a métricas de QoS. Se plantea la integración de aplicaciones de monitoreo de red (*MonApp*) e ingeniería de tráfico (*TEApp*) que en conjunto mantienen información del estado de la red y computan el mejor camino en términos de QoS. Respecto a *MonApp* se cuenta con algoritmos que surgen de la tesis [26].

Partiendo del trabajo mencionado, en [21] se propone la arquitectura de la Figura 1.2 y la implementación de ONRApp (Overlay Network Routing Application), una aplicación que apunta al enrutamiento y medición de QoS en una red sobrepuesta, dejando la implementación de algoritmos de monitoreo e ingeniería de tráfico para trabajos futuros. Plantea el diseño de aplicaciones encargadas del encaminamiento de datos (RouteApp) y medición de QoS (MeasureApp), haciendo uso del protocolo OpenFlow y el controlador ONOS. Su arquitectura interna se muestra en la Figura 1.3, y sus componentes serán explicados más adelante. En relación con lo propuesto en [21], se han desarrollado implementaciones básicas de *RouteApp* sobre otro controlador SDN, en particular el controlador OpenDaylight, junto con pruebas en maquetas de laboratorio en trabajos como [19, 22].

A continuación, se describe la funcionalidad de las cuatro aplicaciones que conforman la arquitectura propuesta en los trabajos mencionados:

- *TEApp*:
  Esta aplicación es la encargada de definir la estrategia de forwarding. Decide por qué ruta conviene enviar el tráfico a partir de decisiones de *MonApp* y en base a resultados de medición de QoS. Usa *RouteApp* para configurar las políticas de encaminamiento correspondientes.

- *MonApp*:
  Esta aplicación es responsable de decidir qué caminos medir en cada momento. Solicita mediciones de QoS de los posibles caminos a *MeasureApp* y sus salidas van a *TEApp*.

- RouteApp:
  Esta aplicación se encarga de manejar los flujos de datos, permitiendo configurar políticas de encaminamiento personalizadas entre los puntos de presencia ($PoPs$) de la red sobrepuesta.

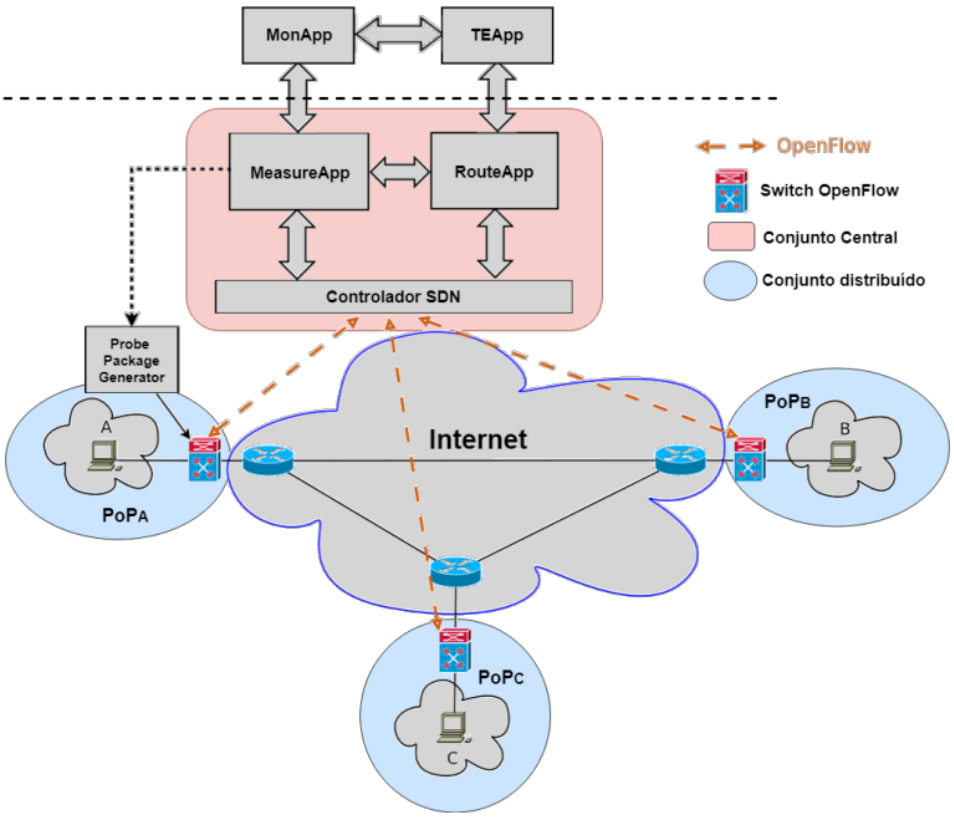


Figura 1.2: Arquitectura de red basada en SDN propuesta en trabajos previos [21].

- MeasureApp:
  Esta aplicación es responsable de la obtención de las métricas de QoS de los posibles caminos en la red sobrepuesta. Para ello envía órdenes a los Probe Packet Generator (PPG), definidos en la sección 3.7, y utiliza los servicios de *RouteApp*.

En el presente proyecto se propone trabajar en base a la arquitectura completa. Se diseña una aplicación que permite tanto el encaminamiento y medición de rutas en una red sobrepuesta así como la aplicación de un algoritmo de ingeniería de tráfico que selecciona la mejor ruta en términos de QoS.

A continuación se repasarán los aspectos más importantes de *RouteApp* y *MeasureApp* (ambos desarrollos del trabajo previo [21]) y las básicas del algoritmo propuesto en la tesis [26] que será la base del desarrollo de *MonApp* y *TEApp* en este trabajo.

1.3.1. RouteApp

RouteApp es encargado de realizar el ruteo de paquetes en una red sobrepuesta, además de tomar las acciones necesarias para que dicho enrutamiento sea

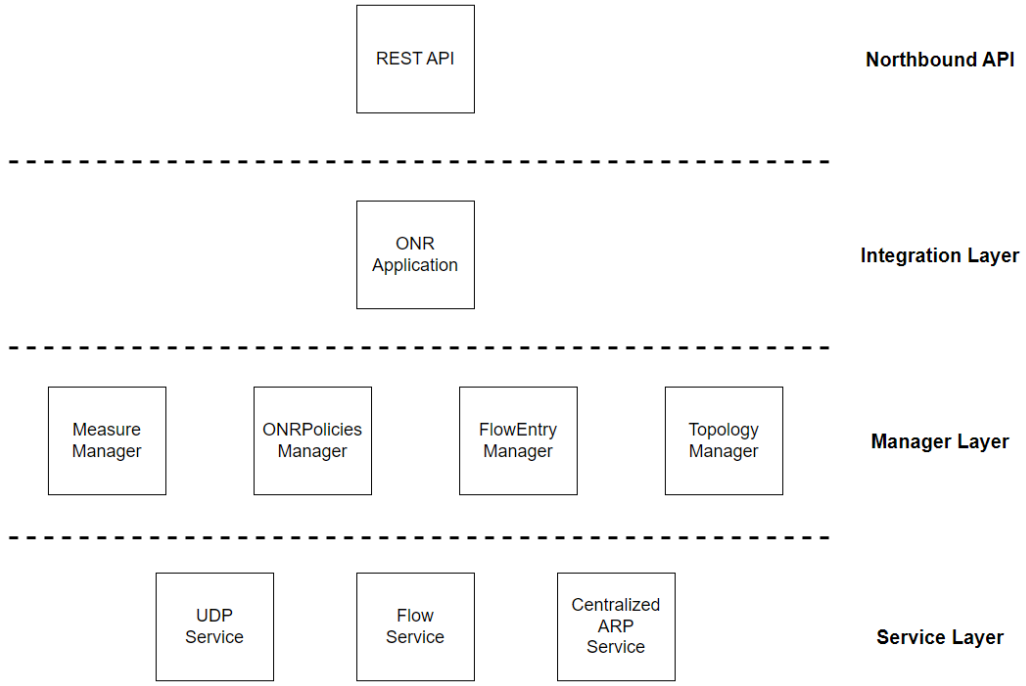


Figura 1.3: Arquitectura interna de ONRApp.

posible. Esto incluye registrar y almacenar información relevante sobre cada punto de presencia (PoP), los enlaces que los conectan, y gestionar las políticas utilizadas para el procesamiento y envío de datos en diferentes flujos de tráfico. El bloque *Topology Manager* (ver figura 1.3) se encarga del registro de PoPs y enlaces, y sus datos relevantes.

Los datos que se registran en esta sección son los siguientes:
Para cada PoP:

- ID del switch
- MAC del switch
- IP del switch
- IP del router del Internet Service Provider (ISP)
- Subred asociada
- Probe Packet Generator (PPG) asociado

Y para cada link (son unidireccionales):

- PoP de origen
- PoP de destino

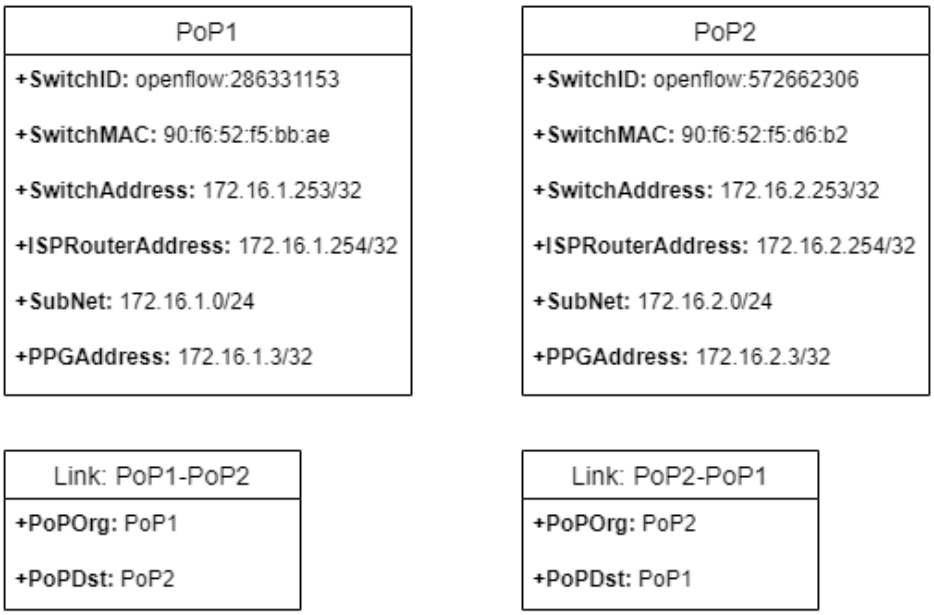


Figura 1.4: Ejemplo de información almacenada al registrar dos PoPs y dos links entre ellos.

Una vez que se tiene una topología de red (al menos dos PoPs y un enlace que los une, como se observa en la Figura 1.4), es posible enfocarse en el encaminamiento de paquetes. Para esto existe el bloque *ONRPolicies Manager* (ver figura 1.3). El mismo recibe solicitudes de políticas de encaminamiento (*Overlay Network Routing Policy, ONRP*), y se encarga de ejecutar las acciones necesarias.

Cada política de encaminamiento cuenta con dos parámetros que la identifican: *Global ONRP ID*, que es único para cada política, y un conjunto de *Local ONRP IDs*, que corresponden a cada enlace incluido en el *Path* (camino que siguen los paquetes desde el PoP de origen al de destino). Si un link es utilizado en una ONRP, el Local ONRP ID que se asocia no puede repetirse para el mismo link en una ONRP diferente. Además, una ONRP puede encontrarse asociada a más de un flujo de tráfico, pero cada flujo se maneja por una única ONRP.

El servicio en cuestión permite entonces la creación, borrado y modificación de políticas de ruteo. El último de estos casos es aplicable cuando se desea modificar la ruta por la que se envía un determinado flujo de tráfico, ya sea por su estado (la que se utilizaba deja de ser la mejor), porque se dio de baja, o se detectó un problema. Esto presenta una ventaja respecto a la opción de eliminar una ONRP y crear otra para el mismo flujo (con diferente ruta), pues reduce la pérdida de paquetes durante el proceso de redirección.

Algoritmo de encaminamiento

Para definir la forma de enrutar el tráfico por la red, se propone en [21] la idea de modificar en cada switch intermedio las direcciones IP de la siguiente forma: en el campo de IP de origen, la del PoP en el que se está realizando la modificación,

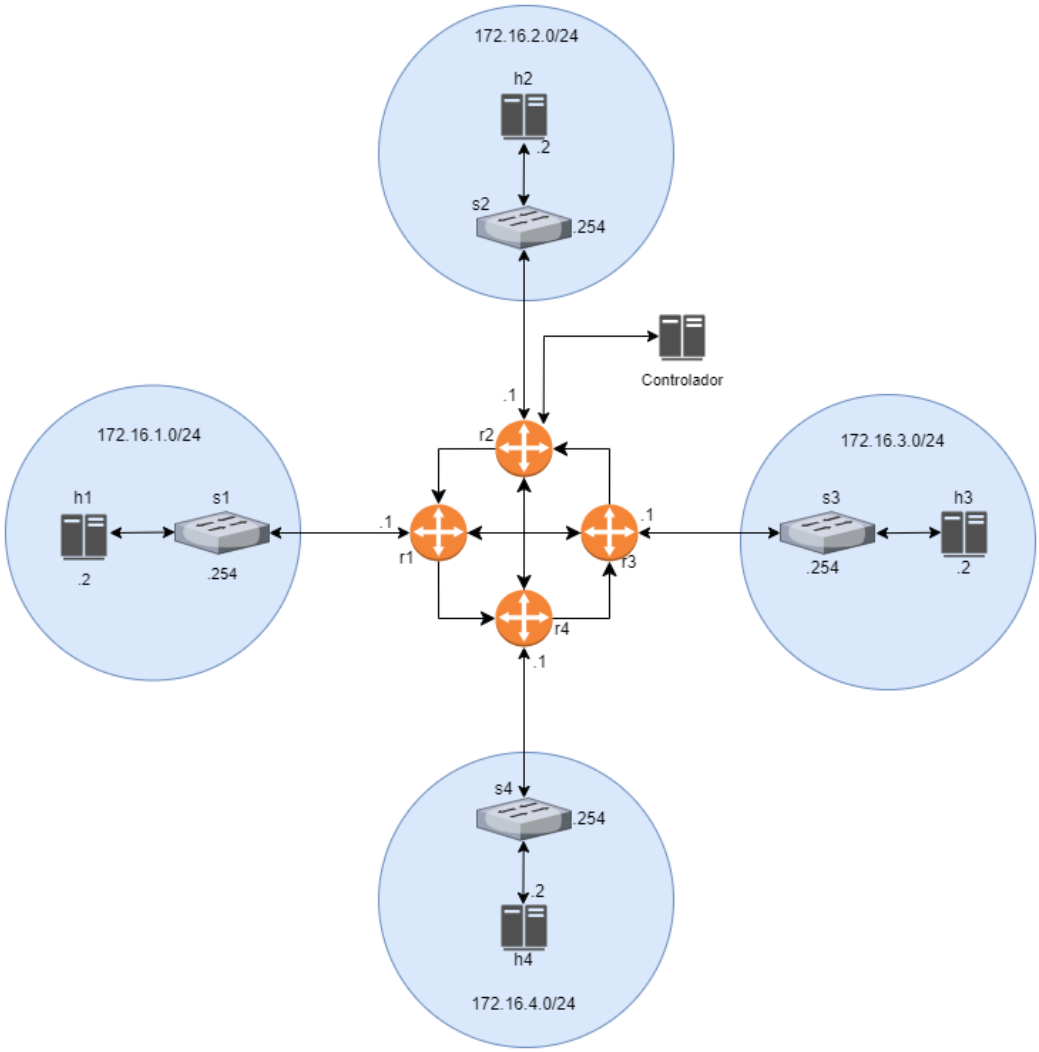


Figura 1.5: Red con 4 PoPs.

y en el de destino, la del PoP que será el *next hop* para el paquete. Observando la topología de la Figura 1.5, si un paquete se quiere enviar desde h1 a h3 pasando por el PoP2, el paquete sufriría las siguientes modificaciones:

- En s1: se configura como IP de origen la de s1 y como IP de destino la del s2.
- En s2: se configura como IP de origen la del s2 y como IP de destino la del s3.

Teniendo en cuenta que distintos flujos de datos pueden compartir algún *link* en su *path*, es fácil notar que en algunos casos, los flujos podrían ser indistinguibles luego del cambio en los campos mencionados. Se observa el siguiente caso en la topología de la figura 1.5: un flujo requiere enviar tráfico desde h1 a h3 pasando por

# ONRP	1	2
Red origen	172.16.1.3/32	172.16.1.0/24
Red destino	172.16.3.3/32	172.16.3.0/24
Protocolo	UDP	*
Puerto origen	15194	*
Puerto destino	15193	*
Path	[PoP1,PoP2,PoP3]	[PoP1,PoP2,PoP4,PoP3]

Figura 1.6: Ejemplo de dos ONRPs.

	Origen	$L_{1,2}$	$L_{2,3}$	Destino
IP origen	172.16.1.3	172.16.1.254	172.16.2.254	172.16.1.3
IP destino	172.16.3.3	172.16.2.254	172.16.3.254	172.16.3.3
Protocolo	UDP	UDP	UDP	UDP
Puerto origen	15194	1	1	15194
Puerto destino	15193	1	1	15193

Figura 1.7: Ejemplo de flujo perteneciente a la ONRP1.

el PoP2 y otro flujo requiere enviar paquetes desde h4 hacia h3 pasando también por PoP2. En ambos casos, cuando los paquetes pasan por s2, se configura la IP de este PoP como dirección de origen y la de s3 como dirección de destino, lo que hace que los flujos no puedan ser diferenciados al llegar al PoP3. Se entiende entonces, que no es suficiente modificar las direcciones IP en los paquetes para encaminar el tráfico de forma correcta.

Para solucionar este problema, se plantea utilizar algún campo del encabezado de capa de transporte como forma de identificar distintos flujos. Es relevante tener en cuenta que el protocolo OpenFlow, que se utiliza en el sistema, sólo es capaz de modificar los campos de encabezado de los siguientes protocolos de transporte: UDP, TCP y SCTP. Sin embargo, como estos representan la mayoría del tráfico en Internet, no supone una limitante de gran importancia.

Una ONRP se define de la forma siguiente:

$$ONRP = \{Subnet\_Src, Subnet\_Dst, Protocol, Port\_Src, Port\_Dst, Path\}$$

donde los campos *Port_Src*, *Port_Dst* y *Protocol* pueden no estar definidos, como se puede ver en la ONRP 2 de la Figura 1.6.

Entonces, además de modificar las direcciones IP en cada switch intermedio como se mencionó, se utilizan los campos de puertos de destino y origen para identificar los flujos de tráfico. En cada salto se coloca en el campo *Port_Src* el *local ONRP ID* asociado al link entre ese PoP y el siguiente, como en la Figura 1.7. Puede observarse que en ambos casos el *local ONRP ID* vale 1, lo que se debe a que es la primera ONRP en utilizar esos links. Si se registra una segunda ONRP que utilice algún link de los que ya están en uso, el *local ONRP ID* para ese caso será diferente.

ONAT ID	1
IP origen	172.16.1.3
IP destino	172.16.3.3
Protocolo	UDP
Puerto origen	15194
Puerto destino	15193

Figura 1.8: Tabla ONAT asociada a un flujo de la ONRP1.

Con las modificaciones mencionadas, surge la necesidad de guardar de alguna forma los datos que se encontraban en estos campos cuando el paquete ingresó en el PoP de origen, para poder recuperarlos una vez que llega a su destino. Para esto se introduce la entrada *Overlay Network Address Translation (ONAT)* que se identifica con un ONAT_ID (único dentro de cada ONRP). En el PoP de origen se guardan los siguientes datos: IP origen, IP destino, Protocolo, Puerto L4 origen, Puerto L4 destino, y se asigna un ONAT_ID que se almacena en el campo $Port\_Dst$ del paquete (tal como se ve en la Figura 1.7), y permanece incambiado durante el ruteo. Cuando el paquete llega al PoP de destino, éste conoce el identificador así como el ONRP_ID, y es capaz de recuperar toda la información relevante. Si se vuelve al flujo de la Figura 1.7, la entrada de ONAT asociada es la de la Figura 1.8.

Algoritmo de prioridad

Cuando existe alguna ONRP cuya subred asociada se encuentra incluida en la subred de otra política, es posible utilizar la prioridad de los flujos para distinguirlas. Además, es útil a la hora de realizar búsquedas: cuando se desea crear una nueva ONRP, puede realizarse una comparación con las políticas existentes con esa misma prioridad en lugar de todas las que existan para identificar si la misma ya fue creada con anterioridad.

Con este fin, se registra la prioridad del tráfico, que es calculada con el siguiente algoritmo:

$$ONRP_{Priority} = (X_1 + X_2)MaxNetMaskLenght$$
$$+(X_3 + 4X_4 + 2X_5)MaxNetMaskLenght^2$$

donde:
X_1: largo de la máscara de la subred de origen
X_2: largo de la máscara de la subred de destino
X_3: 1 si el protocolo de capa de transporte está especificado, 0 en caso contrario
X_4: 1 si el puerto de origen está especificado, 0 en caso contrario
X_5: 1 si el puerto de destino está especificado, 0 en caso contrario
MaxNetMaskLenght = 32

1.3.2. MeasureApp

Introducido también como parte de ONRApp, este bloque es encargado de realizar mediciones de parámetros que determinen la calidad de servicio de la red (en particular RTT). Su funcionamiento consiste en recibir e interpretar solicitudes de medición enviadas desde otro bloque, definir el camino que los paquetes de medida deben seguir desde el origen hacia el destino solicitado, por el camino deseado, y luego calcular el valor de RTT antes de entregar el resultado de la medida.

Además del origen, destino, y camino a seguir, la solicitud de medida entrega los siguientes parámetros: *average*, *throughput* y *packet_size*. El primero de ellos, indica la cantidad de veces que debe enviarse un paquete (ida y vuelta) para luego realizar el promedio entre todas las medidas conseguidas, y así calcular el tiempo que se anunciará. El segundo, es la cantidad de paquetes que deben enviarse en "ráfaga" a la hora de realizar la medida. El tercero es un entero positivo que indica el tamaño de la carga útil del protocolo UDP a ser enviado en un mensaje medido en bytes.

Existen dos motivos principales por los que la medición de QoS de un *path* se realiza inyectando paquetes UDP:

- La forma de que los paquetes de medida transiten un *path* específico a través de la red es utilizando el algoritmo de encaminamiento diseñado que trata únicamente el tráfico generado con los protocolos TCP y UDP.
- TCP es un protocolo confiable y orientado a conexión, lo que implica una sobrecarga de tiempo en el establecimiento y finalización de la conexión.

Cuando se recibe una solicitud, deben crearse dos políticas de ruteo para indicar por dónde rutear los paquetes en dirección al destino, y luego de vuelta al origen. Además, el bloque revisa si todos los *links* que componen los caminos elegidos están registrados y operativos, y actualiza el estado de la medida.

1.3.3. Sistema de decisión en base a medidas

Idealmente, para realizar las decisiones sobre ruteo se querría saber el estado de todas las rutas, en cada momento de decisión. Esto implicaría realizar mediciones constantes, inundando la red de paquetes innecesarios que pueden generar congestión e interferir con las propias medidas. Por este motivo, es necesario tomar las decisiones de alguna forma que no requiera medir todas las veces, comprometiendo lo menos posible la integridad de las decisiones. Existen diversos algoritmos que pueden ser utilizados con la finalidad de tomar este tipo de decisiones. En particular, se hará foco en uno presentado en [26], que se basa en el concepto de aproximación por horizonte errante. Esto implica planificar un horizonte de tiempo

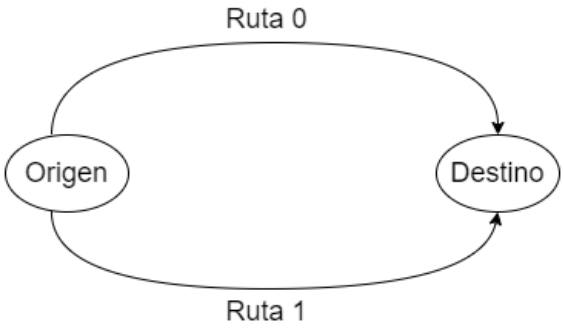


Figura 1.9: Escenario sencillo de dos rutas posibles.

futuro y tomar decisiones de control óptimas para ese horizonte, pero solo implementar la primera parte de ese plan. A medida que el tiempo avanza, el horizonte de tiempo planificado se "recorre", y se repite el proceso de planificación y control con nueva información disponible, lo que permite adaptar continuamente las decisiones.

En primer lugar, se define el problema planteado como un proceso de decisión markoviano (*Markov Decision Problems*, MDP). Los componentes que definen a un MDP son:

- Épocas de decisión, t
- Conjunto de estados, S
- Conjunto de acciones, A
- Probabilidades de transición, P
- Retornos, R

En cada intervalo de tiempo determinado, se debe decidir qué acción se realiza mediante una política que tiene en cuenta el estado en el que se encuentra el sistema, buscando maximizar un retorno. Formulando el problema de esta forma, cada ruta i cuenta con L^i niveles de retardo posibles, matriz de transición P^i, costo de medida c^i, y la cantidad de épocas de decisión transcurridas desde que se realizó la observación τ^i. Las posibles acciones a tomar son medir la ruta ($a = 1$) o no medir ($a = 0$).

En la Figura 1.9, se muestra un escenario sencillo con dos rutas. Cada ruta tiene dos estados posibles (dos niveles de retardo) como se muestra en la Figura 1.10. A continuación se brindan los parámetros que caracterizan a las rutas.

- Ruta 0:

$$P_0 = \begin{bmatrix} 0{,}7 & 0{,}3 \\ 0{,}3 & 0{,}7 \end{bmatrix}$$
$$l_0 = 0{,}5 \ , \ l_1 = 2$$
$$c_0 = 0{,}05$$

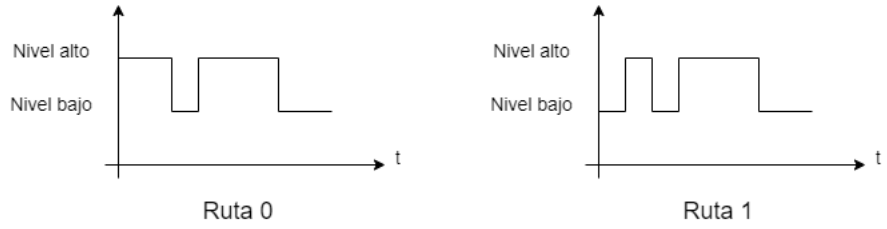


Figura 1.10: Dos posibles rutas con dos niveles de retardo cada una (bajo y alto).

- Ruta 1:

$$P_1 = \begin{bmatrix} 0{,}9 & 0{,}1 \\ 0{,}1 & 0{,}9 \end{bmatrix}$$
$$l_0 = 1 \ , \ l_1 = 3$$
$$c_1 = 0{,}15$$

En este caso en particular, se cuenta con dos rutas posibles entre un origen y un destino, con dos niveles de RTT posibles cada una. Se considera que la mejor ruta en cada instante, es la que tiene el menor RTT, pero el sistema no puede observarse (medir ambas rutas) en todos los tiempos de decisión, pues esto tiene un costo asociado. Por lo tanto, en cada uno de estos tiempos, el algoritmo decide qué rutas se miden (una, ambas o ninguna) teniendo en cuenta parámetros como el tiempo desde la última medida, y el estado en el que se encuentran las rutas, y las probabilidades de cambio de las mismas. Cuando se decide no medir una ruta, su estado siguiente se define a partir de la probabilidad de que esté en cada nivel, es decir, se estima. Es importante tener en cuenta que con cada iteración en la que se estima el nivel de la ruta sin medirla, la incertidumbre sobre su estado crece, aumentando la probabilidad de realizar una predicción errónea.

Según las pruebas realizadas en el trabajo en cuestión, que se realizaron tanto con trázas de tráfico sintéticas como reales, se encuentra que el algoritmo tiene un muy buen desempeño, realizando pocas medidas y eligiendo con gran frecuencia (94 % - 99 %) la ruta deseada.

1.4. Objetivos del proyecto

Los objetivos planteados para este proyecto son los siguientes:

- Desarrollar una aplicación en base a ONRApp [21] y lo expuesto en la tesis [26], que utilice OpenDaylight como controlador de la red SDN.
- Diseñar en Mininet una red y ejecutar en ella las pruebas de validación de la aplicación desarrollada.
- Diseñar una maqueta de laboratorio que permita realizar pruebas operativas de la aplicación desarrollada.

1.5. Estructura del documento

El resto de la documentación se estructura de la siguiente forma:

En el capítulo 2 se presentan las herramientas utilizadas como lo son el controlador OpenDaylight y el protocolo OpenFlow.

El capítulo 3 describe el diseño del sistema, desde la arquitectura utilizada hasta el funcionamiento de los distintos módulos que conforman la aplicación, y criterios establecidos para su implementación.

En el capítulo 4 se muestra el diseño de los ambientes de pruebas, tanto en entorno simulado como en laboratorio.

El capítulo 5 exhibe los resultados de las pruebas realizadas a la aplicación desarrollada.

Finalmente, en el capítulo 6 se presentan las conclusiones del proyecto y se plantean los posibles trabajos a futuro.

En este capítulo se brindó una introducción al trabajo realizado, presentando los motivos que impulsaron su desarrollo y los principales objetivos planteados. Se realizó una descripción de trabajos previos que sirven de base para la ejecución de este proyecto, junto con una breve explicación de los diseños allí planteados. Por un análisis más detallado referirse a los trabajos [21] y [26]. Habiendo explicado la motivación y el contexto, en el siguiente capítulo se presentarán las principales herramientas utilizadas para llevar a cabo este trabajo.

Capítulo 2

Herramientas

En el presente capítulo, se realiza una descripción de las herramientas utilizadas durante el desarrollo de este proyecto. El objetivo es presentar el controlador OpenDaylight (ODL), y brindar un resumen de OpenFlow, el protocolo de comunicación con mayor adopción dentro del paradigma SDN para la programación del plano de datos.

En la sección 2.1, se realiza una introducción al controlador OpenDaylight, analizando su arquitectura y tecnologías subyacentes. Finalmente, en la sección 2.2 se presenta el protocolo OpenFlow y se resumen las principales características de sus distintas versiones, desde la 1.0 hasta la 1.5.

2.1. Controlador OpenDaylight

OpenDaylight es un controlador SDN de código abierto que surge en 2013 como una colaboración entre Cisco e IBM. En la tabla 2.1 se listan las versiones disponibles hasta la fecha junto con el año de lanzamiento. ODL presenta su primera versión funcional *Hydrogen* a principios de 2014, y cuenta con una comunidad activa que ha lanzado su decimoséptima versión *Chlorine* en Octubre de 2022.

2.1.1. Arquitectura del controlador

El controlador ODL está basado en Java y cuenta con una arquitectura modular y extensible que permite la integración con diversas tecnologías y proveedores de equipos de red. En la Figura 2.1, se presenta un diagrama de la arquitectura que muestra las principales capas y componentes del controlador.

Una característica que lo hace destacar sobre otros controladores es la disponibilidad de numerosas interfaces southbound, siendo compatible con múltiples protocolos, como OpenFlow, NETCONF y BGP/PCEP. Además, cuenta con una capa de abstracción, conocida como MD-SAL (Model-Driven Service Abstraction Layer), que se sitúa sobre dichas interfaces y oculta los detalles de la infraestructura de red subyacente.

Tabla 2.1: Versiones de OpenDaylight hasta la fecha. Para el desarrollo de este trabajo se utilizó la versión Oxygen del controlador.

Nombre de versión	**Fecha de lanzamiento**
Chlorine	Octubre 2022
Sulfur	Mayo 2022
Phosphorus	Setiembre 2021
Silicon	Marzo 2021
Aluminium	Setiembre 2020
Magnesium	Marzo 2020
Sodium	Setiembre 2019
Neon	Marzo 2019
Fluorine	Agosto 2018
Oxygen	Marzo 2018
Nitrogen	Setiembre 2017
Carbon	Junio 2017
Boron	Noviembre 2016
Beryllium	Febrero 2016
Lithium	Junio 2015
Helium	Octubre 2014
Hydrogen	Febrero 2014

Model-Driven Service Abstraction Layer

Es un componente fundamental en la arquitectura del controlador y se basa en el concepto de modelado de datos utilizando el lenguaje YANG (Yet Another Next Generation). Esta capa proporciona una visión abstracta de la información de la red en forma de modelos de datos YANG. Esto facilita la interoperabilidad y la programación de aplicaciones que se basan en la representación lógica de la red.

A su vez, MD-SAL actúa como una interfaz para las aplicaciones y servicios que desean interactuar con la red. Proporciona un conjunto de APIs que permiten a las aplicaciones enviar solicitudes, recibir notificaciones y acceder a datos de configuración y estado de la red.

En conclusión, MD-SAL proporciona a las aplicaciones una manera coherente y unificada de interactuar con la red, lo que facilita el desarrollo de aplicaciones SDN en el entorno de OpenDaylight.

Tecnologías subyacentes

En este apartado, se presenta una breve descripción de las tecnologías en las que se sustenta el proyecto OpenDaylight [5].

- **OSGi:**
  Open Services Gateway initiative (OSGi) es el back-end de OpenDaylight y proporciona un marco de trabajo para la creación de módulos independien-

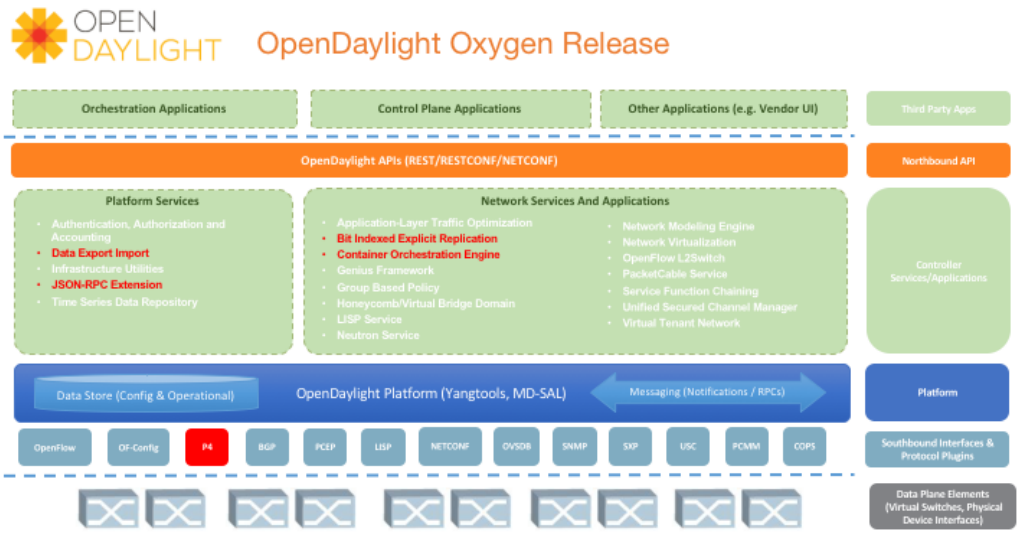


Figura 2.1: Arquitectura del controlador OpenDaylight. [8]

tes que pueden ser instalados, actualizados o desinstalados dinámicamente en tiempo de ejecución. OSGi facilita la modularidad que caracteriza al controlador donde cada módulo o *bundle* representa una función o servicio específico a utilizar para satisfacer las necesidades del entorno particular.

- **Apache Karaf:**
  OpenDaylight utiliza Apache Karaf como el contenedor principal en el que se ejecuta el controlador. Karaf proporciona una plataforma que simplifica el despliegue y ejecución de módulos basados en OSGi. Introduce el concepto de *feature* que refiere a la agrupación de diferentes *bundles* para implementar una funcionalidad específica. Las *features* se pueden agregar o quitar, y combinar para construir una aplicación personalizada que incluya sólo las funcionalidades necesarias. En el Apéndice C se presenta un análisis de las *features* disponibles en el controlador y la forma de instalación.

- YANG:
  Como se mencionó anteriormente, el controlador se basa en modelos de datos para describir y gestionar la información de la red. Con YANG se pueden modelar diferentes aspectos de la red, incluyendo tanto datos de configuración como datos de estado de los elementos de la red, notificaciones y RPCs (Remote Procedure Calls).

2.1.2. OpenFlow Plugin

OpenDaylight brinda un complemento llamado OpenFlow Plugin, desarrollado con el objetivo de admitir OpenFlow 1.0 y 1.3.x, y permitir ampliarlo a medida que OpenFlow evoluciona [11].

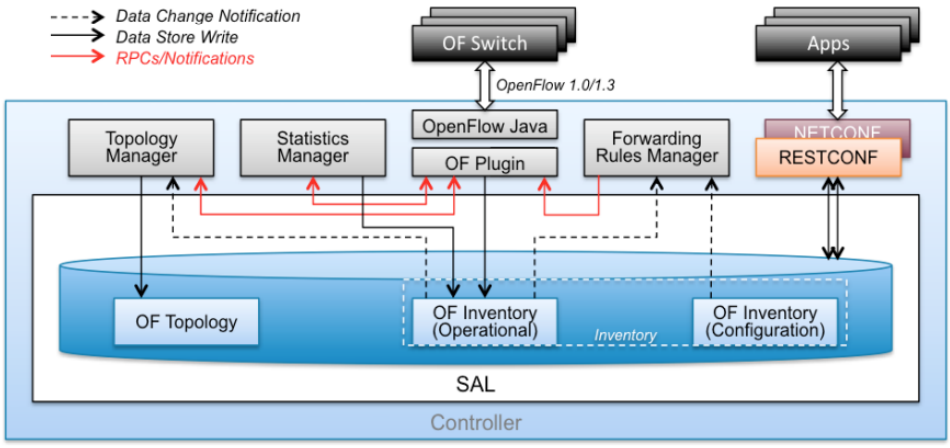


Figura 2.2: Diseño del proyecto OpenFlow Plugin basado en la arquitectura MD-SAL. [11]

En la Figura 2.2 se presenta su arquitectura, que está compuesta por los siguientes elementos:

- **OpenFlow Java:** es una librería que implementa los codec de OpenFlow, el mismo traduce los mensajes OpenFlow en su representación interna y viceversa.
- **OpenFlowPlugin:** gestiona las sesiones de los switch OpenFlow, proporciona una API de servicio OpenFlow de bajo nivel que permite: agregar-modificar-flujo, eliminar-flujo, etc.
- **Statistics Manager:** es el responsable de recopilar estadísticas y estado de los switch Openflow y almacenar esta información en la base de datos operacional, para uso de las aplicaciones.
- **Topology Manager:** es el encargado de descubrir la topología Openflow mediante LLDP (Link Layer Discovery Protocol) y almacenar esta información en la base de datos operacional, para uso de las aplicaciones.
- **Forwarding Rules Manager:** Es el módulo "top level" de OpenFlow que expone las funcionalidades OpenFlow a las aplicaciones del controlador y proporciona la API de nivel de aplicación. Es el principal elemento encargado de gestionar el inventario de los switch OpenFlow y la configuración (programación) de flujos en los switches. También concilia la configuración del usuario con el estado de la red descubierto por el plugin OpenFlow.

Una de las funcionalidades principales de este proyecto es que permite realizar modificaciones de las tablas de flujo de los switches OpenFlow mediante APIs brindadas por el controlador. Las interfaces RESTCONF proporcionadas por el

controlador permiten programar los dispositivos, lo cual es esencial para el desarrollo del trabajo planteado. Las solicitudes que se hacen al controlador mediante estas interfaces, se pueden construir en formato JSON o XML [12].

2.2. OpenFlow

Aplicado a SDN, OpenFlow define el protocolo de comunicación entre los planos de datos y de control, y parte del comportamiento del primero de ellos, pero no determina el funcionamiento del controlador.

OpenFlow es un protocolo de comunicación introducido para posibilitar la separación del plano de datos y el plano de control en las redes de computadoras, facilitando y flexibilizando su gestión y programación. La utilización del mismo permite también la integración de los dispositivos (routers, switches) con un controlador externo centralizado que se encarga de tomar las decisiones de ruteo de la red, teniendo en cuenta el conocimiento completo de la topología, y pudiendo realizar cambios en los caminos de forma dinámica.

El protocolo utiliza tablas de flujo en los dispositivos de la red que son creadas mediante reglas programables que indican cómo deben manejarse los distintos tipos de tráfico que pueden recibirse. Es decir, cuando el router o switch recibe un paquete, consulta esta tabla para decidir qué debe hacer. Esto proporciona a la red la posibilidad de adaptarse a distintas situaciones de tráfico, tales como congestión, distintas calidades de servicio requeridas, entre otras.

2.2.1. Switch OpenFlow

Se define la estructura de un dispositivo de red que sea capaz de implementar el protocolo en cuestión, llamado *Switch OpenFlow* [20]. Como se espera de un switch, éste se encarga de distribuir por el puerto correspondiente los paquetes que le llegan, realizando las modificaciones necesarias para asegurarse de que lleguen a su destino.

Cuenta además con una función que se encarga de realizar un *packet matching* en cada mensaje recibido, que le permite al switch decidir qué acción tomar, existiendo las siguientes posibilidades: el paquete recibido se reenvía por un puerto local (modificando los encabezados necesarios para el correcto ruteo del mismo), el paquete se descarta, o se envía al controlador. En este último caso, se hace utilizando un canal seguro que funciona en los dos sentidos: del controlador al switch, y del switch al controlador, y sirve tanto para datos como para mensajes de control.

Si el controlador necesita enviar un paquete de datos al switch, lo hace utilizando un mensaje de *"PACKET_OUT"*, mientras que los paquetes de datos que van del switch al controlador para que éste dé instrucciones sobre su manejo, se mandan con un mensaje de *"PACKET_IN"*.

2.2.2. Versiones de OpenFlow

OpenFlow 1.0

Esta fue la primera versión oficial de OpenFlow, presentada en 2009. Estableció los conceptos básicos de la separación entre el plano de control y el plano de datos en las redes. Introdujo las tablas de flujo, las reglas de coincidencia y las acciones que permiten programar el comportamiento de los dispositivos de red.

- **Puerto OpenFlow**
  Un puerto OpenFlow se asocia a un puerto físico del switch. A cada puerto se le pueden asociar diferentes colas en las que colocar los paquetes, dependiendo de la calidad de servicio (QoS) que se le quiera brindar al flujo. En el caso en que un paquete recibido por el switch deba ser reenviado por un puerto local, la acción que especifica este puerto también puede indicar en qué cola debe situarse.

- **Tablas de flujo**
  Las tablas de flujo estan formadas por distinas *flow entries*, que contienen los siguientes campos: encabezados, contadores, y acciones. El primero de ellos se utiliza para buscar coincidencias con los paquetes recibidos, y determinar si éste pertenece a un determinado flujo definido, para así realizar las acciones indicadas en el campo correspondiente. El contador se utiliza para almacenar estadísticas sobre cada tipo de flujo. En el caso en que no se encuentren coincidencias con ninguna entrada de la tabla, el paquete se envía al controlador, quien lo procesa e indica qué acciones deben realizarse.

- **Packet Matching**
  Cuando el switch recibe paquetes, ya sea por un puerto de entrada o desde el controlador, los compara con sus tablas de flujo para identificar coincidencias. Los campos en los que pueden buscarse las coincidencias son los siguientes:

  - Puerto de entrada
  - ID de la VLAN
  - Prioridad de la VLAN
  - Dirección ethernet de origen
  - Dirección ethernet de destino
  - Tipo de trama ethernet
  - Dirección IP de origen
  - Dirección IP de destino
  - Protocolo IP
  - Bits de Tipo de Servicio de IP
  - Puerto de origen TCP/UDP

- Puerto de destino TCP/UDP

Algunos de estos campos pueden no estar definidos. Los mismos se procesan en orden, y una vez que se encuentra una coincidencia con un campo, la búsqueda se detiene.

- **Acciones y envío de paquetes**
  Cuando se encuentra una coincidencia de un paquete con un flow, se debe bien reenviarlo o descartarlo. Para el primer caso, generalmente se especifica por qué puerto debe enviarse. Existen 5 puertos virtuales predefinidos en esta versión del protocolo, y son los siguientes:

  - LOCAL: reservado para las comunicaciones entre el controlador y el dispositivo de red.
  - ALL: se utiliza para enviar un flujo de paquetes por todos los puertos del switch, excepto el puerto de entrada.
  - CONTROLLER: usado para enviar paquetes al controlador.
  - IN_PORT: es utilizado cuando se desea reenviar un paquete por el mismo puerto por el que ingresó.
  - TABLE: reservado para paquetes desde el controlador hacia el switch.

- **Mensajes entre el switch y el controlador**
  La transmisión en ambos sentidos se hace a través de un canal seguro preestablecido utilizando *Transport Layer Security (TLS)* sobre TCP. Este canal puede funcionar en banda o fuera de banda, pero el caso más común es el primero, en el que las tablas se construyen de forma que estos mensajes se envíen al puerto virtual LOCAL. Los mensajes comienzan con un encabezado OpenFlow, que especifica la versión del protocolo, el tipo y largo del mensaje, y el ID.

  Estos mensajes pueden agruparse en tres grandes categorías, presentadas a continuación.

  - Simétricos: pueden enviarse tanto por el controlador como por el switch, sin haber sido solicitados por la otra parte. Por ejemplo, los mensajes de tipo HELLO, que son utilizados para establecer la conexión y definir la versión más reciente del protocolo que ambas partes soportan.
  - Asíncronos: se envían del switch al controlador sin que éste los solicite. Por ejemplo, los mensajes de PACKET_IN que el switch utiliza para enviar paquetes de datos al controlador, o el tráfico del plano de control.
  - Controlador-switch: es la categoría más amplia, y consta de mensajes enviados por el controlador con el fin de configurar el switch y administrar las tablas de flujos. Un ejemplo de esto son los mensajes de PACKET_OUT.

OpenFlow 1.1

Lanzada en 2011, esta versión trajo mejoras en la especificación y en la calidad de servicio. También introdujo la capacidad de agrupar múltiples acciones en una sola instrucción, y la existencia de múltiples tablas de flujo, que se identifican por un ID único.

- **Tablas de flujo múltiples**
  Esta nueva característica es la que provoca el mayor cambio en esta versión. Trae una mayor flexibilidad, pues es posible encadenar las entradas de la tabla mediante una instrucción en una *flow entry* que apunta a otra tabla de flujo. Esta instrucción es llamada *GOTO*. Cuando es ejecutada, se lleva a cabo nuevamente un *packet matching* comenzando por la primer entrada de la nueva tabla de flujo. Esto hace que se incremente tanto la complejidad en la lógica del *packet matching*, como la cantidad de modificaciones que el paquete puede tener mientras pasa por el switch. Al conjunto ordenado de las tablas por las que pasa un paquete se lo denomina *pipeline*.

- **Instrucciones**
  Las diferentes instrucciones de las tablas que componen un *pipeline* pueden ser ejecutadas de dos formas distintas. En la primera de ellas, las acciones de la entrada correspondiente de cada tabla son agregadas a un *action set* que se ejecuta al final del *pipeline* en un orden específico determinado por el protocolo. La otra opción, es que las acciones se ejecuten inmediatamente después de pasar por la tabla en la que se describen, lo que se logra con la instrucción *apply action*.

OpenFlow 1.2

Esta versión agregó soporte para túneles virtuales, mejoró la capacidad de expresión de las reglas de flujo y permitió una mayor separación entre los dominios de flujo y los controladores.

- **Extensión de PACKET_IN**
  En versiones anteriores del protocolo, el mensaje incluía los campos del encabezado que habían resultado en que el paquete se enviara al controlador. En esta versión, se incluyen otros campos, como el puerto virutal de entrada, el puerto físico de entrada, y el campo *metadata*, que se construye durante el procesamiento del paquete en el *pipeline*.

- **Múltiples controladores**
  En esta nueva versión de OpenFlow, el switch puede ser configurado para mantener conexiones simultaneas con múltiples controladores. Esto soluciona un problema planteado en versiones anteriores: si existía una pérdida de conexión con el controlador, el sistema quedaba funcionando sin un control centralizado. Con la nueva característica, en caso de pérdida de conexión con el controlador que está en uso, se puede seguir trabajando con uno de respaldo.

Existen tres roles que el controlador puede asumir en relación a un switch: *master*, *slave* o *equal*, y determinan el grado hasta el cuál el controlador puede cambiar la configuración del switch.

OpenFlow 1.3

Se introdujeron mejoras significativas en la escalabilidad y el rendimiento. Muchas implementaciones se basan en esta versión, con el fin de estabilizar los controladores. Como fue un gran salto desde la versión anterior, pasó tiempo suficiente sin que se lanzara una nueva, permitiéndole a los diseñadores de hardware desarrollar soporte para las nuevas características presentadas.

- **Soporte más flexible para *table-miss***
  Hasta el momento, en caso de que un paquete no encontrara ninguna coincidencia en una tabla de flujo, había tres opciones: descartarlo, enviarlo al controlador, o a la próxima tabla. Desde esta versión, es posible configurar una entrada de *table-miss* en la tabla que coincida con todos los paquetes, con la prioridad más baja para asegurar que si hay una coincidencia, se encuentre antes de llegar a este punto. Con esta funcionalidad, además de las opciones anteriores, se puede elegir aplicarle instrucciones a los paquetes que no coinciden con otra entrada, lo que permite una lógica de *packet-matching* más sofisticada.

- ***Cookies* en PACKET_IN**
  Con el creciemiento comercial del protocolo, se hizo muy importante mejorar el desempeño y disminuir el costo computacional de utilizarlo. Para contribuir a esto, se agregó el campo *cookies* en el encabezado de los mensajes PACKET_IN. Cuando estos mensajes llegan al controlador, el mismo toma este campo y busca una coincidencia en la memoria *cache*. Si la encuentra, utiliza las instrucciones indicadas en la entrada de flujo correspondiente, y si no, procesa el paquete y agrega la referencia a la memoria para utilizarla para futuros paquetes.

OpenFlow 1.4

En este lanzamiento del protocolo, se agregaron características interesantes como soporte de puertos ópticos, en los que es posible configurar y monitorear la frecuencia de recepción y transmisión. En adición a esto, se agregaron razones más descriptivas para el envío de los mensajes de tipo PACKET_IN, y la posibilidad de monitoreo de flujo en casos de múltiples controladores. De esta forma, un controlador que ha sido configurado como respaldo, puede controlar en tiempo real los cambios realizados en un switch por el controlador principal.

OpenFlow 1.5 [13]

Presentada en 2014, esta versión presentó mejoras en la expresión de las reglas de coincidencia, el control de calidad del servicio y la seguridad. Se mejoró la

expresión de coincidencias mediante el uso de "campos acumulativos" y se añadió soporte para grupos de flujo.

- **Egress tables**
  Hasta el momento, todo el procesamiento se hacía en base al puerto de entrada. Con la implementación de estas tablas, se permite procesar paquetes en el contexto de su puerto de salida asignado. Es decir, cuando un paquete es enviado hacia el puerto por el que debe enviarse, comienza a ser procesado por la primer *egress table*, que puede tener otras encadenadas a continuación.

En este capítulo, se describieron las principales herramientas utilizadas que son el controlador OpenDaylight y el protocolo OpenFlow. Para el desarrollo de este proyecto, se utilizó la versión Oxygen del controlador y switches OpenFlow con versión 1.0 y 1.3. En el siguiente capítulo se proporciona un análisis en profundidad de la arquitectura y el funcionamiento del sistema diseñado.

Capítulo 3

Diseño del sistema

En el presente capítulo se aborda el sistema diseñado en este proyecto. Se muestra la arquitectura diseñada y los módulos desarrollados para cumplir los objetivos de la aplicación. Además, se detalla el funcionamiento e implementación de la metodología de medición y configuración de políticas de ruteo, así como el algoritmo de ingeniería de tráfico.

3.1. Arquitectura

Se define la Arquitectura del Proyecto como el conjunto de componentes físicos o virtuales necesario para el correcto funcionamiento de la aplicación. En la Figura 3.1 se muestra un diagrama esquemático de la misma.

3.1.1. Controlador OpenDaylight

El controlador OpenDaylight es el encargado de administrar los switches OpenFlow, además en su base de datos contiene datos fundamentales para el funcionamiento de la aplicación por lo que va a ser consultada reiteradamente por la misma haciendo uso de la REST API que posee (se explicará detalladamente su uso en los próximos capítulos). Para permitir la comunicación entre los switches OpenFlow y la Aplicación a través de Internet es necesario que el servidor donde se aloja tenga una dirección IP versión 4 pública.

3.1.2. SmartRoute

SmartRoute es el nombre de la aplicación que se desarrolló en este proyecto. Está escrita en el lenguaje de programación Python y es utilizada por los administradores de red para poder realizar el ruteo de paquetes y mediciones de QoS entre diferentes PoPs, además implementa ingeniería de tráfico. Posee una interfaz en la cual se pueden ingresar los datos requeridos para ejecutar las distintas tareas que es capaz de realizar a causa de la comunicación bidireccional que posee con los PPG y el controlador. En consecuencia se le debe asignar una dirección IP pública.

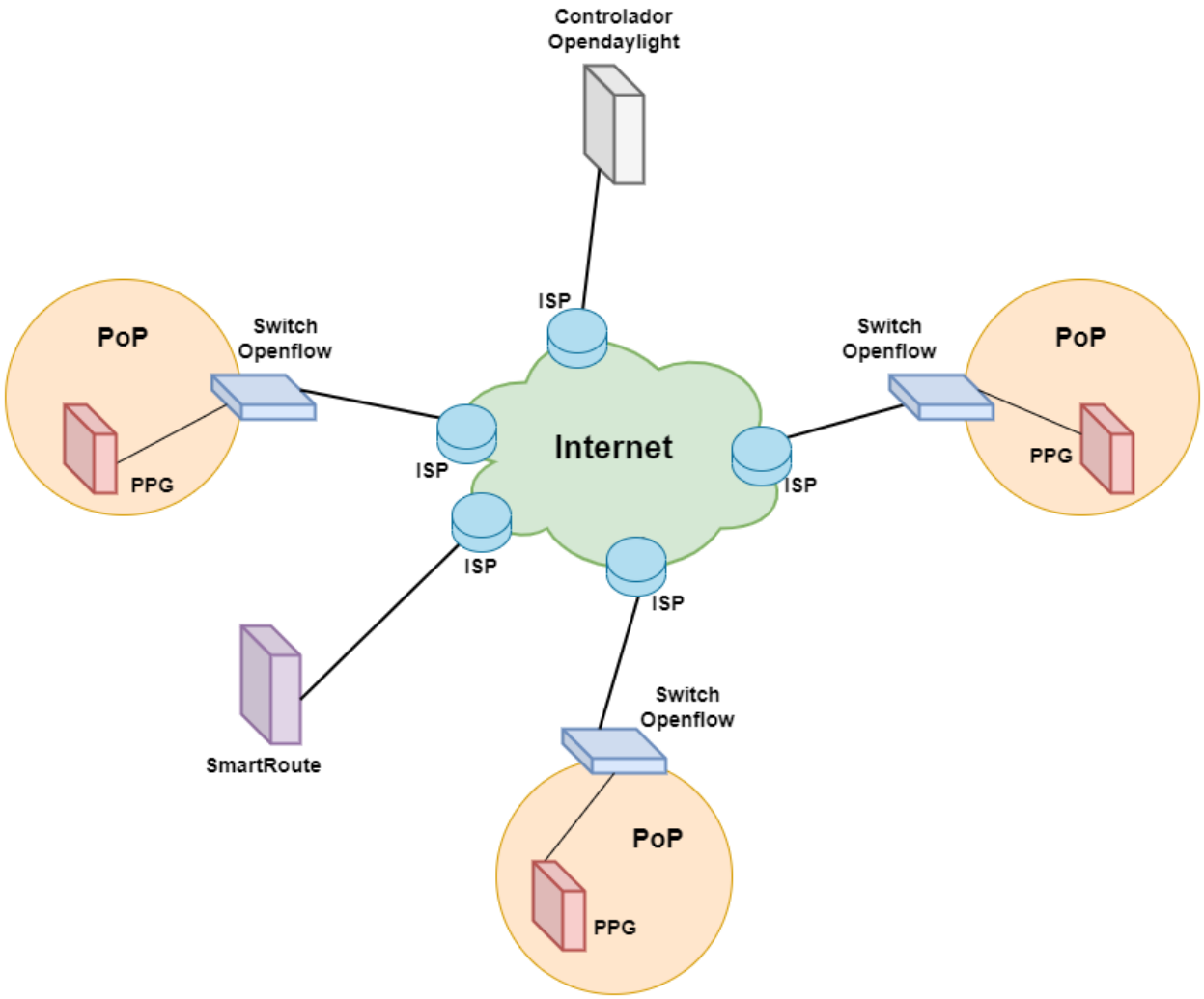


Figura 3.1: Esquema de la Arquitectura.

3.1.3. PoP

El PoP está conformado por un switch OpenFlow, un PPG y los equipos usuarios. Posee dos subnet distintas, una conformada por IP públicas para establecer la conexión con el ISP y otra por direcciones IP privadas para comunicarse con el PPG y los equipos que se encuentren dentro del PoP.

Switch OpenFlow

Es el equipo que se encuentra entre las dos áreas mencionadas, maneja una dirección IP pública para comunicarse con los equipos que se encuentran en Internet. Un ejemplo es la aplicación, el controlador y el resto de los PoPs, fundamentales para el correcto funcionamiento del sistema. Además posee una IP en el rango privado para poder comunicarse con el PPG y los equipos que poseen los usuarios

del PoP. Por medio del switch OpenFlow los equipos que utilizan direcciones IP en el rango privado son capaces de salir a la red de Internet global y comunicarse con el resto de los PoPs, esto será mostrado en los próximos capítulos.

PPG

El PPG se encuentra dentro del PoP y posee direcciones IP en el rango privado. Es el encargado de realizar las medidas que son solicitadas por la aplicación y enviar su resultado a la misma.

Proveedores de Internet (ISP)

Los proveedores de Internet son los que permiten la interconexión entre los PoPs, la aplicación y el controlador. Son los encargados de suministrar las direcciones IP públicas necesarias para el correcto funcionamiento del sistema.

3.2. SmartRoute

En esta subsección se presentarán los componentes principales de la aplicación y las interfaces de comunicación con el entorno. Se comenzará mostrando los medios de comunicación que se emplean y luego se indicarán los componentes principales.

3.2.1. Medios de comunicación

Las tres entidades externas que interactúan directamente con la aplicación son el controlador OpenDaylight, el usuario y los PPG. Como se puede observar en la Figura 3.2 el usuario es capaz de comunicarse con la aplicación por medio de la interfaz de línea de comandos (CLI). Por otro lado, la aplicación se comunica con el controlador OpenDaylight utilizando la REST API que el mismo proporciona, mientras que para la interacción con los PPG se emplean sockets UDP. En las secciones posteriores se mostrará detalladamente la implementación de cada una de las interfaces de comunicación mencionada.

3.2.2. Componentes Principales

Los componentes principales de la aplicación se emplean en capas teniendo en cuenta el uso de cada uno de ellos y el nivel de abstracción, colocando en la parte superior la capa que interactúa con el cliente y en la inferior la que se comunica con el resto de los equipos. En la Figura 3.3 se puede apreciar lo mencionado.

Capa de interacción

Esta capa es la que se encuentra a más alto nivel y es con la que el usuario interactúa con la aplicación. Para lograr la comunicación con el usuario se utiliza la CLI por la cual se envían mensajes y en los casos que corresponda se espera la

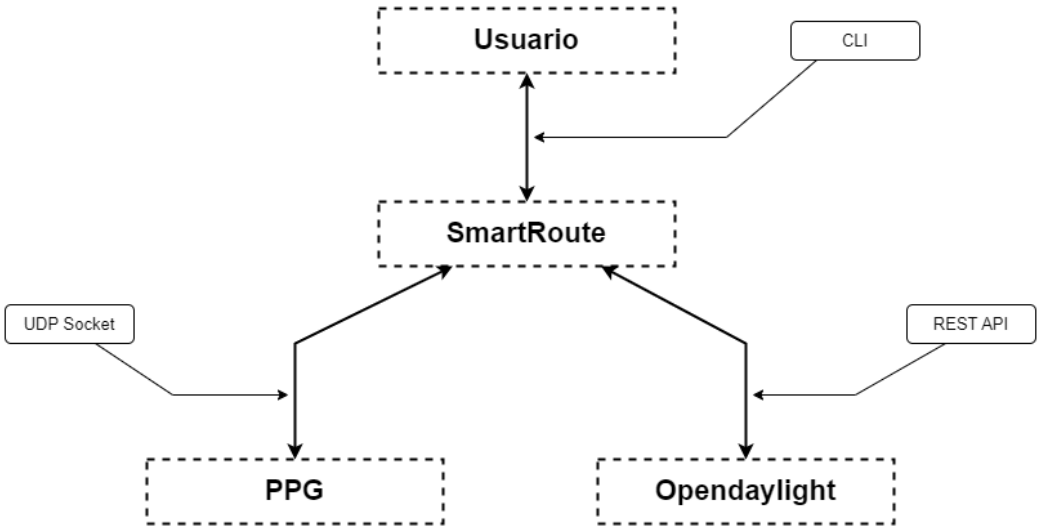


Figura 3.2: Interfaces de comunicación.

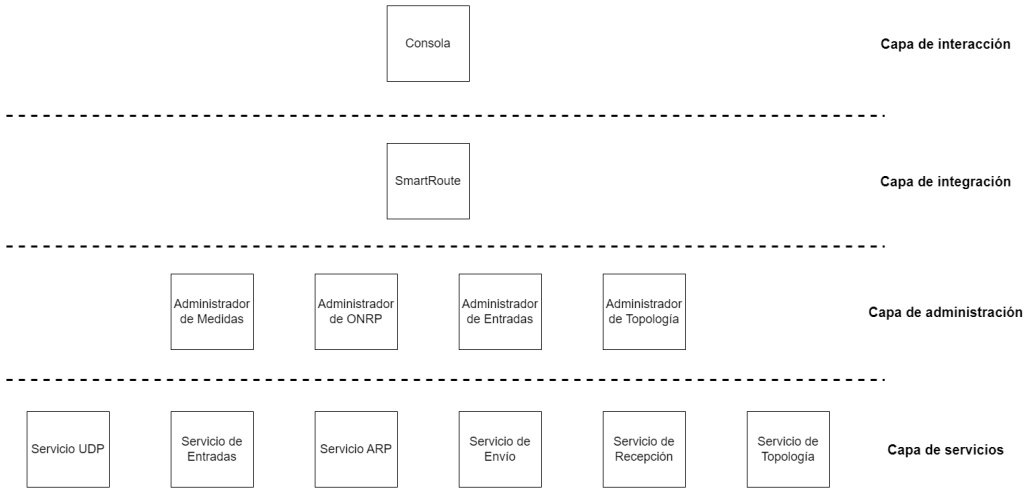


Figura 3.3: Diagrama de capas de SmartRoute.

respuesta del usuario. En la Figura 3.4 se puede observar la pantalla principal de la consola en donde se muestran las opciones que tiene el administrador.

Capa de integración

En esta capa se integran las distintas funcionalidades de la aplicación. Desde la capa superior se reciben las instrucciones ingresadas por el usuario, las cuales son procesadas y ejecutadas mediante funciones que utilizan los recursos presentados por las capas inferiores.

```
Seleccione una opcion del menu:
    1) Nuevo PoP
    2) Nueva ONRP
    3) Nueva medida
    4) Pops
    5) Onrps
    6) Links
    7) Imprimir medidas
    8) Modificar ONRP
    9) Actualizar Arp Table
    10) Activar/Desactivar TeApp
    11) Salir de la App
```

Figura 3.4: Menú CLI.

Capa de administración

La capa de administración es la encargada de gestionar los datos de medidas, políticas de ruteo, topología y entradas de las tablas de los switches OpenFlow. Para ello se poseen cuatro tipos de administradores:

- Administrador de medidas: Es el que administra las medidas solicitadas y las realizadas. Con la capa inferior interactúa con el servicio UDP para poder enviar y recibir medidas, en la misma capa interactúa con el "Administrador de ONRP" para generar las políticas necesarias para realizar las medidas.

- Administrador de ONRP: Este administrador gestiona todas las políticas de ruteo que se encuentran en el sistema. Interactúa con el "Administrador de Topología" para validar la existencia de los enlaces que contiene la ONRP a crear, en caso afirmativo se comunica con el "Administrador de Entradas" para que se generen las entradas necesarias en las tablas de los switches OpenFlow administrados. Asimismo consume de la capa de servicios, el de "Recepción" y el de "Envío" para enviar y recibir paquetes desde los switches OpenFlow. Con el fin de mantener ordenados los datos gestionados, este administrador posee un tipo de dato llamado "ONRPolicy" que contiene los parámetros mostrados en la Figura 3.5.

- Administrador de Entradas: El "Administrador de Entradas" es el encargado de gestionar las entradas de las tablas de los switches OpenFlow (flows) administrados por el sistema. De la capa inferior utiliza el servicio ARP para visualizar las direcciones MAC de los equipos y el servicio de Entradas para crear o eliminar las flows en los switches OpenFlow. En la misma capa se comunica con el "Administrador de ONRP", el cual le solicita la creación y eliminación de flows.

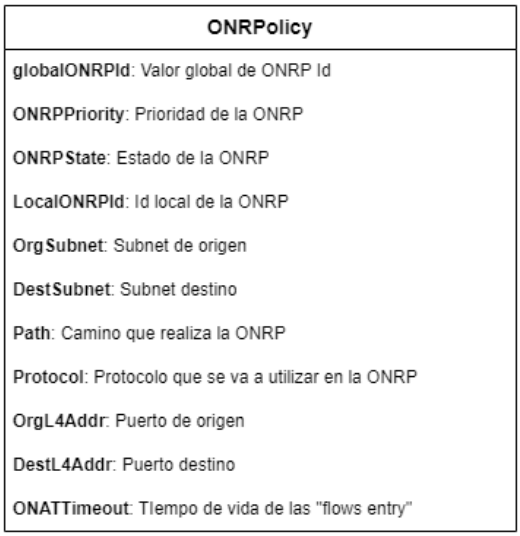


Figura 3.5: Tipo de dato ONRPolicy.

- Administrador de Topología: Es el encargado de administrar toda la topología del sistema. En la misma capa se tiene el "Administrador de ONRP" que se comunica para validar la topología que se tiene actualmente, por lo que es necesario que el administrador en cuestión consuma los servicios de "Topología". Para poder realizar la gestión de forma correcta este administrador gestiona tres tipos de datos que se muestran en la Figura 3.6.

Capa de servicios

La capa de servicios es la que interactúa directamente con los equipos que conforman el sistema y se encarga de suministrarle los datos a las capas superiores.

- Servicio UDP: Permite el envío y recepción de mensajes UDP.
- Servicio de entradas: Se comunica con el controlador para eliminar o crear entradas en las tablas de los switches OpenFlow.
- Servicio ARP: Proporciona los datos de dirección MAC e IP de los equipos que se encuentran en la red administrada.
- Servicio de Envío: Se establece una comunicación con el Controlador que permite enviar paquetes desde los switches OpenFlow.
- Servicio de recepción: Facilita la obtención de los paquetes enviados desde los switches OpenFlow hacia el controlador.
- Servicio de topología: Se genera la comunicación con el controlador para obtener los switches activos que se encuentran en la red.

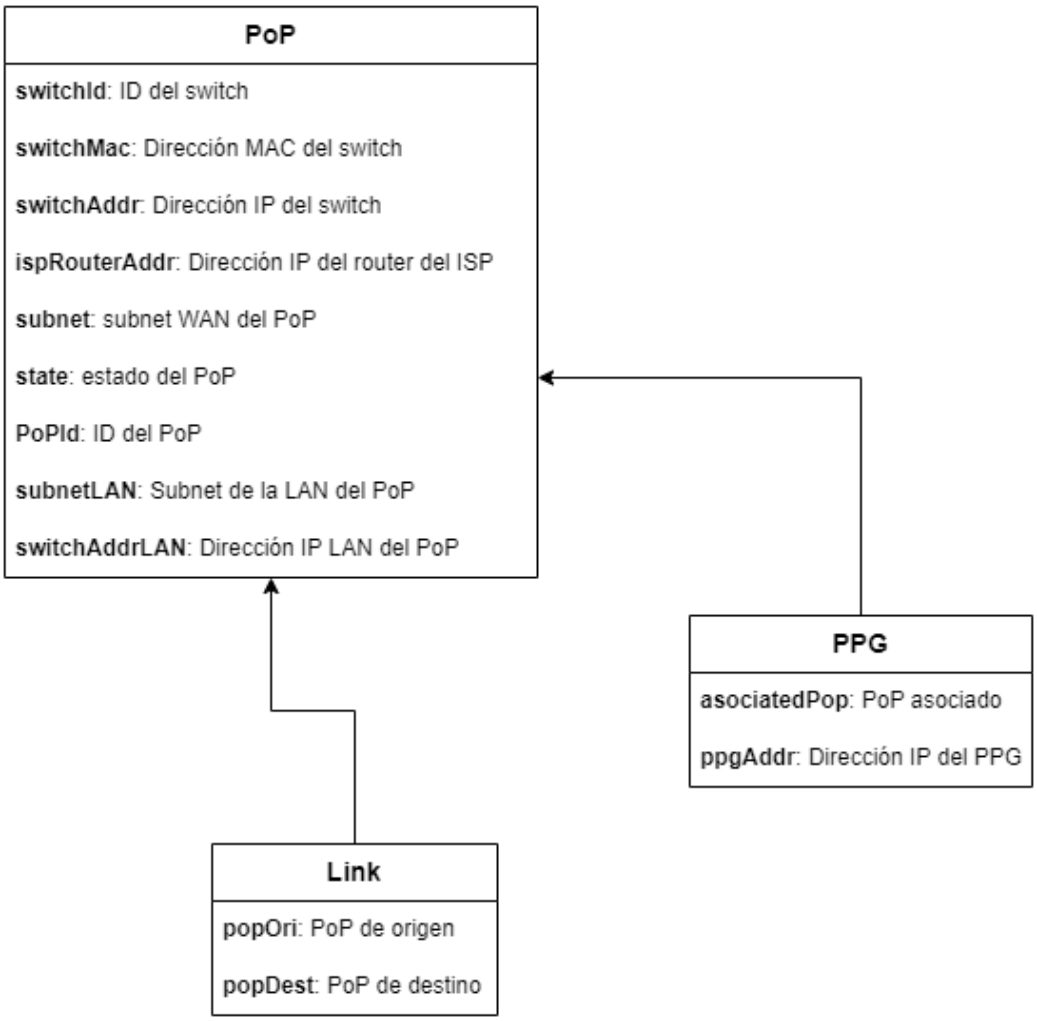


Figura 3.6: Tipo de dato PoP, PPG y Link.

3.3. Diseño de SmartRoute

En este apartado se describe el diseño de las principales aplicaciones presentadas en la sección 1.3 y cómo interactúan entre ellas para lograr la funcionalidad deseada de *SmartRoute*. En la Figura 3.7 se presenta un diagrama que ilustra la organización e interacción entre las aplicaciones.

Utilizan los servicios y administradores descriptos en la sección anterior y permiten separar los objetivos de *SmartRoute*: configuración de políticas de ruteo (*RouteApp*), medición de QoS (*MeasureApp*) y decisión de la estrategia de forwarding en base al monitoreo del estado de la red (*TEApp*). El usuario puede solicitar alguna de estas opciones desde un menú que brinda la aplicación a través de la interfaz de línea de comandos (CLI) mencionada anteriormente. En cada caso, se solicitan los parámetros requeridos que el usuario debe brindar para llevar a cabo dichas acciones.

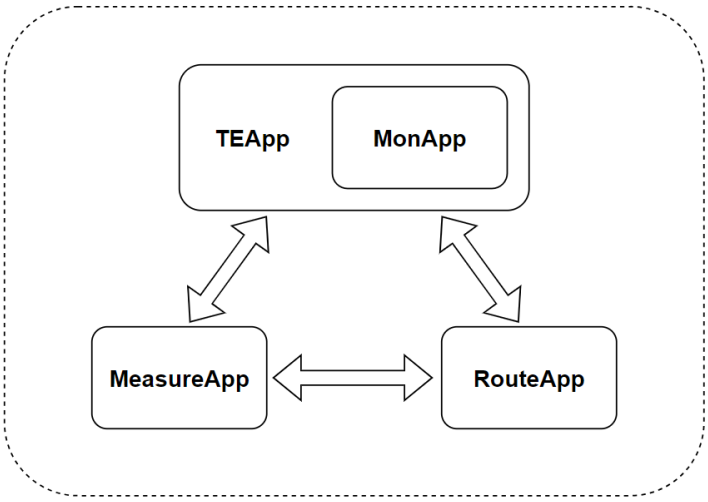


Figura 3.7: Diagrama representativo de la interacción de las aplicaciones.

En particular, puede solicitar la medición de QoS de un camino en la red o la configuración de una ONRP personalizada. Tanto *MeasureApp* como *RouteApp* pueden ser utilizadas por el usuario directamente desde el menú, pero también sirven a *TEApp* para su funcionamiento. A su vez, *MeasureApp* utiliza servicios de *RouteApp* para configurar las políticas de ruteo asociadas a los mensajes de medida del *path* solicitado.

El objetivo de *TEApp* es elegir la ruta de menor retardo entre determinados PoP origen y destino. Esta es la métrica de QoS elegida para trabajar, como se explica más adelante en la sección 3.7. El algoritmo funciona en conjunto con *MonApp* que proporciona información del estado de la red, y determina en cada momento las rutas a medir o estimar en base a mediciones anteriores. A su vez, se comunica con *MeasureApp* para solicitar las mediciones pertinentes y con *RouteApp* para configurar la ruta seleccionada.

3.3.1. Configuración de políticas de ruteo (*RouteApp*)

Al solicitar la creación de una ONRP se debe brindar la siguiente información:

- *SubnetOri*: Subnet de origen.
- *SubnetDest*: Subnet de destino.
- *Protocolo*: Protocolo de capa de transporte. Es un campo opcional, puede quedar sin especificar.
- *PuertoOri*: Puerto origen de capa de transporte. Es un campo opcional, puede quedar sin especificar.
- *PuertoDest*: Puerto destino de capa de transporte. Es un campo opcional, puede quedar sin especificar.

- *Path*: Camino que seguirá el tráfico asociado a la ONRP.

- *ONATTimeout*: Tiempo de vida de la ONRP. Si se especifica un valor mayor a cero, las *flow entries* asociadas a dicha ONRP serán eliminadas una vez se agote el tiempo. En caso de establecerlo en cero la ONRP quedará configurada de forma permanente.

RouteApp se separa en dos etapas principales: la etapa de creación que refiere a la recopilación de los parámetros de la ONRP, seguida de la configuración de los switches para establecer el *path* en la red.

Creación de la ONRP

En esta etapa se ejecuta la función *createONRP* encargada de recopilar y calcular los parámetros que caracterizan la ONRP, y crear el objeto de clase ONRPolicy correspondiente.

En primer lugar consulta al Administrador de Topología si los links del *path* fueron registrados, en cuyo caso se procede a generar el identificador único de la ONRP (*globalONRPId*) y los identificadores locales para los links (*LocalONRPId*) descriptos en la sección 1.3.1. Luego se calcula la prioridad en base al algoritmo presentado en dicha sección.

En este punto, se tienen todos los parámetros relevantes de la ONRP con los que se crea el objeto de clase *ONRPolicy* correspondiente. Finalmente, se inicializa el estado de la ONRP en "*BeingCreated*" y se procede a ejecutar la siguiente etapa.

Configurar el *path* en la red

El objetivo de esta etapa es configurar el plano de datos para que los paquetes asociados a la ONRP sigan el *path* solicitado. Esto lo ejecuta otra función llamada *addPath*.

Si el Administrador de Topología verifica que los links se encuentran activos, se solicita al Administrador de Entradas la creación y configuración de las *flow entries* en los switches involucrados. En primer lugar, procesa y almacena la información de *flow entries* a instalar, según el formato indicado en la sección 3.5.2. Esto lo implementan las funciones *CreateFlows* y *CreateONAT*, donde la última además de generar la *flow entry* de ONAT asigna las *flow tables* de ONAT a ser utilizadas por la ONRP.

Una vez creadas las *flow entries*, el Administrador de Entradas procede con su configuración en los switches de los PoP según corresponda. Si todas son cargadas correctamente, se actualiza el estado de la ONRP en "*Active*" o de lo contrario en "*Down*".

Con esto se da por finalizado el proceso de creación de la ONRP. El usuario puede consultar la información y el estado de las distintas ONRPs configuradas desde el menú de la interfaz.

3.3.2. Medición de QoS (*MeasureApp*)

Cuando un ente externo, sea el usuario u otra aplicación, solicita la medición de una ruta, se genera un flujo de datos a ser identificado y encaminado por *RouteApp*. Esto es con el objetivo de que el mensaje de medida pueda seguir el camino requerido. En la solicitud se debe indicar el *path* a medir, como una lista de los PoPs que lo conforman, una etiqueta que sirve al usuario para identificar la medida, y los parámetros *average*, *throughput* y *packetsize* presentados en la sección 1.3.2.

Las acciones de *MeasureApp* se dividen en tres etapas que se explicarán a continuación. En una primera etapa de inicialización, se busca generar las condiciones para lograr llevar a cabo la medición. Una vez realizado esto, se procede a generar y enviar la solicitud de medida al PPG encargado de iniciar la medición. Por último, se espera la respuesta de la medida para darla por finalizada y registrar el resultado obtenido.

Inicialización de medida

Al comienzo de esta etapa se procede a generar el identificador de la medida que facilita a la aplicación el procesamiento e identificación de las distintas medidas en ejecución. A su vez se asigna a la medición el puerto de origen de capa de transporte que los PPG deben utilizar para intercambiar los mensajes. Una vez asignados estos parámetros, se inicializa el estado de la medida en "$Measure\_Starting$".

El siguiente paso es determinar las direcciones de los PPG origen y destino de la medida a partir del *path* recibido en la solicitud. En base a esta información y el puerto asignado se generan las ONRPs asociadas a la medida. Para evitar saturar las *flow tables* de los switch se asigna un tiempo de vida de 120 segundos a las ONRPs, dado que sólo serán de utilidad para una medida particular. Cuando se terminan de configurar las ONRPs correspondientes, se actualiza el estado de la medida a "$Measure\_Started$".

Con esto se da por finalizada la etapa de inicialización y se está en condiciones de generar la solicitud de medida como se explica a continuación.

Solicitud de medida

En este punto se procede a generar la solicitud de medida que será encaminada hacia el PPG origen encargado de iniciar la medición.

Para ello se hace uso del identificador de la medida y el puerto asignados anteriormente junto con las direcciones de los PPGs involucrados y los parámetros asociados a la medición brindados por el usuario. En la sección 3.7.3 se describen en profundidad los formatos de mensajes asociados a las mediciones, entre los que se encuentra el mensaje inicial que se envía desde la aplicación solicitando la medida.

Una vez generado el mensaje de solicitud, se hace uso del servicio UDP para enviarlo hacia el switch asociado al PPG de origen. Finalmente, se actualiza el estado de la medida a "$Measure\_Sent$".

Finalización de una medida

Una medición se da por finalizada cuando se recibe el resultado junto al identificador asociado desde un PPG. Para ello, desde el programa principal se inicia un *thread* que hace uso del servicio UDP y ejecuta un servidor para la escucha de mensajes de respuesta a medidas.

Cuando llega un mensaje de respuesta al servidor se extrae el identificador de la medida junto con el resultado. A modo informativo para el usuario, se asocia el resultado obtenido a la etiqueta mencionada anteriormente. Finalmente, a partir del identificador se actualiza el estado de la medida a "$Measure\_finish$".

3.3.3. Algoritmo de Ingeniería de tráfico (*TEApp*)

El algoritmo de *TEApp* se desarrolla como un *thread* que es inicializado desde el programa principal cuando el usuario solicita la activación de ingeniería de tráfico. Al solicitar la activación, el usuario debe indicar el par *origen* - *destino* al que se aplicará el algoritmo, junto con la ruta vigente entre dichos nodos. Esto último lo puede consultar desde el menú de la interfaz, que permite solicitar información del estado de las *ONRPs* configuradas o la puede crear en el caso de que aún no exista una ruta asociada. A su vez, se debe especificar el período asociado al bucle de *TEApp*. Es decir, cada cuánto tiempo se realizará el monitoreo de la red, en conjunto con *MonApp*, para decidir si es conveniente cambiar de ruta o no.

Se distinguen dos etapas principales en el algoritmo. Por un lado la inicialización, donde se relevan y calculan datos necesarios para la ejecución del mismo, y luego el bucle con la lógica principal del algoritmo.

Inicialización

Al comienzo se cargan parámetros requeridos por *MonApp* como la matriz de probabilidad de cambio de estado de las distintas rutas, los costos de medir y los posibles niveles de retardo como se explicó en la sección 1.3.3. Esto es utilizado por *TEApp* para determinar el estado de las rutas a partir de las medidas de *delay*.

Una vez relevados estos datos, se solicita a *MeasureApp* la medición de las rutas existentes entre el par *origen* - *destino* a analizar. En base a las mediciones, el algoritmo determina el estado inicial de las rutas y genera un vector de estados de la siguiente forma:

$$state = (estado\_ruta\_1, tiempo\_ruta\_1, estado\_ruta\_2, tiempo\_ruta\_2, ...)$$

Se indica en orden, el estado de cada ruta junto con el tiempo desde que la ruta fue medida, que al inicio vale cero. Esto es utilizado por *MonApp* para hacer las estimaciones de retardo y determinar las rutas a medir en cada iteración. Aunque *MonApp* permite trabajar con una cantidad arbitraria de estados, en este proyecto se trabajará con dos posibles estados para las rutas, alto o bajo. A modo de ejemplo, un posible vector de estados inicial, trabajando con dos rutas y dos posibles estados para cada una sería $state = (0, 0, 1, 0)$. En este caso, la primera

ruta se encuentra en estado bajo y la segunda en alto, mientras que los tiempos valen cero dado que se trata del instante inicial.

Bucle *TEApp*

En cada iteración se solicita a *MonApp* que determine las rutas a medir. Una vez realizadas las mediciones pertinentes, se actualiza el vector de estado de las rutas según la siguiente lógica:

- Si la ruta fue medida, determina el nuevo estado y restablece el instante de medida asociado en cero.
- Si la ruta no fue medida, mantiene el estado anterior y suma uno al instante de medida asociado.

Finalmente, se elige la ruta de menor retardo estimado por *MonApp*. Si la misma difiere de la ruta que se encuentra actualmente configurada, se solicita a *RouteApp* que haga las modificaciones necesarias para que la *ONRP* siga el nuevo camino. Por el contrario, si coincide con la ruta actual, la misma permanece inalterada y se espera a la siguiente iteración para repetir el procedimiento.

Una vez activado, *TEApp* permanece en ejecución, iterando según el tiempo especificado por el usuario. Se habilita una opción para desactivar el algoritmo si lo deseara, en cuyo caso se detiene el *thread* iniciado.

3.4. Procesamiento de mensajes ARP y direcciones MAC

Una dirección MAC es un identificador único que se le asigna a las tarjetas de red con el objetivo de que cada uno de los dispositivos producidos sean inequívocamente identificables. En las redes cableadas e inalámbricas, está formada por 48 bits representados en dígitos hexadecimales; los primeros 24 se utilizan para identificar al fabricante siendo únicos para cada uno de ellos, mientras que los 24 restantes sirven para identificar al producto.

En una red LAN antes de enviar un paquete es necesario saber cual es la dirección MAC del destinatario. Esto se debe a que los dispositivos descartan los paquetes recibidos si contienen una dirección MAC de destino distinta a la que se le asignó. No obstante, existen direcciones MAC como la de broadcast (FF:FF:FF:FF:FF:FF) que pueden ser utilizadas como direcciones de destino para ciertos mensajes sin que el receptor los descarte.

Como se mencionó, la dirección MAC responde al hardware del dispositivo y al trabajar en una red IP no es utilizada en capas superiores. Por lo que al momento de enviar un paquete que tiene como destino cierta dirección IP, si el receptor se encuentra en la misma LAN se deberá contar con su dirección MAC y en caso contrario se necesitará saber la dirección MAC del gateway.

La identificación de la dirección MAC de destino que debe tener el paquete a enviar a cierta dirección IP se obtiene haciendo uso del protocolo ARP. Este genera una tabla denominada “tabla ARP” que en sus entradas tiene la dirección

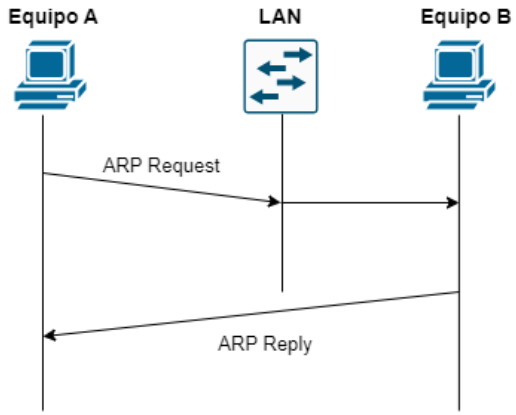


Figura 3.8: Secuencia ARP.

IP con la dirección MAC correspondiente, por lo que para enviar un paquete basta con ver la tabla mencionada. Para lograr generar la tabla se envía la secuencia de paquetes mostrados en la Figura 3.8:

- ARP Request: Este mensaje es enviado a todos los equipos de la LAN utilizando como destino la dirección MAC de broadcast. En él se pregunta quién tiene la IP requerida que para fijar idea la definiremos como IP_B.
- ARP Reply: El equipo B tiene la dirección IP_B, por lo que genera un paquete respondiendo que él tiene la IP y se lo envía directamente al Equipo A quien es el que envió la solicitud.

3.4.1. Procesamiento de mensajes ARP

En este proyecto el switch perteneciente a cada PoP posee una dirección IP e incluso funciona como el gateway de la red LAN, por lo que es necesario que responda de forma correcta a los mensajes ARP Request. Para ello la aplicación SmartRoute debe procesar de forma correcta los mensajes mencionados. La Figura 3.9 muestra la secuencia de los mensajes, siendo procesados de la siguiente forma:

- ARP Request: Al momento de cargar un PoP a la aplicación se genera una entrada en el switch OpenFlow perteneciente al PoP en la cual los paquetes hacen *match* si son del tipo ARP y la acción a realizar es enviarlo al controlador.
- Packet In: Al llegar al controlador son almacenados y la aplicación los procesa. Lo primero que se realiza es validar sobre qué IP se pregunta, si pertenece a un switch de un PoP se continua y de lo contrario se descarta. Luego, se extrae la dirección IP y MAC de origen, y la IP por la cual se pregunta. Con estos datos se genera el mensaje ARP Reply que es enviado en forma de Packet Out por el controlador al switch que posee la IP solicitada.

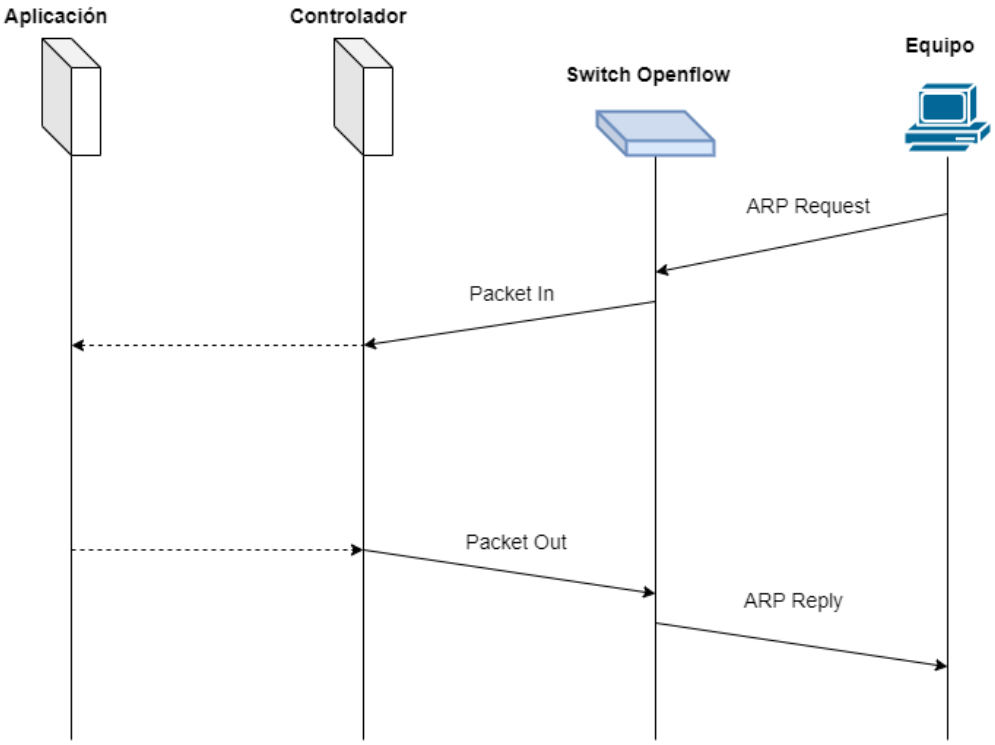


Figura 3.9: Procesamiento de mensajes ARP.

- ARP Reply: Los Packet Out son recibidos por el switch OpenFlow y son enviados en forma de ARP Reply al equipo que envió el mensaje ARP Request.

3.4.2. Procesamiento de direcciones MAC

En esta implementación la aplicación SmartRoute es la encargada de cargar las Flows necesarias a los switches OpenFlow para el correcto funcionamiento del sistema. En muchos casos una vez que los paquetes hagan *match* deben ser enviados a otros equipos de la red, por lo que es necesario mantener un registro de las direcciones MAC de los equipos que están conectados a cada PoP. Para lograr lo mencionado se generan dos métodos, uno dinámico y otro estático.

Modo dinámico

El modo dinámico se aplica utilizando los mecanismos de descubrimiento de topología que posee el controlador OpenDaylight. Para el descubrimiento de los dispositivos conectados al switch OpenFlow, el controlador utiliza los mensajes del protocolo "Link Layer Discovery Protocol" (LLDP). El mecanismo funciona de la siguiente forma, los dispositivos que se encuentran en la red envían paquetes LLDP de los cuales se puede extraer información como la identidad del dispositivo, el

Direcciones IP de los equipos conectados	Dirección MAC asociada a la dirección IP	Puerto del switch donde se encuentra el equipo asociado a la dirección IP
Dirección IP 1	Dirección MAC 1	Conector 1
Dirección IP 2	Dirección MAC 2	Conector 2
Dirección IP 2	Dirección MAC 3	Conector 3

Figura 3.10: Tabla ARP generada.

puerto al cual está conectado y direcciones IP y MAC. Estos paquetes son recibidos por los switches OpenFlow y los envían al controlador notificando un nuevo evento de descubrimiento de enlace. Cuando el controlador recibe un paquete con la notificación mencionada lo procesa y en caso de encontrar cambios en los dispositivos conectados actualiza en su base de datos la topología de red. La información se puede extraer del controlador enviando una solicitud RESTCONF a la siguiente URL "/restconf/operational/network-topology:network-topology/topology/flow:1".

En la aplicación se realizó una función que consulta periódicamente la URL mencionada con el objetivo de detectar las direcciones IP que se tiene en la topología y asociarlas a la dirección MAC del equipo que las posee. Esta información es guardada en una variable (ip_to_mac) dentro de la aplicación que genera una tabla de la forma mostrada en la Figura 3.10.

Modo Estático

En el modo estático es el usuario quien carga la variable ip_to_mac mencionada en la subsección anterior, por lo que se debe mantener un control estricto de los dispositivos conectados a la red.

Para su implementación se usa un archivo de configuración en el directorio de la aplicación llamado "arptable.txt". En este archivo se escribe la dirección MAC, dirección IP y el puerto físico por el cual se alcanza el equipo receptor. Durante la ejecución de la aplicación se puede hacer cambios en el archivo y luego seleccionar una opción para actualizar los datos, el cual leerá nuevamente el archivo mencionado actualizando los parámetros necesarios.

3.5. Uso de Switches OpenFlow

3.5.1. Clasificación de *Flow Tables*

Se organizan las tablas en los switches OpenFlow para separar objetivos y facilitar la identificación y procesamiento de paquetes enviados hacia el controlador. Según el tráfico entrante se definen acciones específicas a realizar en cada tabla. El siguiente desarrollo se basa en que el total de tablas disponibles es de 254, por lo tanto se puede trabajar con las *flow tables* con identificador entre 0 y 253.

Flow Table 0

El objetivo de esta tabla es discernir el tráfico entrante. Se distingue si se trata de comunicaciones entre *PoPs*, mensajes ARP o mensajes asociados a medidas de rutas. Según el tipo de tráfico a ser administrado por la aplicación, continua el pipeline en la tabla que corresponda.

- Tráfico que aún no ha sido encaminado:
  Se trata de paquetes UDP/TCP con IP de origen perteneciente a la subnet del *PoP* actual y destino en la subnet de otro *PoP* de la red. La acción es continuar el pipeline en la *flow table 1*.

- Paquetes que se encuentran en tránsito:
  Se verifica que la IP de origen es la de otro switch de la red, mientras que el destino es el switch actual. En este caso continua el pipeline en la *flow table* 2.

- Mensajes ARP:
  Se determina si el paquete entrante es un *ARP request* solicitando la IP del switch actual y se envía al controlador para generar la *ARP response* correspondiente.

- Tráfico asociado a medidas de rutas:
  Encaminar las solicitudes de medida hacia el PPG de origen y enviar los mensajes de respuesta hacia la aplicación.

- Tráfico restante:
  El resto del tráfico, que no requiere ser procesado por la aplicación, es enviado por el puerto ALL.

Flow Table 1

En esta tabla, los paquetes entrantes tienen origen en el *PoP* actual. Es aquí donde se determina si pertenecen a una *ONRP*, en cuyo caso se asocia la tabla de *ONAT* para continuar el procesamiento.

- Paquetes pertenecientes a una ONRP:
  Si el paquete entrante pertenece a alguna *ONRP*, se envía a la tabla de *ONAT* para inicializar el encaminamiento. Utilizando los campos *match* de la *flow entry* se comprueba que el paquete cumpla los parámetros de la ONRP. Se configura el valor de *ONRP Id* en el campo metadata y continua el pipeline en la tabla de *ONAT de origen*.

- Paquetes que no pertenecen a una ONRP:
  Si se determina que los paquetes no pertenecen a una ONRP, estos son descartados.

Flow Table 2

El objetivo es procesar paquetes que fueron encaminados y se encuentran en tránsito. En conclusión, aquí se procesa tráfico correspondiente a los *switch* intermedio y destino.

- Primer paquete de una ONRP que ingresa al switch intermedio:
  Al crear una ONRP, se instala una entrada en la tabla 2 del switch intermedio para capturar el primer paquete perteneciente a la ONRP que transite por la red. La acción en este caso es encapsular el paquete como *PACKET_IN* y enviarlo al controlador para generar la entrada de encaminamiento correspondiente.

- Tráfico intermedio:
  Se trata de paquetes que previamente han ingresado a la red, se encuentran en un switch intermedio y ya se ha configurado la entrada de encaminamiento. En este caso, se realizan los cambios necesarios en los campos y se encamina a través de la misma interfaz por la que llegó.

- Tráfico destino:
  Son paquetes que previamente han ingresado a la red y se encuentran en el switch destino. La acción en este caso es configurar el *ONRP Id* en el campo metadata y continuar el pipeline en la tabla de *ONAT* destino para recuperar la información original del mensaje.

Flow Table ONAT

Se encargan de realizar la traducción de ingreso a la red en los switch origen y de recuperar la información original del mensaje en los switch destino. Se destinan para esta función las tablas del switch con identificador entre 3 y 253.
A cada ONRP se le asigna de forma aleatoria una tabla de *ONAT* en el switch origen y destino en el rango de valores mencionado. Por ende, cada tabla de *ONAT* puede estar asociada a más de una ONRP. Para distinguirlas se hace uso del campo metadata donde se configura el *ONRP Id* correspondiente.

- ONAT origen

  - Tráfico que aún no ha sido encaminado:
    Se distinguen dos casos para los cuales se toman acciones diferentes.
    Por un lado, se considera el tráfico perteneciente a una ONRP sin un *ONAT Id* asociado. En este caso, se procede a enviar los paquetes al controlador para su procesamiento.
    Luego, se tienen paquetes entrantes que ya tienen asignado un *ONAT Id*. Se procede a realizar la traducción de ingreso y encaminar el tráfico por la red.

- ONAT destino

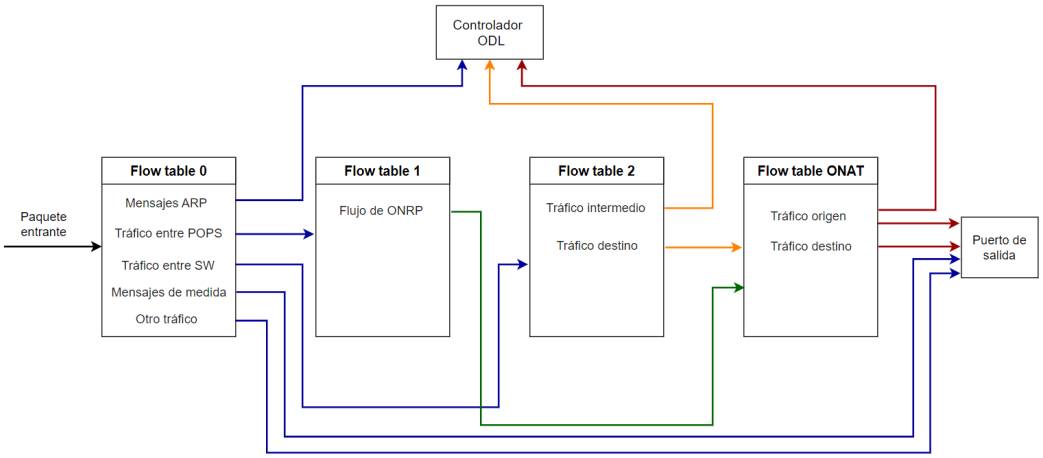


Figura 3.11: Procesamiento de mensajes en las *flow tables*.

- Tráfico destino:
  Cuando los paquetes llegan al switch destino se procede a recuperar la información original del mensaje en base al *ONAT Id* y enviar hacia el host final.

En la Figura 3.11 se muestra un diagrama que resume la información presentada en el presente apartado. Se presenta la organización de las *flow tables* y las acciones que implementan dado un paquete entrante.

3.5.2. Configuración de *Flow Entries*

Como se mencionó en la sección 2.1, el controlador brinda una API rest mediante la cual se pueden manipular las tablas de los switches OpenFlow conectados. En particular, permite realizar operaciones como lectura, edición o eliminación de *flow entries*. En este apartado, se describe la metodología utilizada para configurar los switches según lo requerido por la aplicación.

El objetivo es implementar un mecanismo que facilite la configuración y sea escalable dada la variedad de *flow entries* a cargar. Para ello, se implementan plantillas con los distintos formatos de *flow entry*, según las especificaciones de la aplicación. Asimismo, se crea una función encargada de gestionar dichas plantillas y generar la solicitud correspondiente al controlador.

Función CargarFlows

La función en cuestión toma como entrada los parámetros de configuración junto con un identificador del formato de *flow entry* a utilizar. Con esta información, procede a completar los campos de la plantilla correspondiente, y finalmente genera la solicitud PUT al controlador. Se establece el siguiente formato para el envío de los parámetros de entrada:

[Match_IP_src, Match_IP_dest, Match_L4_portsrc, Match_L4_portdest, Match_eth_src, Match_eth_dest, New_IP_src, New_IP_dest, New_src_port, New_dest_port, New_eth_src, New_eth_dest, Output_node_connector, Protocolo, Switch_id, Table_id, Flow_id, Flow_name, Prioridad, Cookie, Metadata, Función, Hard_timeout]

El identificador de la plantilla a utilizar se indica en el campo *Función*. Según el caso, se pasa la información necesaria en el orden indicado, dejando entre comillas los campos que no se especifican. De esta forma, se construye adecuadamente el cuerpo de la solicitud, así como también el URL asociado.
Hay campos obligatorios que la función requiere para configurar correctamente los switches. Estos son *switch_id*, *table_id*, *flow_id*, *flow_name*, *prioridad*, *función* y *hard_timeout* que sirven tanto para identificar las *flow entries* como para crear el URL de la solicitud.

Supongamos que se desea configurar la tabla 2 del switch openflow:2, de modo que envíe al controlador el flujo con IP de origen 172.16.1.253/32, destino 172.16.2.253/32 y puerto de origen de capa de transporte 4001. En la Figura 3.1 se muestran los parámetros a enviar a la función con el fin de crear la entrada correspondiente en el switch.

Tabla 3.1: Ejemplo de parámetros de configuración de una flow entry.

match_ip_src	172.16.1.253/32
match_ip_dest	172.16.2.253/32
match_L4_portsrc	4001
output_node_connector	CONTROLLER
protocolo	udp
switch_id	openflow:2
table_id	2
flow_id	1
flow_name	flow 1
prioridad	100
función	1
hard_timeout	0

3.6. Tratamiento de paquetes

Un desafío tratado en este trabajo fue el procesamiento de paquetes encapsulados como *PACKET_IN* y enviados hacia el controlador. Como se explicó anteriormente, una de las acciones en las flow tables es enviar los paquetes al controlador para su procesamiento. Por ende, es de importancia lograr capturar estos paquetes para ser procesados en la aplicación como corresponda.

Con este propósito, se han desarrollado dos módulos en el controlador. En primer lugar, disponemos del módulo denominado *PacketInListener* destinado a la recepción de mensajes tipo *PACKET_IN*. Por otro lado, se encuentra el módulo *MyPacket*, que ofrece una interfaz de comunicación *RPC* para la transmisión de paquetes desde un switch OpenFlow. En el Apéndice B se profundiza en el desarrollo de módulos en el controlador OpenDaylight, y específicamente se detalla la implementación de los dos módulos previamente mencionados.

3.6.1. Clasificación

La organización de las *flow tables* descripta en la sección 3.5.1 facilita el procesamiento de mensajes *PACKET_IN* en la aplicación. En función de la tabla desde la que se reciben, se determina el tipo de paquete, si se trata de un switch origen o intermedio, y se lo procesa como corresponda.

- Si *table id = 0*:
  Se trata de un posible mensaje *ARP request*. Lo primero es observar el campo *Type* (Ethernet) del paquete recibido para comprobar que sea *ARP* y se procede a generar el mensaje *ARP response* correspondiente, como se explicó en la sección 3.4.1.

- Si *table id = 2*:
  El paquete fue recibido desde un switch intermedio de una ONRP. En particular, es el primer paquete perteneciente a la ONRP en transitar por la red. Se procede a crear en la misma tabla la entrada encargada de modificar los campos y encaminar hacia el próximo salto.

- Si *table id* $\geqslant$ *3*:
  El paquete fue recibido desde el switch origen de una ONRP. Se trata del primer flujo perteneciente a la ONRP que transita por la red y aún no tiene un *ONAT Id* asociado. Lo primero es asignar un *ONAT Id* y registrar la información del flujo junto con el nuevo identificador. Luego, se crean las entradas en las tablas de *ONAT* de origen y destino encargadas respectivamente del encaminamiento inicial y recuperación de información final.

3.6.2. Reenvío de paquetes

Cuando un paquete es dirigido al controlador y se procesa adecuadamente, debe ser reenviado a través de la red con los campos modificados de forma apropiada. Esto se establece con el propósito de evitar la pérdida del paquete que atraviesa la red y es enviado al controlador. Asimismo, es posible que un paquete sea dirigido al controlador y requiera generar una respuesta que contenga la información necesaria; este es el caso de los mensajes de solicitud ARP.

Para llevar a cabo esto, se aprovecha la utilidad del módulo *MyPacket* previamente mencionado. Tomando como base el payload recibido, se genera el paquete a enviar (en formato de datos en bruto). Una vez completado este proceso, se procede a generar la solicitud *send-packet*. Esta solicitud incluye la información acerca

del switch desde el cual se enviará el paquete, la interfaz de salida y el mensaje previamente creado.

A continuación, se describe el proceso de generación del paquete a enviar en función del *PACKET_IN* recibido, distinguiendo entre mensajes ARP request y paquetes recibidos desde un switch origen o intermedio:

ARP request

En primer lugar, se obtienen las direcciones IP y dirección MAC del contenido de la consulta ARP. Posteriormente, se procede a crear la respuesta basada en el formato de mensaje *ARP response*. Los campos del mensaje se completan utilizando las direcciones IP y MAC obtenidas, así como la dirección MAC por la que se consultó originalmente.

Tráfico origen e intermedio

Para el tráfico recibido desde un switch origen o intermedio, se extrae el payload recibido y se recuperan las direcciones IP, puertos y direcciones MAC. Una vez determinanda la ONRP a la que pertenece el paquete, se identifica el próximo salto y se procede a modificar los campos del payload para que pueda ser encaminado correctamente. Por último, se calculan los nuevos valores de checksum y se modifican los campos correspondientes.

3.7. Probe Packet Generator (PPG)

El Probe Packet Generator (PPG) es el elemento de medición en la arquitectura implementada diseñado con el fin de obtener las métricas de QoS de las rutas presentes en la red. Se implementa como una pieza de software en lenguaje Python a ejecutar en los hosts destinados a esta finalidad.

3.7.1. Arquitectura PPG

La arquitectura de software se basa fuertemente en lo desarrollado en [21] y está conformada por cuatro bloques con sus respectivos objetivos: Task Manager, Measure Initializer, Probes Receiver, Cleaner. En la Figura 3.12, se presenta un diagrama de la arquitectura del PPG con los bloques mencionados y los servicios que utilizan.

Estos bloques se ejecutan en paralelo mediante el uso del serivico Threading de Python. Esto permite la separación de objetivos, según el rol que cumplen en el funcionamiento del PPG.

En este trabajo sólo se implementa la medición del Round-trip time (RTT), pero como se comentará más adelante, es posible adaptar el diseño a otras métricas de QoS.

A continuación, se presentará una breve descripción de la funcionalidad de cada componente junto con detalles de su implementación.

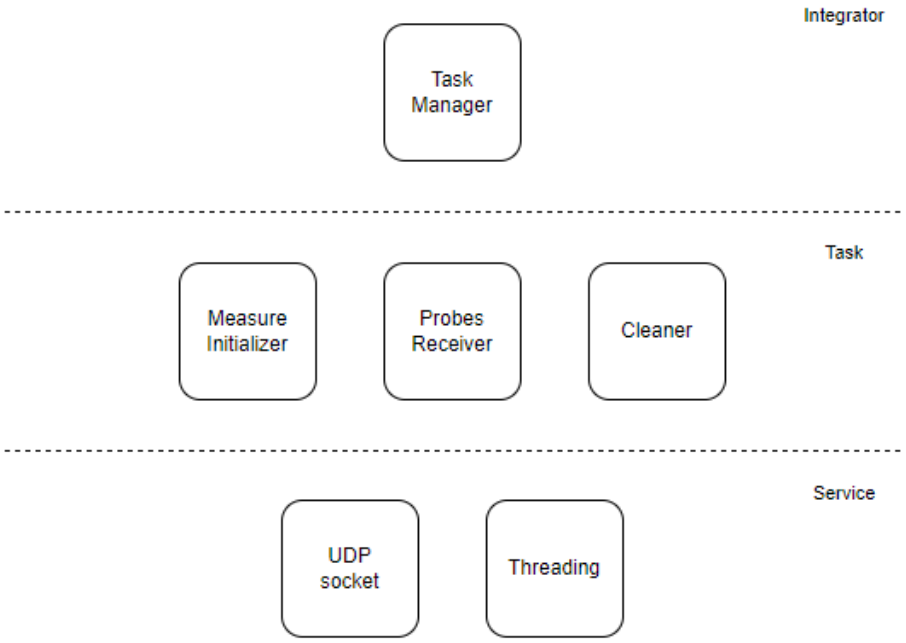


Figura 3.12: Arquitectura de software del PPG.

Task Manager

Este es el componente principal, encargado de inicializar el resto de bloques del sistema. Solicita la ejecución paralela de dichos bloques, mediante el uso de tres threads, uno asignado a cada tarea específica.

Measure Initializer

Este bloque se encarga de inicializar las medidas requeridas por la aplicación. Utiliza el servicio Socket de Python para establecer un servidor que escucha las solicitudes correspondientes. Tras la inicialización del bloque, se procede a cargar desde un archivo de configuración la dirección IP del PPG, así como los puertos necesarios para establecer la comunicación tanto con la aplicación como con otros PPG.

Cuando se recibe una solicitud, se obtiene la dirección IP del PPG destino y el puerto de origen de capa de transporte asignado a la medida ($Measure\_Port$). Este puerto específico permite distinguir medidas que llegan a un mismo destino. A su vez, se obtienen parámetros asociados a la medida que se describirán más adelante en esta sección. Una vez recopilados los datos necesarios, se genera el mensaje de medida hacia el destino correspondiente.

Probes Receiver

Al inicializar este bloque, también se realiza la lectura de un archivo de configuración. De allí se obtienen parámetros como el puerto de capa de transporte por el que los PPG pueden recibir mensajes de medida. Mediante el uso del servicio Socket y los parámetros obtenidos se inicializa el receptor. Una vez que se recibe un mensaje de medida, se analiza el contenido y se distingue entre dos casos:

1. Si se trata del destino de una medición, se debe reenviar el mensaje de medida hacia el origen, con algunos parámetros modificados como se verá más adelante.

2. Si es el origen de la medición, hay que verificar si la misma ha finalizado. En ese caso, se procede a generar el mensaje con el resultado y se envía hacia la aplicación, utilizando los parámetros obtenidos del archivo de configuración. Según los parámetros requeridos por la aplicación, es posible que el mensaje de la medida regrese al origen sin que la medición haya finalizado. En este caso, se debe reenviar el mensaje al PPG de destino.

Cleaner

Este bloque se encarga de mantener un registro de las medidas que se encuentran en proceso y verificar periódicamente el tiempo en que fueron inicializadas. Si alguna medida supera un tiempo límite establecido sin recibir la respuesta correspondiente, el bloque se encarga de descartar o reenviar la medida según el caso. Para determinar cómo proceder se basa en otro valor establecido que corresponde a la cantidad máxima de reintentos que puede alcanzar una medida.

1. Si la cantidad máxima de reintentos no fue alcanzada, se vuelve a enviar la medida, aumentando el número de reintentos de la misma.

2. Si se alcanzó el máximo de reintentos establecido, descarta la medida y responde a la aplicación con un mensaje de error.

Tanto el *timeout* definido como el máximo de reenvíos son criterios establecidos y se cargan desde el archivo de configuración. El *timeout* utilizado por el bloque *Cleaner* es calculado en base al valor establecido y parámetros que caracterizan a la medición.

3.7.2. Comunicación *SmartRoute - PPG*

Un aspecto a considerar durante el diseño consiste en la gestión de la comunicación entre el PPG y la aplicación. El objetivo es dirigir las solicitudes de medición de un *path* desde la aplicación hasta el PPG, así como encaminar la respuesta generada desde el PPG de vuelta hacia la aplicación.

Se asignan dos puertos con el propósito de establecer esta comunicación. Uno está reservado para la recepción de solicitudes de medición en los PPG, mientras que el otro está destinado a recibir los resultados en la aplicación. En base a

Tabla 3.2: $ONRPs$ a crear para la medición de un path entre el PoP_1 y PoP_3.

ONRP	IP Ori	IP Dest	Path	Protocolo	Puerto Ori	Puerto Dest
Ida	PPG_1	PPG_3	[1,2,3]	UDP	$Measure\_Port$	-
Vuelta	PPG_3	PPG_1	[3,2,1]	UDP	$Measure\_Port$	-

los puertos reservados y la dirección IP asociada a la aplicación, se generan dos entradas en la flow table 0 de los switches.

Una de estas entradas está encargada de detectar las solicitudes de medición y dirigirlas hacia el PPG. En este caso, se debe comprobar que la dirección IP de origen corresponda a la aplicación y que el puerto de destino sea el reservado para la escucha en los PPG. Si es así, se realizan las modificaciones necesarias en los campos para encaminar la solicitud hacia el PPG, que se encargará de iniciar la medición.

La segunda entrada se encarga de detectar los resultados de una medición provenientes de un PPG y dirigirlos hacia la aplicación. Se debe verificar que la dirección IP de destino coincida con la de la aplicación y que el puerto de destino sea el asignado para la escucha en la aplicación. Si se cumplen estas condiciones, se modifican los campos para encaminar los resultados de la medición hacia la aplicación.

3.7.3. Metodología de medición

En este apartado se describirá la metodología de medición implementada, desde la solicitud de inicialización de una medida hasta la notificación del resultado. Se detallarán los criterios establecidos para lograr el intercambio de mensajes de medida así como el cálculo del RTT solicitado.

El sistema establece caminos unidireccionales pero las medidas involucran tanto la ida como la vuelta. El método es diseñado de tal forma que los mensajes de medida sigan el mismo camino en ambos sentidos. Para ello se crean dos políticas de ruteo al momento de realizar la medición, una para la ida y otra para la vuelta. Si se desea medir el *path* entre el PoP_1 y PoP_3 a través del PoP_2, se deben crear dos $ONRPs$ como se muestra en la Figura 3.13.

Como se mencionó anteriormente, la aplicación asigna el puerto de origen de capa de transporte que los PPG deben utilizar para llevar a cabo cada medición ($Measure\_Port$). Esto permite distinguir *paths* entre un mismo origen y destino. En la tabla 3.2, se muestran los parámetros de las $ONRPs$ a crear siguiendo el ejemplo anterior.

Mensajes involucrados en la medición

Se han implementado formatos de mensaje que tienen como finalidad estandarizar la comunicación entre los PPG y simplificar el procesamiento de los mensajes intercambiados.

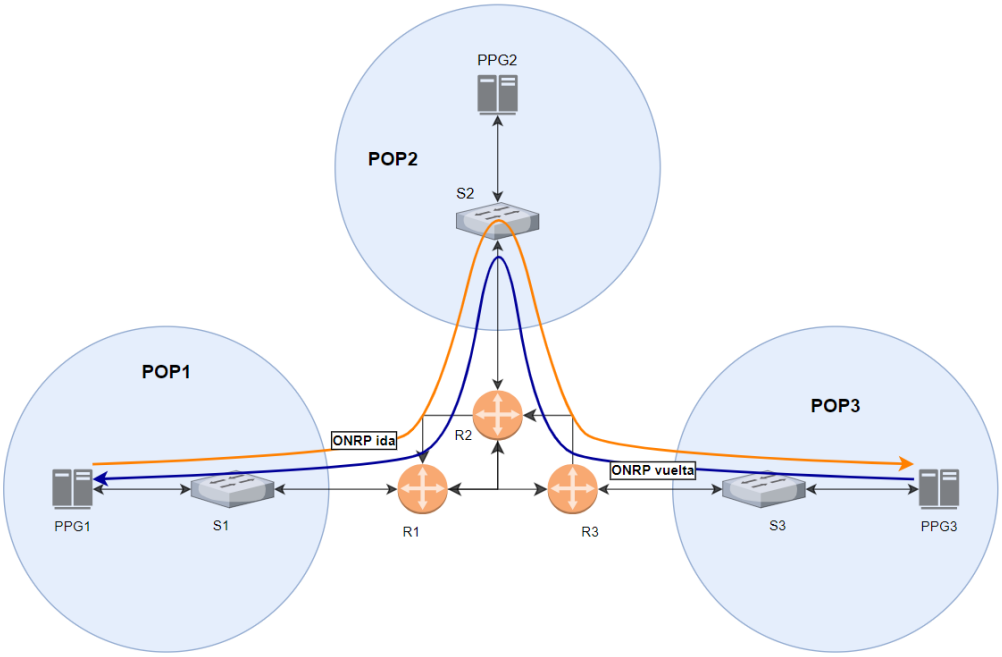


Figura 3.13: Medición de un path entre el PoP_1 y PoP_3.

- Mensaje de inicialización:
  Se trata del mensaje de solicitud de medida enviado desde la aplicación y recibido en el PPG encargado de iniciar la medida. Allí debe viajar la información sobre el puerto de origen al que establecer la comunicación entre los PPG, la dirección IP del PPG destino y los parámetros adicionales asociados a la medición. Se define el siguiente formato de mensaje:

  Measure/Id/PuertoOri/PPGDest/Throughput/Average/PacketSize

  A modo de control, al comienzo del mensaje se especifica que está relacionado con una medición, y posteriormente se vincula con el identificador generado por la aplicación.
  En la solicitud, se especifican parámetros tales como la cantidad de paquetes que deben ser enviados para llevar a cabo la medición (*Throughput*), la cantidad de viajes ida y vuelta que el paquete debe realizar (*Average*), así como el tamaño del paquete (*PacketSize*).

- Mensajes de medida:
  Son intercambiados entre los PPG y permiten obtener la medida solicitada. El PPG encargado de inicializar la medida recibe la solicitud y genera el mensaje de medida a enviar al PPG destino. En primer lugar, registra el tiempo de inicio de la medida que posteriormente utilizará para el cálculo del RTT. A su vez indica los parámetros requeridos asociados a la medición junto con campos establecidos con el objetivo de identificar el fin de la

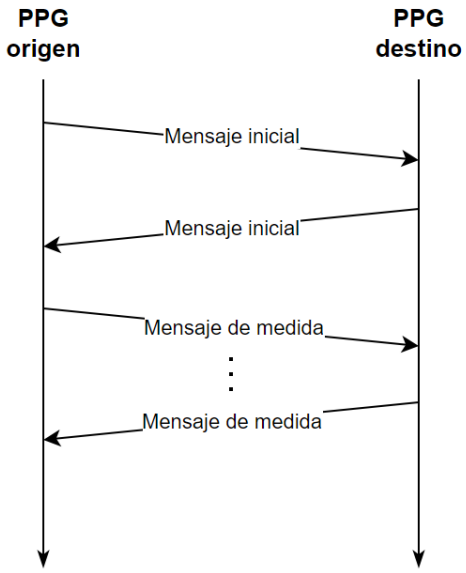


Figura 3.14: Diagrama de comunicación de medida entre PPGs.

medida. A continuación se muestra el formato de mensaje de medida:

RTT/Timestamp/Id/Average/NEnvios/Throughput/NPaquete/PacketSize

El mensaje inicia con la información del tipo de medida a realizar. Como se mencionó anteriormente, en este trabajo sólo se considera el valor de RTT. Sin embargo, se completa este campo para distinguir frente a otro posible método de medida que se quiera incorporar.
El *timestamp* corresponde al tiempo en que inicia la medida y es introducido por el PPG de origen.
El valor *NEnvios* corresponde a la cantidad de veces que el paquete de medida fue enviado por un PPG. Cada vez que un paquete es reenviado a través de un PPG, se debe aumentar este parámetro. Esto es de utilidad para verificar que el paquete cumpla el requerimiento de *Average*.
Análogamente, se utiliza el valor *NPaquete* para identificar los paquetes pertenecientes a una misma medida. Esto está asociado al valor de *Throughput* solicitado y permite determinar si han llegado todos los paquetes de la medida.

En la Figura 3.14 se presenta un diagrama que ilustra la comunicación de mensajes de medida entre los PPG involucrados. Previo al intercambio de mensajes de medida, se envía un mensaje inicial con el objetivo de inicializar el *path* de medida. De esta forma, se evita que la medida incluya los tiempos de comunicación con el controlador.

- Mensaje de respuesta:
  Una vez se ha concluido la medición, se procede a calcular el valor de RTT requerido por la aplicación. Esta tarea es llevada a cabo por el PPG que

inició la medición, y posteriormente, genera la respuesta hacia la aplicación con el siguiente formato:

Measure/Id/medida

Al igual que la solicitud de medición, se indica que el mensaje está asociado a una medida. Luego se registra el identificador de la medida junto con el valor de RTT calculado.

Cálculo del RTT

El valor del RTT se calcula en el PPG de origen una vez que se ha determinado que la medición ha finalizado. El *timestamp* se establece en dicho punto al inicializar la medida, lo que garantiza que no haya conflicto de sincronización de relojes, ya que tanto el momento de inicio como el de finalización de la medición se registran en el mismo dispositivo.

El cálculo del resultado depende de los valores de *Throughput* y *Average* solicitados por la aplicación. En el caso sencillo en el que ambos parámetros tienen un valor de 1, se envía un solo paquete que realiza un trayecto de ida y vuelta. En consecuencia, el valor final de RTT es directamente la diferencia entre el tiempo en que se recibió el paquete y el momento en que se envió inicialmente.

En el caso general, el RTT se calcula según se presenta en la ecuación 3.1, donde se tienen en cuenta los parámetros mencionados previamente. Se puede verificar que en la situación simplificada, la ecuación se reduce al retardo del paquete enviado.

$$RTT = \frac{\sum_{i=1}^{N} \frac{\tau_i}{Avg}}{N} \tag{3.1}$$

- N: número de paquetes que se envían para realizar la medida (*Throughput*).
- Avg: cantidad de viajes origen → destino → origen a realizar (*Average*).
- τ_i: retardo calculado para el *i-ésimo* paquete de la medida.

En el presente capítulo, se describió en profundidad el sistema propuesto en este proyecto. Se presentaron los componentes de la arquitectura implementada, y se analizó la funcionalidad de los algoritmos de ruteo, medición e ingeniería de tráfico que conforman la aplicación *SmartRoute*. A su vez, se plantearon aspectos importantes respecto a la implementación del sistema, como el procesamiento de mensajes ARP, el uso de los switches OpenFlow y el funcionamiento de los PPG. En el siguiente capítulo se describirá el ambiente de pruebas implementado para la validación del sistema.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Ambiente de pruebas

En este capítulo se describe el ambiente de pruebas utilizado para validar la aplicación diseñada en un entorno emulado y en una maqueta de laboratorio. En la sección 4.1 se brinda una descripción de Mininet, el emulador de red utilizado y la topología implementada en este entorno. Luego, en la sección 4.2 se detalla la arquitectura de la maqueta de laboratorio diseñada y los componentes utilizados.

4.1. Mininet

4.1.1. ¿Qué es Mininet?

Mininet es un emulador de red que crea una red virtual de hosts, conmutadores, controladores y enlaces. Los hosts de Mininet ejecutan software de red estándar de Linux, y sus conmutadores admiten OpenFlow para un enrutamiento personalizado altamente flexible y redes definidas por software.
Mininet proporciona una manera sencilla de obtener el comportamiento correcto del sistema y de experimentar con topologías. [4]

4.1.2. Topología diseñada

La topología propuesta consiste en cuatro switches OpenFlow (versión 1.3), cuatro enrutadores y ocho hosts, dos vinculados a cada uno de los switches. Adicionalmente, se dispone de un host conectado a cada switch, para cumplir la función de PPG. En la Figura 4.1 se muestra una representación gráfica de esta topología.

Se define un host adicional con la dirección IP 172.16.10.2 y se conecta al enrutador R_2. La utilidad de este host es simular una conexión de red externa a la topología y es aquí donde se ejecutará la aplicación. De esta forma, la aplicación puede establecer comunicación con los PPG en el caso de requerir la ejecución de medidas.

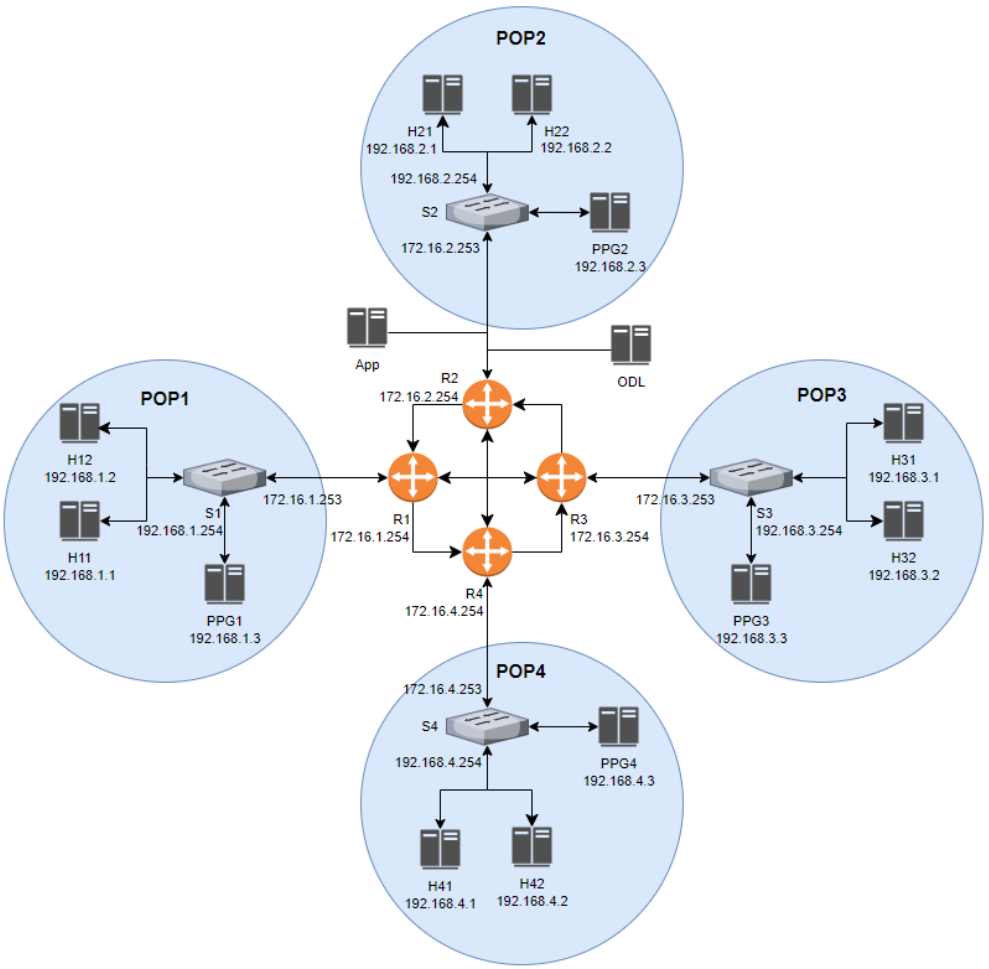


Figura 4.1: Representación gráfica de la topología diseñada en Mininet.

4.1.3. Consideraciones de la implementación

A continuación, se detallan aspectos a tener en cuenta al implementar la topología en Mininet:

- Para lograr conectar Mininet con el controlador OpenDaylight es necesario instalar la funcionalidad *l2switch*. Esto impone una restricción en la versión del controlador a utilizar, dado que a partir de la versión Fluorine esta funcionalidad fue removida. En conclusión, Oxygen es la versión más reciente que cuenta con *l2switch*, y es la elegida para el desarrollo del presente trabajo.

- Cada vez que se inicia la topología, Mininet asigna de forma aleatoria la dirección MAC a los elementos en la red. Esto debe tenerse en cuenta al momento de crear una *flow entry* que utilice direcciones MAC.

4.2. Maqueta

4.2.1. Convertir router TP-LINK a switch OpenFlow

OpenWrt

OpenWrt es un sistema operativo de código abierto basado en Linux, diseñado específicamente para dispositivos de red. Fue desarrollado con el fin de reemplazar el firmware de fábrica en diferentes dispositivos, para proporcionar a los usuarios un mayor control.

- **Instalación**
  Para instalar el sistema operativo, es necesario realizar una actualización del firmware del router. En este caso, se parte de un router con una versión de OpenWrt ya instalada.

  En un principio, es necesario descargar desde el sitio web de OpenWrt [14] la versión deseada, teniendo en cuenta que debe ser compatible con el modelo específico de router que se tiene. Una vez realizado este paso, y estando conectado mediante un cable al router, se debe acceder a éste (mediante la IP 192.168.1.1), y buscar la opción de *flashear* imagen. Luego de esto, el router ya cuenta con la imagen de software necesaria, y es posible acceder al él mediante la web de Luci, o mediante una conexión ssh, en ambos casos con la IP mencionada, y las credenciales (root/admin), estando conectado mediante un cable ethernet a alguno de los puertos LAN del router.

Open vSwitch

Open vSwitch (OVS) es un switch virtual de código abierto diseñado para su uso en entornos de virtualización y redes definidas por software (SDN). Opera en el nivel de software, y puede gestionar el tráfico entre máquinas virtuales (VMs), y entre VMs y la infraestructura física de red, puesto que puede gestionar tanto puertos físicos como virtuales.

- **Instalación**
  Para esta sección, es necesario que el router cuente con conexión a Internet. Deben ejecutarse los siguientes comandos:
  opkg update
  opkg install openvswitch

- **Configuración switch**
  En primera instancia, se debe modificar la configuración del *firewall* del dispositivo de forma que acepte el envío y recepción de paquetes entre todas las interfaces (cambiar todos los “ reject”por “ accept”). Esto puede hacerse desde la consola del switch (modificando el archivo *firewall*, que se encuentra en la ruta */etc/config*) o desde la interfaz gráfica, bajo el menú *System*.

Luego, se modifica la configuración de las interfaces del switch. Hasta el momento, se accede mediante uno de los puertos de LAN, que están configurados en un *bridge* con la IP por defecto. Luego de este paso, será posible acceder al switch conectándose a la interfaz de WAN, con la dirección IP que se le asigne.

Una vez más, la configuración puede realizarse desde la consola (en la misma ruta que el archivo anterior) o desde la interfaz gráfica de Luci (bajo el menú *Network*).

Las interfaces se asignan como se muestra en la Figura 4.1.

Tabla 4.1: Asignación de interfaces del switch.

Nombre	Interfaz física
WAN	eth0.2
LAN1	eth0.3
LAN2	eth0.4
LAN3	eth0.5
LAN4	eth0.6

Todas ellas se configuran con la opción de protocolo estático, y solo en la interfaz WAN se asigna la IP elegida. Luego de guardar estos cambios (en caso de modificar por consola, se debe reiniciar el switch utilizando el comando " reboot"), y una vez que se accedió mediante esta interfaz, se elimina el *bridge* que existía por defecto.

Creación de *bridge*

Para esto, se deberá crear un archivo llamado sdn.sh en la ruta /usr/share/openvswitch/scripts. El mismo tendrá el siguiente contenido:

```
ovs-vsctl add-br br-lan \
        -- set bridge br-lan other-config:hwaddr=00:00:aa:bb:cc:da \
        -- set bridge br-lan protocols=OpenFlow10,OpenFlow13 \
        -- set-controller br-lan tcp:164.73.38.44:6633 \
        -- add-port br-lan eth0.3 \
        -- add-port br-lan eth0.4 \
        -- add-port br-lan eth0.5 \
        -- add-port br-lan eth0.6
```

Figura 4.2: Contenido del archivo sdn.sh.

En este ejemplo, *br-lan* es el nombre del *bridge*, e IP es la dirección IP del controlador, con el puerto 6633. La dirección MAC configurada en *other-config:hwaddr* debe ser diferente para cada switch que se configure, pues el controlador utiliza el parámetro para identificar cada bridge.

Luego de creado el archivo, se inicia el switch virtual utilizando *ovs-ctl start*, se utiliza el comando *sudo chmod +x*, y luego se ejecuta el script creado.

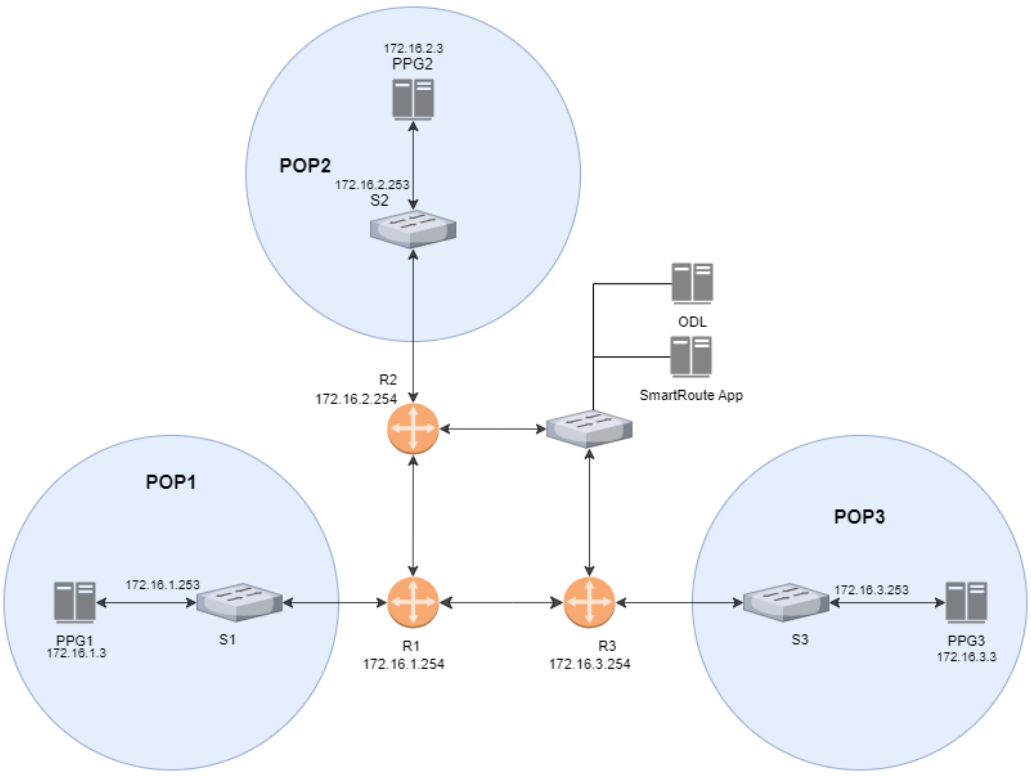


Figura 4.3: Topología de la maqueta implementada.

4.2.2. Maqueta implementada

Se implementó una maqueta con tres routers, tres switches, y tres PPGs. Para los routers, se utilizaron Mikrotik RouterBOARD 433AH, y para los switches TP-Link WR1043ND con OpenFlow. Para el controlador se utilizó una computadora con una máquina virtual con ODL, y los PPGs fueron representados por tres laptops. La topología se realizó como se observa en la Figura 4.3.

En el capitulo presentado se mostraron los distintos ambientes de pruebas utilizados para la validación de la aplicación. El método de instalación y principales comandos usados en los ambientes presentados se encuentra en el apéndice A. En el siguiente capítulo se muestran las pruebas realizadas para la validación de la aplicación.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Pruebas de funcionamiento

En este capítulo se exhiben las pruebas realizadas de los distintos módulos de la aplicación en base a los ambientes de pruebas descritos en el capítulo 4. Se describen las distintas pruebas desarrolladas junto con los resultados obtenidos. En la sección 5.1 se detallan las pruebas realizadas en Mininet, mientras que en la sección 5.2 se presentan las ejecutadas en la maqueta de laboratorio.

5.1. Pruebas en Mininet

A continuación se presentan las pruebas realizadas en ambiente simulado con Mininet. En la Figura 5.1 se muestra la topología utilizada para las pruebas en dicho entorno.

5.1.1. RouteApp

Primera prueba: validación de encaminamiento

Lo primero es comprobar el correcto encaminamiento del tráfico dada una determinada política de ruteo. La política implementada en la red para esta prueba se detalla en la tabla 5.1.

Tabla 5.1: ONRP implementada para la primera prueba de validación.

ONRP Id	IP Org	IP Dst	Path	Protocolo	Puerto Org	Puerto Dst
1	192.168.1.0/24	192.168.4.0/24	[1, 2, 4]	-	-	-

En la Figura 5.2, se muestra la captura de la interfaz de entrada del switch s1. Se observa el paquete con IP de origen h11, destino h41, puerto de origen 4001 y puerto destino 5001.

Luego, en la Figura 5.3 se observa el paquete procesado por s1 antes de ser enviado desde el PoP_1 hacia el PoP_2. Se verifica el cambio de las direcciones originales del mensaje por las direcciones asignadas a los switches s1 y s2. A su

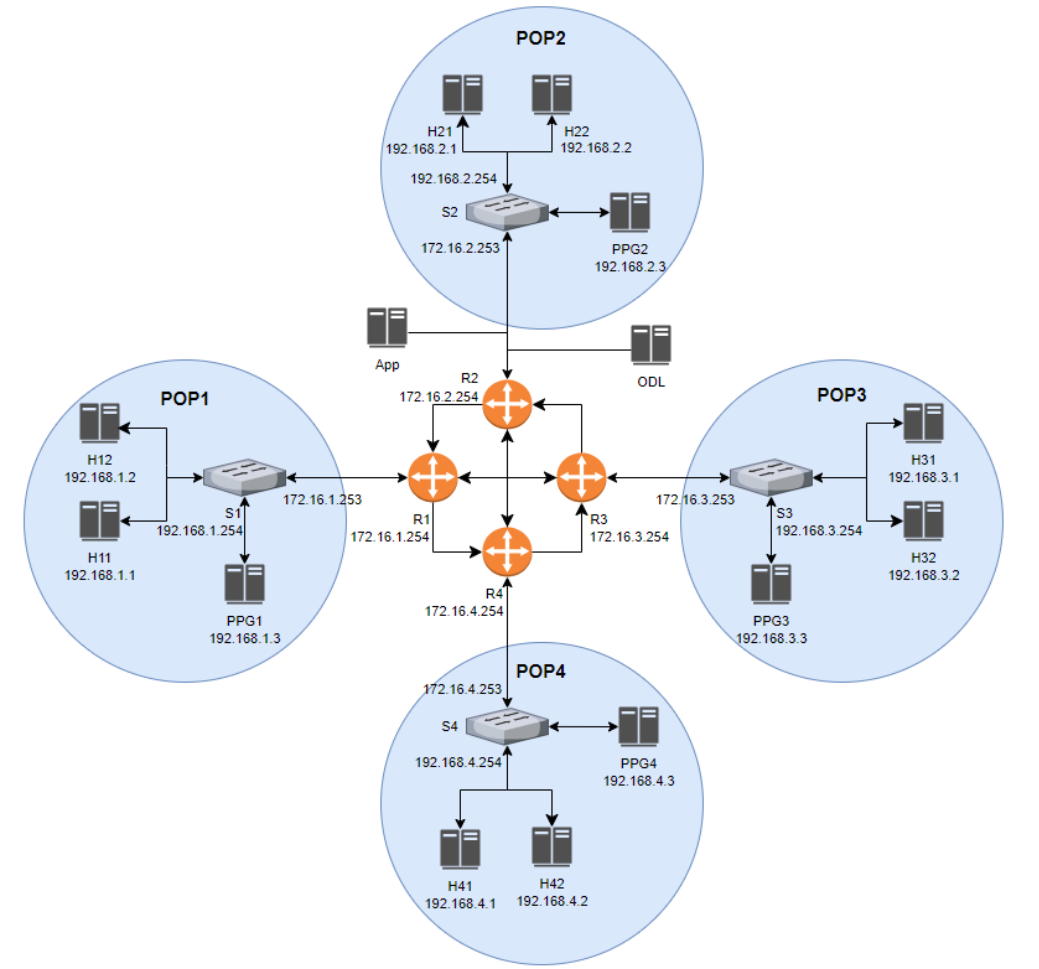


Figura 5.1: Topología implementada para las pruebas en Mininet.

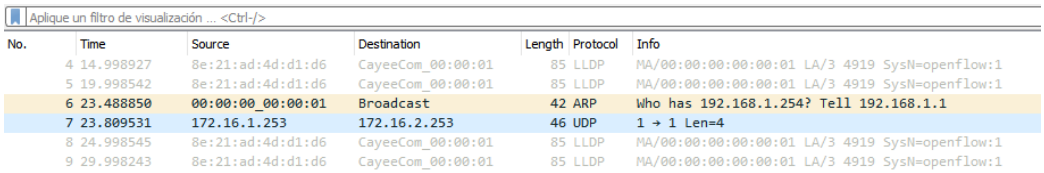
Aplique un filtro de visualización ... <Ctrl-/>

No.	Time	Source	Destination	Length	Protocol	Info
4	14.998927	82:b2:71:f3:86:2a	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:01 LA/1 4919 SysN=openflow:1
5	19.998542	82:b2:71:f3:86:2a	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:01 LA/1 4919 SysN=openflow:1
6	23.488716	00:00:00_00:00:01	Broadcast	42	ARP	Who has 192.168.1.254? Tell 192.168.1.1
7	23.596192	8e:21:ad:4d:d1:d6	00:00:00_00:00:01	60	ARP	192.168.1.254 is at 8e:21:ad:4d:d1:d6
8	23.596199	192.168.1.1	192.168.4.1	46	UDP	4001 → 5001 Len=4
9	24.998546	82:b2:71:f3:86:2a	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:01 LA/1 4919 SysN=openflow:1
10	29.998243	82:b2:71:f3:86:2a	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:01 LA/1 4919 SysN=openflow:1
11	34.998309	82:b2:71:f3:86:2a	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:01 LA/1 4919 SysN=openflow:1

Figura 5.2: Captura de tráfico en Wireshark: interfaz de entrada a s1 (hacia la red interna del PoP_1).

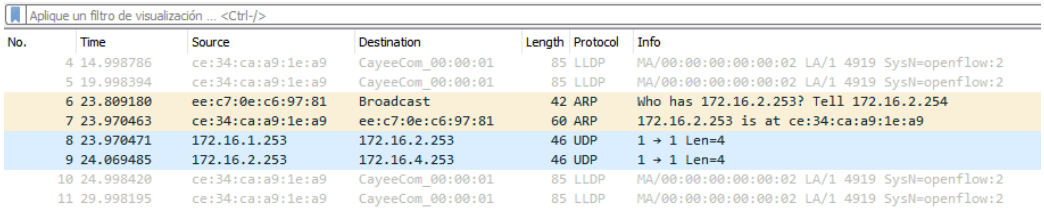
vez, a los puertos de origen y destino se les asigna el Link Id y el ONAT Id respectivamente. En este caso se tiene configurada sólo una ONRP, por lo que todos los links del *path* tomarán el identificador 1. Por otro lado, el ONAT Id también vale 1 dado que es el primer flujo de la ONRP a ser registrado.

Continuando con el camino de la ONRP, en la Figura 5.4 se muestra que el paquete enviado desde el PoP_1 llega al switch s2, es modificado y enviado nuevamente



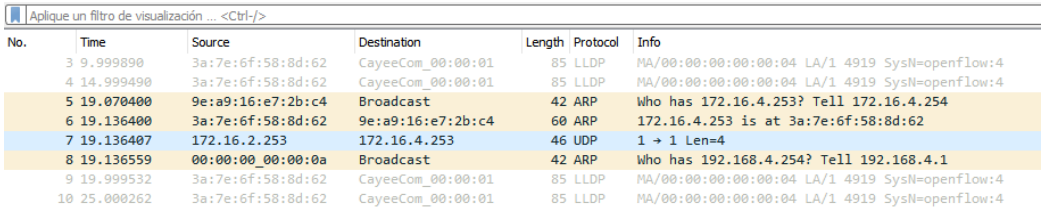
No.	Time	Source	Destination	Length	Protocol	Info
4	14.998927	8e:21:ad:4d:d1:d6	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:01 LA/3 4919 SysN=openflow:1
5	19.998542	8e:21:ad:4d:d1:d6	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:01 LA/3 4919 SysN=openflow:1
6	23.488850	00:00:00_00:00:01	Broadcast	42	ARP	Who has 192.168.1.254? Tell 192.168.1.1
7	23.809531	172.16.1.253	172.16.2.253	46	UDP	1 → 1 Len=4
8	24.998545	8e:21:ad:4d:d1:d6	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:01 LA/3 4919 SysN=openflow:1
9	29.998243	8e:21:ad:4d:d1:d6	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:01 LA/3 4919 SysN=openflow:1

Figura 5.3: Captura de tráfico en Wireshark: interfaz de s1 hacia r1.



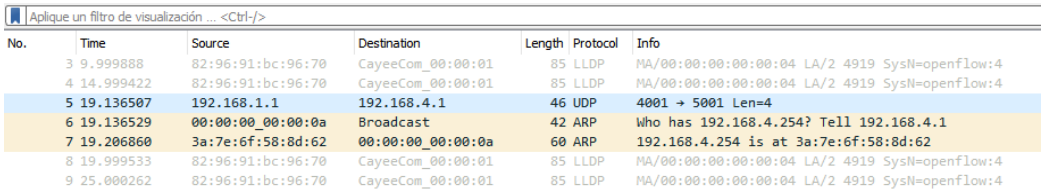
No.	Time	Source	Destination	Length	Protocol	Info
4	14.998786	ce:34:ca:a9:1e:a9	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:02 LA/1 4919 SysN=openflow:2
5	19.998394	ce:34:ca:a9:1e:a9	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:02 LA/1 4919 SysN=openflow:2
6	23.809180	ee:c7:0e:c6:97:81	Broadcast	42	ARP	Who has 172.16.2.253? Tell 172.16.2.254
7	23.970463	ce:34:ca:a9:1e:a9	ee:c7:0e:c6:97:81	60	ARP	172.16.2.253 is at ce:34:ca:a9:1e:a9
8	23.970471	172.16.1.253	172.16.2.253	46	UDP	1 → 1 Len=4
9	24.069485	172.16.2.253	172.16.4.253	46	UDP	1 → 1 Len=4
10	24.998420	ce:34:ca:a9:1e:a9	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:02 LA/1 4919 SysN=openflow:2
11	29.998195	ce:34:ca:a9:1e:a9	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:02 LA/1 4919 SysN=openflow:2

Figura 5.4: Captura de tráfico en Wireshark: interfaz de s2 hacia r2.



No.	Time	Source	Destination	Length	Protocol	Info
3	9.999890	3a:7e:6f:58:8d:62	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:04 LA/1 4919 SysN=openflow:4
4	14.999490	3a:7e:6f:58:8d:62	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:04 LA/1 4919 SysN=openflow:4
5	19.070400	9e:a9:16:e7:2b:c4	Broadcast	42	ARP	Who has 172.16.4.253? Tell 172.16.4.254
6	19.136400	3a:7e:6f:58:8d:62	9e:a9:16:e7:2b:c4	60	ARP	172.16.4.253 is at 3a:7e:6f:58:8d:62
7	19.136407	172.16.2.253	172.16.4.253	46	UDP	1 → 1 Len=4
8	19.136559	00:00:00_00:00:0a	Broadcast	42	ARP	Who has 192.168.4.254? Tell 192.168.4.1
9	19.999532	3a:7e:6f:58:8d:62	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:04 LA/1 4919 SysN=openflow:4
10	25.000262	3a:7e:6f:58:8d:62	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:04 LA/1 4919 SysN=openflow:4

Figura 5.5: Captura de tráfico en Wireshark: interfaz de s4 hacia r4.



No.	Time	Source	Destination	Length	Protocol	Info
3	9.999888	82:96:91:bc:96:70	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:04 LA/2 4919 SysN=openflow:4
4	14.999422	82:96:91:bc:96:70	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:04 LA/2 4919 SysN=openflow:4
5	19.136507	192.168.1.1	192.168.4.1	46	UDP	4001 → 5001 Len=4
6	19.136529	00:00:00_00:00:0a	Broadcast	42	ARP	Who has 192.168.4.254? Tell 192.168.4.1
7	19.206860	3a:7e:6f:58:8d:62	00:00:00_00:00:0a	60	ARP	192.168.4.254 is at 3a:7e:6f:58:8d:62
8	19.999533	82:96:91:bc:96:70	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:04 LA/2 4919 SysN=openflow:4
9	25.000262	82:96:91:bc:96:70	CayeeCom_00:00:01	85	LLDP	MA/00:00:00:00:00:04 LA/2 4919 SysN=openflow:4

Figura 5.6: Captura de tráfico en Wireshark: interfaz de salida de s4 (hacia h41).

hacia r2 para continuar hacia el PoP destino.

El paquete llega a la interfaz del switch s4 conectada a r4, como se puede observar en la Figura 5.5. Finalmente, el paquete es modificado y enviado hacia h41. En la Figura 5.6 se comprueba que el paquete llega al host destino tal como fue enviado desde el origen.

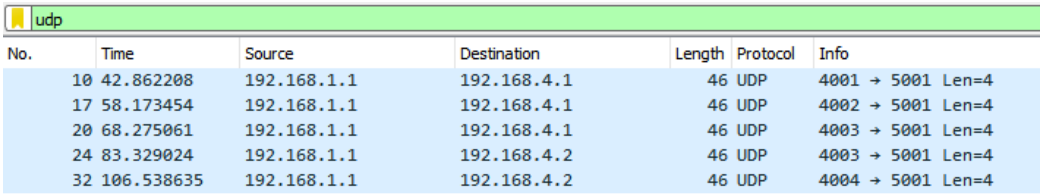
Segunda prueba: identificación de flujos de una misma ONRP

En esta prueba se genera tráfico perteneciente a la ONRP ya implementada con el objetivo de validar la identificación de flujos asociados a una misma política mediante la asignación de ONAT Ids. En la tabla 5.2 se muestra la información de los flujos generados pertenecientes a la ONRP 1.

En la Figura 5.7, se muestran los paquetes que llegan al switch s1 con los parámetros de la tabla 5.2. Todos pertenecen a la ONRP 1 por lo que van a seguir

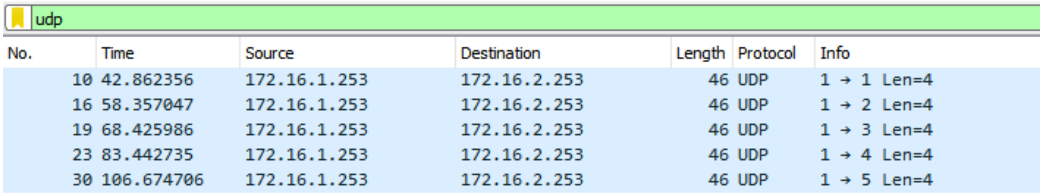
Tabla 5.2: Flujos de prueba pertenecientes a una misma ONRP.

IP Org	IP Dst	Protocolo	Puerto Org	Puerto Dst
192.168.1.1/32	192.168.4.1/32	UDP	4001	5001
192.168.1.1/32	192.168.4.1/32	UDP	4002	5001
192.168.1.1/32	192.168.4.1/32	UDP	4003	5001
192.168.1.1/32	192.168.4.2/32	UDP	4003	5001
192.168.1.1/32	192.168.4.2/32	UDP	4004	5001


udp

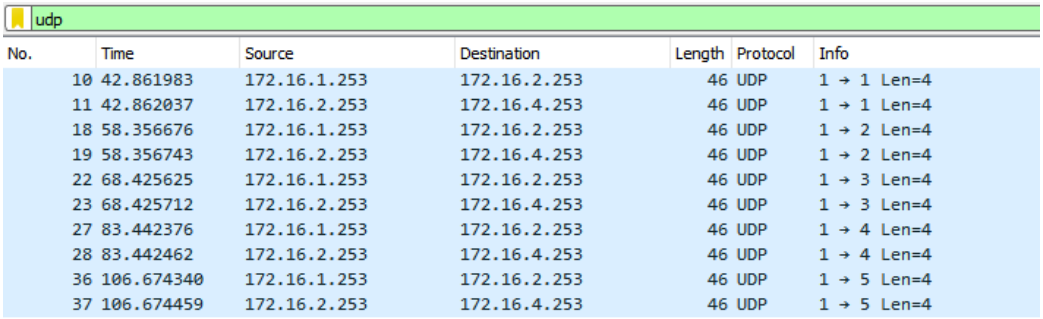
No.	Time	Source	Destination	Length	Protocol	Info
10	42.862208	192.168.1.1	192.168.4.1	46	UDP	4001 → 5001 Len=4
17	58.173454	192.168.1.1	192.168.4.1	46	UDP	4002 → 5001 Len=4
20	68.275061	192.168.1.1	192.168.4.1	46	UDP	4003 → 5001 Len=4
24	83.329024	192.168.1.1	192.168.4.2	46	UDP	4003 → 5001 Len=4
32	106.538635	192.168.1.1	192.168.4.2	46	UDP	4004 → 5001 Len=4

Figura 5.7: Captura de tráfico en Wireshark: interfaz de entrada a s1 (hacia la red interna del PoP_1).


udp

No.	Time	Source	Destination	Length	Protocol	Info
10	42.862356	172.16.1.253	172.16.2.253	46	UDP	1 → 1 Len=4
16	58.357047	172.16.1.253	172.16.2.253	46	UDP	1 → 2 Len=4
19	68.425986	172.16.1.253	172.16.2.253	46	UDP	1 → 3 Len=4
23	83.442735	172.16.1.253	172.16.2.253	46	UDP	1 → 4 Len=4
30	106.674706	172.16.1.253	172.16.2.253	46	UDP	1 → 5 Len=4

Figura 5.8: Captura de tráfico en Wireshark: interfaz de s1 hacia r1.


udp

No.	Time	Source	Destination	Length	Protocol	Info
10	42.861983	172.16.1.253	172.16.2.253	46	UDP	1 → 1 Len=4
11	42.862037	172.16.2.253	172.16.4.253	46	UDP	1 → 1 Len=4
18	58.356676	172.16.1.253	172.16.2.253	46	UDP	1 → 2 Len=4
19	58.356743	172.16.2.253	172.16.4.253	46	UDP	1 → 2 Len=4
22	68.425625	172.16.1.253	172.16.2.253	46	UDP	1 → 3 Len=4
23	68.425712	172.16.2.253	172.16.4.253	46	UDP	1 → 3 Len=4
27	83.442376	172.16.1.253	172.16.2.253	46	UDP	1 → 4 Len=4
28	83.442462	172.16.2.253	172.16.4.253	46	UDP	1 → 4 Len=4
36	106.674340	172.16.1.253	172.16.2.253	46	UDP	1 → 5 Len=4
37	106.674459	172.16.2.253	172.16.4.253	46	UDP	1 → 5 Len=4

Figura 5.9: Captura de tráfico en Wireshark: interfaz de s2 hacia r2.

el *path* [PoP_1,PoP_2,PoP_4]. En la Figura 5.8 se observa como siguen hacia el PoP_2 con los cambios correspondientes en las direcciones y puertos de capa de transporte. En el puerto de origen viaja la información de identificadores locales que en este caso será igual para todos los paquetes dado que pertenecen a la misma ONRP. Los flujos son diferenciados gracias a la asignación de ONAT Ids. Estos identificadores viajan en el puerto destino de capa de transporte. En 5.8 se comprueba que a cada paquete se le asigna un ONAT Id distinto.

udp

No.	Time	Source	Destination	Length	Protocol	Info
10	42.861922	172.16.2.253	172.16.4.253	46	UDP	1 → 1 Len=4
17	58.356628	172.16.2.253	172.16.4.253	46	UDP	1 → 2 Len=4
20	68.425598	172.16.2.253	172.16.4.253	46	UDP	1 → 3 Len=4
24	83.442349	172.16.2.253	172.16.4.253	46	UDP	1 → 4 Len=4
33	106.674351	172.16.2.253	172.16.4.253	46	UDP	1 → 5 Len=4

Figura 5.10: Captura de tráfico en Wireshark: interfaz de s4 hacia r4.

udp

No.	Time	Source	Destination	Length	Protocol	Info
10	42.861980	192.168.1.1	192.168.4.1	46	UDP	4001 → 5001 Len=4
17	58.356692	192.168.1.1	192.168.4.1	46	UDP	4002 → 5001 Len=4
21	68.425662	192.168.1.1	192.168.4.1	46	UDP	4003 → 5001 Len=4
26	83.442412	192.168.1.1	192.168.4.2	46	UDP	4003 → 5001 Len=4
34	106.674437	192.168.1.1	192.168.4.2	46	UDP	4004 → 5001 Len=4

Figura 5.11: Captura de tráfico en Wireshark: interfaz de s4 hacia la red interna del PoP_4.

En la Figura 5.10 se muestra como llegan los paquetes desde r4 hacia el switch s4, siendo diferenciados únicamente por el ONAT Id. Haciendo uso de este identificador, en s4 se puede recuperar la información original del mensaje antes de ser enviado hacia el host destino. En 5.11 se observa que los paquetes llegan a h41 idénticos a como fueron enviados por el host h11.

Tercer prueba: identificación de flujo para múltiples ONRPs

En esta prueba se implementan múltiples ONRPs y se comprueba el correcto encaminamiento del tráfico generado con la utilización de identificadores locales. El uso de estos identificadores permite distinguir el tráfico perteneciente a distintas ONRPs que comparten un mismo link. En la tabla 5.3 se muestran los parámetros de las 5 ONRPs implementadas para esta prueba junto con la prioridad calculada. Los caminos asociados a cada ONRP se ilustran en la Figura 5.12, donde se distinguen mediante el identificador asignado.

Tabla 5.3: ONRPs implementadas para las pruebas de validación.

ONRP Id	1	2	3	4	5
IP origen	192.168.1.0/24	192.168.1.1/32	192.168.1.0/24	192.168.1.0/24	192.168.1.0/24
IP destino	192.168.4.0/24	192.168.4.2/32	192.168.3.0/24	192.168.3.0/24	192.168.4.0/24
Path	[PoP1,PoP2,PoP4]	[PoP1,PoP4]	[PoP1,PoP2,PoP3]	[PoP1,PoP3]	[PoP1,PoP3,PoP4]
Protocolo	-	-	-	-	UDP
Puerto origen	-	-	-	4444	4444
Puerto destino	-	-	-	5555	5555
Prioridad	792	1056	792	6936	7960

Una vez implementadas las políticas, se genera tráfico asociado a cada una de ellas como se observa en la Figura 5.13. En la Figura 5.14 se muestra que los

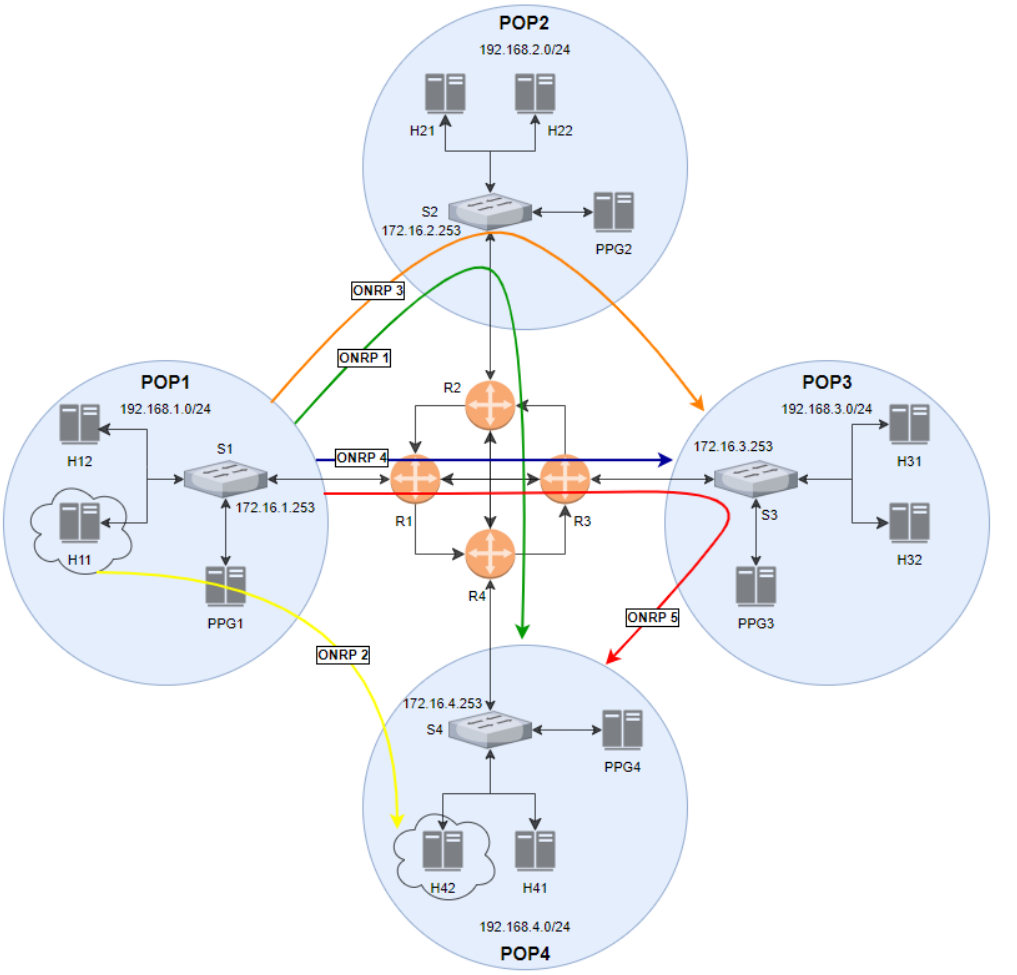
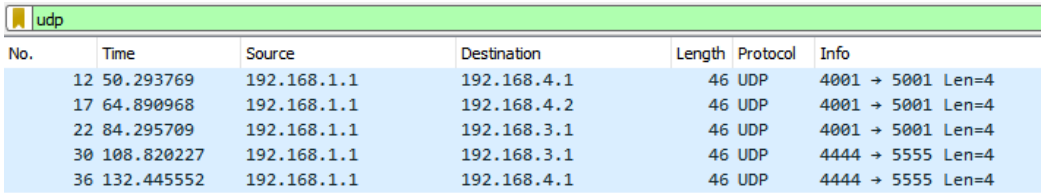


Figura 5.12: Caminos asociados a las ONRPs detalladas en 5.3.

paquetes son modificados y enviados a r_1 para ser encaminados hacia el siguiente PoP.

El primer paquete, asociado a la $ONRP_1$, sale dirigido hacia el PoP_2. A esta ONRP se le asigna el identificador local 1 en el *link* [PoP_1,PoP_2]. El segundo paquete pertenece a la $ONRP_2$ por lo que seguirá el camino directo del PoP_1 al PoP_4. Para este *link* la $ONRP_2$ toma el identificador local 1. El siguiente paquete asociado a la $ONRP_3$ es enviado hacia el PoP_2. Se le asigna el identificador local 2 para el *link* [PoP_1,PoP_2]. Esto se debe a que la $ONRP_1$ ya consumió un identificador para dicho enlace. El paquete asociado a la $ONRP_4$ sigue el camino directo desde el PoP_1 al PoP_3. En este *link* se le asigna el identificador local 1. La $ONRP_5$ comparte el *link* [PoP_1,PoP_3] con la $ONRP_4$. Por lo tanto, el paquete asociado a dicha política sale dirigido hacia el PoP_3 con el identificador local 2.

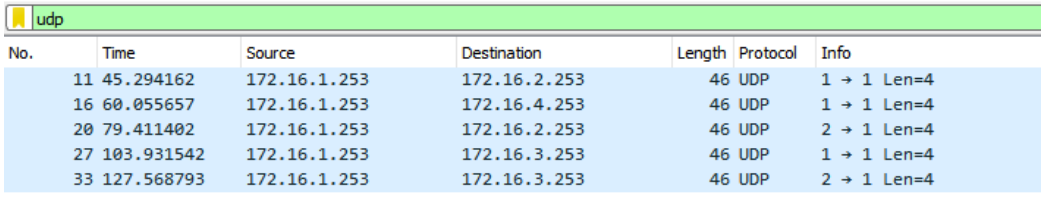
En la Figura 5.15 se observan los paquetes asociados a las políticas 1 y 3 que llegan al switch s_2. Estos son modificados y enviados nuevamente a r_2 para ser dirigidos hacia el PoP_4 y PoP_3 respectivamente.



udp

No.	Time	Source	Destination	Length	Protocol	Info
12	50.293769	192.168.1.1	192.168.4.1	46	UDP	4001 → 5001 Len=4
17	64.890968	192.168.1.1	192.168.4.2	46	UDP	4001 → 5001 Len=4
22	84.295709	192.168.1.1	192.168.3.1	46	UDP	4001 → 5001 Len=4
30	108.820227	192.168.1.1	192.168.3.1	46	UDP	4444 → 5555 Len=4
36	132.445552	192.168.1.1	192.168.4.1	46	UDP	4444 → 5555 Len=4

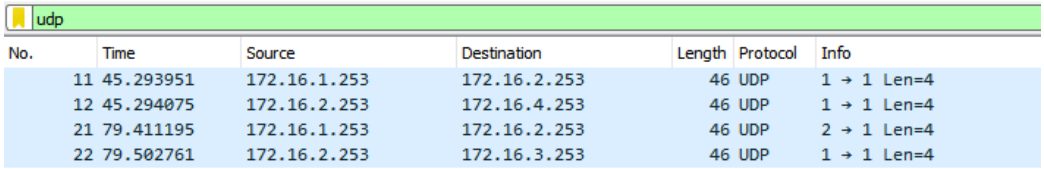
Figura 5.13: Captura de tráfico en Wireshark: interfaz de s1 hacia la red interna del PoP_1.



udp

No.	Time	Source	Destination	Length	Protocol	Info
11	45.294162	172.16.1.253	172.16.2.253	46	UDP	1 → 1 Len=4
16	60.055657	172.16.1.253	172.16.4.253	46	UDP	1 → 1 Len=4
20	79.411402	172.16.1.253	172.16.2.253	46	UDP	2 → 1 Len=4
27	103.931542	172.16.1.253	172.16.3.253	46	UDP	1 → 1 Len=4
33	127.568793	172.16.1.253	172.16.3.253	46	UDP	2 → 1 Len=4

Figura 5.14: Captura de tráfico en Wireshark: interfaz de s1 hacia r1.

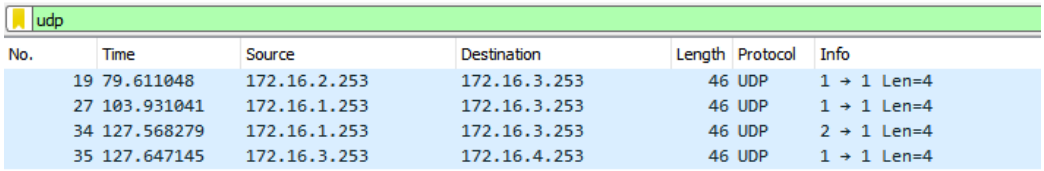


udp

No.	Time	Source	Destination	Length	Protocol	Info
11	45.293951	172.16.1.253	172.16.2.253	46	UDP	1 → 1 Len=4
12	45.294075	172.16.2.253	172.16.4.253	46	UDP	1 → 1 Len=4
21	79.411195	172.16.1.253	172.16.2.253	46	UDP	2 → 1 Len=4
22	79.502761	172.16.2.253	172.16.3.253	46	UDP	1 → 1 Len=4

Figura 5.15: Captura de tráfico en Wireshark: interfaz de s2 hacia r2.

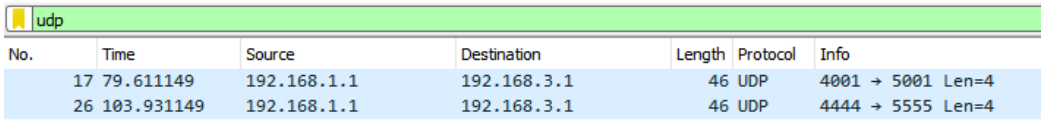
Por la interfaz de s_3 hacia r_3, llegan los paquetes asociados a la $ONRP_3$, $ONRP_4$ y $ONRP_5$ como se muestra en la Figura 5.16. Los primeros dos tienen como destino el PoP_3 por lo que son modificados y enviados hacia el host destino, con la información original, como se observa en la Figura 5.17. Por otro lado, el tercer paquete, asociado a la $ONRP_5$, es modificado y enviado nuevamente a r_3 para seguir el camino hacia el PoP_4.



udp

No.	Time	Source	Destination	Length	Protocol	Info
19	79.611048	172.16.2.253	172.16.3.253	46	UDP	1 → 1 Len=4
27	103.931041	172.16.1.253	172.16.3.253	46	UDP	1 → 1 Len=4
34	127.568279	172.16.1.253	172.16.3.253	46	UDP	2 → 1 Len=4
35	127.647145	172.16.3.253	172.16.4.253	46	UDP	1 → 1 Len=4

Figura 5.16: Captura de tráfico en Wireshark: interfaz de s3 hacia r3.



udp

No.	Time	Source	Destination	Length	Protocol	Info
17	79.611149	192.168.1.1	192.168.3.1	46	UDP	4001 → 5001 Len=4
26	103.931149	192.168.1.1	192.168.3.1	46	UDP	4444 → 5555 Len=4

Figura 5.17: Captura de tráfico en Wireshark: interfaz de s3 hacia la red interna del PoP_3.

En la Figura 5.18, se observa que los paquetes asociados a la $ONRP_1$, $ONRP_2$ y $ONRP_5$ llegan al switch s_4 desde r_4. En este punto, se recupera la información original del mensaje y, como se muestra en la Figura 5.19, los paquetes llegan al destino idénticos a como fueron enviados desde el origen.

udp

No.	Time	Source	Destination	Length	Protocol	Info
11	45.293954	172.16.2.253	172.16.4.253	46	UDP	1 → 1 Len=4
19	60.055325	172.16.1.253	172.16.4.253	46	UDP	1 → 1 Len=4
35	127.647351	172.16.3.253	172.16.4.253	46	UDP	1 → 1 Len=4

Figura 5.18: Captura de tráfico en Wireshark: interfaz de s4 hacia r4.

udp

No.	Time	Source	Destination	Length	Protocol	Info
11	45.294015	192.168.1.1	192.168.4.1	46	UDP	4001 → 5001 Len=4
18	60.055414	192.168.1.1	192.168.4.2	46	UDP	4001 → 5001 Len=4
34	127.647442	192.168.1.1	192.168.4.1	46	UDP	4444 → 5555 Len=4

Figura 5.19: Captura de tráfico en Wireshark: interfaz de s4 hacia la red interna del PoP_4.

5.1.2. MeasureApp

En esta subsección se muestran las pruebas realizadas en Mininet para el módulo de medida, por lo que está fuertemente relacionado con las secciones 3.3.2 y 3.7. Los equipos utilizados para realizar la captura de paquetes son el que aloja a la aplicación (172.16.10.2), el PPG1 (192.168.1.3) perteneciente al PoP_1 y el PPG2 (192.168.2.3) que se encuentra en el PoP_2. También se va a observar la IP 172.16.1.253 que es la del switch OpenFlow del PoP_1.

Primera Prueba: Validación de medida usando *throughput* 1 y *average* 1

En esta prueba los valores de *average* y *throughput* se fijan en 1. Es la prueba básica ya que se utiliza solamente 1 paquete para realizar la medición.

Captura realizada en la aplicación

En la Figura 5.20 se puede observar que el paquete va desde la aplicación (172.16.10.2) hasta el PoP_1 (172.16.1.253) y luego de un tiempo se obtiene la respuesta desde el PoP_1.

No.	Time	Source	Destination	Protocol	Length	Info
73	0.505049	172.16.10.2	172.16.1.253	UDP	77	37163 → 2800 Len=35
204	4.303874	172.16.1.253	172.16.10.2	UDP	57	58416 → 2801 Len=15

Figura 5.20: Captura en la aplicación para la Primera Prueba.

- Paquete Número 73: El mensaje enviado es "Measure/2/40544/192.168.2.3/1/1/100", por lo que se está realizando una solicitud de medida hacia el PPG del PoP_2 (192.168.2.3), con un *average* de 1 y un *throughput*. Además, se puede observar que el ID de la medida es 2.

- Paquete Número 204: Contiene el mensaje "Measure/2/0.112", entonces se puede afirmar que es la respuesta para la medida con el ID 2 y se tuvo como resultado 0.112 segundos.

Captura realizada en el PPG1

La Figura 5.21 muestra los paquetes enviados y recibidos por el PPG del PoP_1, a continuación se detalla los mensajes que contienen:

No.	Time	Source	Destination	Protocol	Length	Info
14	43.989733	172.16.10.2	192.168.1.3	UDP	77	37163 → 2800 Len=35
15	44.716101	192.168.1.3	192.168.2.3	UDP	69	40544 → 2802 Len=27
16	44.741550	192.168.2.3	192.168.1.3	UDP	51	40544 → 2802 Len=9
18	45.735295	192.168.1.3	192.168.2.3	UDP	72	40544 → 2802 Len=30
19	45.836771	192.168.2.3	192.168.1.3	UDP	72	40544 → 2802 Len=30
20	45.922441	192.168.1.3	172.16.10.2	UDP	57	58416 → 2801 Len=15

Figura 5.21: Captura en el PPG1 para la Primera Prueba.

- Paquete Número 14: El mensaje es "Measure/2/40544/192.168.2.3/1/1/100", por lo que se corrobora que se recibió lo enviado por la aplicación (172.16.10.2) en el paquete número 73.

- Paquete Número 15: En este paquete el mensaje enviado al PoP_2 es "MensIni/1/40544/192.168.1.3" que es el de inicialización.

- Paquete Número 16: Esta es la respuesta al paquete número 15, su contenido es "MensIni/2".

- Paquete Número 18: Su carga útil es "RTT/22:20:58.8/2/1/1/1/1/100". En él se puede observar los datos según lo mostrado en la sección 3.7.

- Paquete Número 19: Este es la respuesta al paquete número 18, el mensaje que contiene es "RTT/22:20:58.8/2/1/2/1/1/100". El valor de NEnvios es 2 por lo que no sería necesario enviar un nuevo paquete sino que se debe procesar la medida para ser enviada a la aplicación.

- Paquete Número 20: Es el envío del resultado de la medida, su carga útil es "Measure/2/0.112" lo cual se valida con el paquete número 204 analizado capturado en la aplicación.

Captura realizada en el PPG2

Por último, para verificar el correcto funcionamiento del sistema en este caso se deben analizar las capturas tomadas en el PPG del PoP_2. En él se tienen que observar los mensajes enviados por el PPG del PoP_1 y la respuesta a los mismos. En la Figura 5.22 se muestra la secuencia de paquetes enviados y a continuación se detalla su contenido:

No.	Time	Source	Destination	Protocol	Length	Info
13	39.715504	192.168.1.3	192.168.2.3	UDP	69	40544 → 2802 Len=27
14	39.740732	192.168.2.3	192.168.1.3	UDP	51	40544 → 2802 Len=9
16	40.734702	192.168.1.3	192.168.2.3	UDP	72	40544 → 2802 Len=30
17	40.836056	192.168.2.3	192.168.1.3	UDP	72	40544 → 2802 Len=30

Figura 5.22: Captura en el PPG2 para la Primera Prueba.

- Paquete Número 13: En este se obtiene lo enviado por el PPG1 en el paquete número 15, su contenido es "MensIni/1/40544/192.168.1.3".
- Paquete Número 14: El mensaje que contiene es "MensIni/2" por lo que se corrobora con el paquete número 16 recibido por el PPG1.
- Paquete Número 16: Es el paquete número 18 enviado por el PPG1, contiene "RTT/22:20:58.8/2/1/1/1/1/100".
- Paquete Número 17: La diferencia en el mensaje entre este paquete y el 16 es que se le suma 1 al valor de NEnvios. Su carga es "RTT/22:20:58.8/2/1/2/1/1/100", y coincide con lo visto en el paquete número 19 del PPG1.

Segunda prueba: Validación de medida usando *throughput* 3 y *average* 1

Para realizar esta prueba se mantiene el valor de *average* en 1 y se cambia el valor de *throughput* a 3, por lo que es esperable que para realizar la medida se envíe una ráfaga de tres paquetes.

Captura realizada en la aplicación

La Figura 5.23 muestra la secuencia de paquetes enviados y recibidos por la aplicación, a continuación se analiza el contenido de los paquetes:

No.	Time	Source	Destination	Protocol	Length	Info
80	8.955528	172.16.10.2	172.16.1.253	UDP	77	49737 → 2800 Len=35
213	12.642123	172.16.1.253	172.16.10.2	UDP	73	58820 → 2801 Len=31

Figura 5.23: Captura en la aplicación para la Segunda Prueba.

- Paquete Número 80: El mensaje enviado es "Measure/3/41162/192.168.2.3/3/1/100", por lo que se está realizando una solicitud de medida hacia el PPG del PoP_2 (192.168.2.3), con una *average* de 1 y un *throughput* de 3. Además, se puede

observar que el ID de la medida es 3 entonces la respuesta a esperar es para la medida con ese ID.

- Paquete Número 204: Contiene el mensaje "Measure/3/0.037" que es la respuesta para la medida con el ID 3 y se tiene como resultado 0.037 segundos.

Captura realizada en el PPG1

En esta subsección se analizan los mensajes enviados y recibidos por el PPG1, en la Figura 5.24 se puede observar la secuencia de paquetes enviados que se analizará a continuación.

No.	Time	Source	Destination	Protocol	Length	Info
16	67.381659	172.16.10.2	192.168.1.3	UDP	77	49737 → 2800 Len=35
21	69.961965	192.168.1.3	192.168.2.3	UDP	69	41162 → 2802 Len=27
23	70.052252	192.168.2.3	192.168.1.3	UDP	51	41162 → 2802 Len=9
24	70.967895	192.168.1.3	192.168.2.3	UDP	72	41162 → 2802 Len=30
25	70.968022	192.168.1.3	192.168.2.3	UDP	72	41162 → 2802 Len=30
26	70.968114	192.168.1.3	192.168.2.3	UDP	72	41162 → 2802 Len=30
27	70.975589	192.168.2.3	192.168.1.3	UDP	72	41162 → 2802 Len=30
28	70.975800	192.168.2.3	192.168.1.3	UDP	72	41162 → 2802 Len=30
29	70.975979	192.168.2.3	192.168.1.3	UDP	72	41162 → 2802 Len=30
30	71.067904	192.168.1.3	172.16.10.2	UDP	73	58820 → 2801 Len=31

Figura 5.24: Captura en el PPG1 para la Segunda Prueba.

- Paquete Número 16: Es el mensaje de inicialización enviado por la aplicación, su carga útil es "Measure/3/41162/192.168.2.3/3/1/100" por lo que se valida con lo descrito en el paquete número 80 enviado por la aplicación.

Los paquetes 21 y 23 son los primeros mensajes intercambiados y se envían para generar la creación de las ONRP necesarias y evitar que esto se refleje en el tiempo de medida. Los mensajes siguen la línea de los paquetes 15 y 16 del PPG1 presentados en la primera prueba, por lo que se procederá a analizar el resto de paquetes.

- Paquete Número 24: El contenido es "RTT/22:27:49.0/3/1/1/3/1/100". En él se puede observar que el valor de NPaquete es 1, el NEnvios es 1 y el Throughput es 3.
- Paquete Número 25: Se envía el mensaje "RTT/22:27:49.056/3/1/1/3/2/100". Este contiene los valores de NPaquete, NEnvios y Throughput igual a 2, 1 y 3 respectivamente.
- Paquete Número 26: La carga útil es "RTT/22:27:49.056/3/1/1/3/3/100". En este caso el NPaquete es 3, el NEnvios es 1 y el Throughput es 3.

En los paquetes del 24 al 26 se puede observar que se envía correctamente la ráfaga de paquetes variando el NPaquete de forma correcta, por lo que en los siguientes paquetes se va a mostrar la respuesta a los mismos.

- Paquete Número 27: El contenido es "RTT/22:27:49.0/3/1/2/3/1/100". En él se puede observar que el NPaquete es 1, el NEnvios es 2 y el Throughput es 3.
- Paquete Número 28: Se envía el mensaje "RTT/22:27:49.056/3/1/2/3/2/100". Este contiene los valores de NPaquete, NEnvios y Throughput igual a 2, 2 y 3 respectivamente.
- Paquete Número 29: La carga útil es "RTT/22:27:49.056/3/1/2/3/3/100". En este caso el NPaquete es 3, el NEnvios es 2 y el Throughput es 3.

Finalmente, se ve el contenido del paquete número 30 que es el enviado a la aplicación con el resultado de la medida, y coincide con el paquete número 204 recibido por la aplicación.

- Paquete Número 30: El contenido útil para la medida es "Measure/3/0.037".

Captura realizada en el PPG2

Las capturas son realizadas en el PPG2 y se muestran en la Figura 5.25. A continuación se muestran los mensajes de los paquetes capturados siendo estos las respuestas a los enviados por el PPG1. Al igual que en la subsección anterior los paquetes 16 y 18 no se mostrarán en detalle a causa de que son los mensajes de creación de las entradas dinámicas de las ONRP analizados en la primera prueba.

No.	Time	Source	Destination	Protocol	Length	Info
16	54.962028	192.168.1.3	192.168.2.3	UDP	69	41162 → 2802 Len=27
18	55.052028	192.168.2.3	192.168.1.3	UDP	51	41162 → 2802 Len=9
19	55.967950	192.168.1.3	192.168.2.3	UDP	72	41162 → 2802 Len=30
20	55.968056	192.168.1.3	192.168.2.3	UDP	72	41162 → 2802 Len=30
21	55.968148	192.168.1.3	192.168.2.3	UDP	72	41162 → 2802 Len=30
22	55.975574	192.168.2.3	192.168.1.3	UDP	72	41162 → 2802 Len=30
23	55.975769	192.168.2.3	192.168.1.3	UDP	72	41162 → 2802 Len=30
24	55.975982	192.168.2.3	192.168.1.3	UDP	72	41162 → 2802 Len=30

Figura 5.25: Captura en el PPG2 para la Segunda Prueba.

- Paquete Número 19: El contenido es "RTT/22:27:49.0/3/1/1/3/1/100". Como se puede observar es el paquete número 24 enviado por el PPG1.
- Paquete Número 20: Se recibe el mensaje "RTT/22:27:49.056/3/1/1/3/2/100" que es el paquete número 25 analizado en la subsección 5.1.2.
- Paquete Número 21: La carga útil es "RTT/22:27:49.056/3/1/1/3/3/100", y corresponde paquete número 26 enviado por el PPG1.

Los próximos paquetes son los generados en respuesta a los tres anteriores, se le suma uno a la variable NEnvios y son enviados al PPG1. En los paquetes 27, 28 y 29 de la sección 5.1.2 se puede ver esta respuesta por lo que a continuación solamente se van a mostrar los mensajes enviados:

- Paquete Número 22: El contenido es "RTT/22:27:49.0/3/1/2/3/1/100".
- Paquete Número 23: Se envía el mensaje "RTT/22:27:49.056/3/1/2/3/2/100".
- Paquete Número 24: La carga útil es "RTT/22:27:49.056/3/1/2/3/3/100".

Tercera Prueba: Validación de medida usando *throughput* 1 y *average* 3

En el presente apartado se analizan las pruebas de validación realizadas fijando el valor de *throughput* en 1 y *average* en 3, por lo que se espera que el paquete realice 3 veces el camino indicado para la medida.

Captura realizada en la aplicación

La Figura 5.26 muestra las secuencias de paquetes enviados y recibidos por la aplicación.

No.	Time	Source	Destination	Protocol	Length	Info
71	0.219359	172.16.10.2	172.16.1.253	UDP	77	39160 → 2800 Len=35
206	4.279872	172.16.1.253	172.16.10.2	UDP	57	59970 → 2801 Len=15

Figura 5.26: Captura en la aplicación para la Tercera Prueba.

- Paquete Número 71: El contenido útil es "Measure/4/44310/192.168.2.3/1/3/100". Este mensaje es enviado al PPG1 y la medida se debe realizar a la IP 192.168.2.3 (PPG2). El valor de *throughput* es 1, *average* 3 y el ID de la medida es 4.
- Paquete Número 206: El mensaje capturado es "Measure/4/0.133" por lo que se trata de la respuesta para la medida con ID 4 y su valor es 0.133 segundos.

Captura realizada en el PPG1

Las capturas son mostradas en la Figura 5.27 y se analizarán a continuación.

No.	Time	Source	Destination	Protocol	Length	Info
11	45.161580	172.16.10.2	192.168.1.3	UDP	77	39160 → 2800 Len=35
16	47.798882	192.168.1.3	192.168.2.3	UDP	69	44310 → 2802 Len=27
17	47.905373	192.168.2.3	192.168.1.3	UDP	51	44310 → 2802 Len=9
18	48.811632	192.168.1.3	192.168.2.3	UDP	72	44310 → 2802 Len=30
19	48.926522	192.168.2.3	192.168.1.3	UDP	72	44310 → 2802 Len=30
20	48.969985	192.168.1.3	192.168.2.3	UDP	72	44310 → 2802 Len=30
21	49.048488	192.168.2.3	192.168.1.3	UDP	72	44310 → 2802 Len=30
22	49.090198	192.168.1.3	192.168.2.3	UDP	72	44310 → 2802 Len=30
23	49.202747	192.168.2.3	192.168.1.3	UDP	72	44310 → 2802 Len=30
24	49.221743	192.168.1.3	172.16.10.2	UDP	57	59970 → 2801 Len=15

Figura 5.27: Captura en el PPG1 para la Tercera Prueba.

- Paquete Número 11: Es el paquete número 71 enviado por la aplicación, el mensaje es "Measure/4/44310/192.168.2.3/1/3/100".

Los paquetes 16 y 17 son los enviados para generar las entradas dinámicas en los switches correspondientes y así evitar que estos tiempo impacten en la medición. Estos no se analizarán debido a que fue realizado en secciones anteriores. En la próxima lista se muestran los mensajes de los paquetes del 18 al 23, observar que el campo NEnvios crece hasta 6 por lo que se recorre el camino ida y vuelta 3 veces dando lugar a una medición de *average* 3.

- Paquete Número 18: "RTT/22:32:36.896/4/3/1/1/1/100".
- Paquete Número 19: "RTT/22:32:36.896/4/3/2/1/1/100".
- Paquete Número 20: "RTT/22:32:36.896/4/3/3/1/1/100".
- Paquete Número 21: "RTT/22:32:36.896/4/3/4/1/1/100".
- Paquete Número 22: "RTT/22:32:36.896/4/3/5/1/1/100".
- Paquete Número 23: "RTT/22:32:36.896/4/3/6/1/1/100".

Finalmente, el paquete que se muestra a continuación es el que se envía a la aplicación.

- Paquete Número 18: El mensaje es "Measure/4/0.133" que coincide con el paquete número 206 recibido por la aplicación. Se envía el resultado de medida para el ID 4.

Con el fin de no resultar repetitivo no se muestran los mensajes capturados en el PPG2 debido a que al igual que en las pruebas anteriores son los mismos que los analizados en el PPG1 descontando la comunicación con la aplicación.

Cuarta Prueba: Validación de tiempo de medida

Para esta prueba, utilizando los comando de Mininet se fija un retardo de 0.8 segundos en el enlace entre router 1 y 2. Se espera recibir una medida con un resultado mayor a dos veces el retardo impuesto a causa de que la medida es realizada ida y vuelta y se tiene el retardo del resto de los enlaces. Se corrobora capturando los paquetes enviados y recibidos por la aplicación, en la Figura 5.28.

No.	Time	Source	Destination	Protocol	Length	Info
71	0.347058	172.16.10.2	172.16.1.253	UDP	77	53661 → 2800 Len=35
278	7.346584	172.16.1.253	172.16.10.2	UDP	57	50647 → 2801 Len=15
349	44.628205	172.16.10.2	172.16.1.253	UDP	77	52983 → 2800 Len=35
534	51.739699	172.16.1.253	172.16.10.2	UDP	57	34371 → 2801 Len=15

Figura 5.28: Captura en la aplicación para la Cuarta Prueba.

- Paquete Numero 71: "Measure/1/43135/192.168.2.3/1/1/100"

- Paquete Numero 71: “Measure/1/1.702”
- Paquete Numero 349: “Measure/2/42001/192.168.2.3/1/1/100”
- Paquete Numero 534: “Measure/2/1.709”

Por el contenido de los paquetes se puede observar que se envían dos medidas con el ID 1 y 2, y sus resultados de medida fueron 1.702 s y 1.709 s respectivamente.

Quinta Prueba: Falla en la medida

En esta sección se fuerza la falla en la medida al no ejecutar el software de medida en el PPG2. Se muestran las capturas del tráfico en la aplicación y el PPG1.

Captura realizada en la aplicación

En la Figura 5.29 se puede observar la secuencia de envío y recepción de paquetes, y a continuación se describe el mensaje que contiene cada uno de ellos.

No.	Time	Source	Destination	Protocol	Length	Info
79	120.726986	172.16.10.2	172.16.1.253	UDP	77	35754 → 2800 Len=35
282	148.995135	172.16.1.253	172.16.10.2	UDP	58	47219 → 2801 Len=16

Figura 5.29: Captura en la aplicación para la Quinta Prueba.

- Paquete Número 79: “Measure/1/43259/192.168.2.3/1/1/100”. Es un mensaje de inicialización de medida enviado al PPG1 con destino de medida el PPG2, con ID de medida igual a 1.
- Paquete Número 282: “Measure/1/Failed”. Este mensaje señala que la medida con el ID 1 falló.

Captura en el PPG1

En la Figura 5.30 se puede observar la secuencia de paquetes enviados. El mensaje que contiene cada uno de ellos es:

- Paquete Número 18: “Measure/1/43259/192.168.2.3/1/1/100”.
- Paquete Número 24: “MensIni/1/43259/192.168.1.3”.
- Paquete Número 25: “RTT/01:41:25.585/1/1/1/1/1/100”.
- Paquete Número 29: “MensIni/1/43259/192.168.1.3”.
- Paquete Número 30: “RTT/01:41:31.148/1/1/1/1/1/100”.
- Paquete Número 34: “MensIni/1/43259/192.168.1.3”.

No.	Time	Source	Destination	Protocol	Length	Info
18	85.048199	172.16.10.2	192.168.1.3	UDP	77	35754 → 2800 Len=35
24	87.611724	192.168.1.3	192.168.2.3	UDP	69	43259 → 2802 Len=27
25	88.645570	192.168.1.3	192.168.2.3	UDP	72	43259 → 2802 Len=30
29	93.188281	192.168.1.3	192.168.2.3	UDP	69	43259 → 2802 Len=27
30	94.208377	192.168.1.3	192.168.2.3	UDP	72	43259 → 2802 Len=30
34	98.788784	192.168.1.3	192.168.2.3	UDP	69	43259 → 2802 Len=27
35	99.847283	192.168.1.3	192.168.2.3	UDP	72	43259 → 2802 Len=30
39	104.444022	192.168.1.3	192.168.2.3	UDP	69	43259 → 2802 Len=27
41	105.460478	192.168.1.3	192.168.2.3	UDP	72	43259 → 2802 Len=30
45	109.012278	192.168.1.3	192.168.2.3	UDP	69	43259 → 2802 Len=27
48	111.058572	192.168.1.3	192.168.2.3	UDP	72	43259 → 2802 Len=30
49	113.079140	192.168.1.3	172.16.10.2	UDP	58	47219 → 2801 Len=16

Figura 5.30: Captura en el PPG1 para la Quinta Prueba.

- Paquete Número 35: “RTT/01:41:36.786/1/1/1/1/1/100”.
- Paquete Número 39: “MensIni/1/43259/192.168.1.3”.
- Paquete Número 41: “RTT/01:41:42.400/1/1/1/1/1/100”.
- Paquete Número 45: “MensIni/1/43259/192.168.1.3”.
- Paquete Número 48: “RTT/01:41:47.998/1/1/1/1/1/100”.
- Paquete Número 49: “Measure/1/Failed”.

Como se puede observar, una vez que le llega la solicitud de medida, el paquete de inicialización y medida se retransmiten hasta llegar al número máximo de retransmisión. Luego, al no obtener respuesta, en el paquete 49 se envía el mensaje de falla a la aplicación.

5.1.3. TEApp

El objetivo de esta prueba es validar el algoritmo de ingeniería de tráfico que implica el funcionamiento de la arquitectura completa. Como se explicó en la sección 3.3, *TEApp* funciona en conjunto con *MonApp* que indica en cada instante las rutas a medir en base al costo asociado a la medición y el estado de la red. A su vez, solicita la ejecución de *MeasureApp* y *RouteApp* según corresponda.

El escenario de esta prueba se muestra en la Figura 5.31. Se seleccionan como origen y destino el PoP_1 y PoP_2 respectivamente, y se registran tres $PoPs$ en la red. De esta forma se trabaja con dos posibles rutas, la directa y la indirecta con un salto en el PoP_3.

Por otro lado, se modifica la topología en Mininet para añadir *delay* a los enlaces $R_1 - R_2$ y $R_3 - R_2$. Se configura un *delay* variable en $R_1 - R_2$ de forma que la ruta directa se encuentre en nivel bajo durante la mayor parte del tiempo, y que cambie a nivel alto añadiendo un *delay* de $500ms$ durante $15s$. Dado que en Mininet no se observan diferencias considerables entre el retardo de la ruta directa e indirecta, se añade un *delay* fijo de $100ms$ al enlace $R_3 - R_2$ de modo que en nivel bajo las rutas se diferencien.

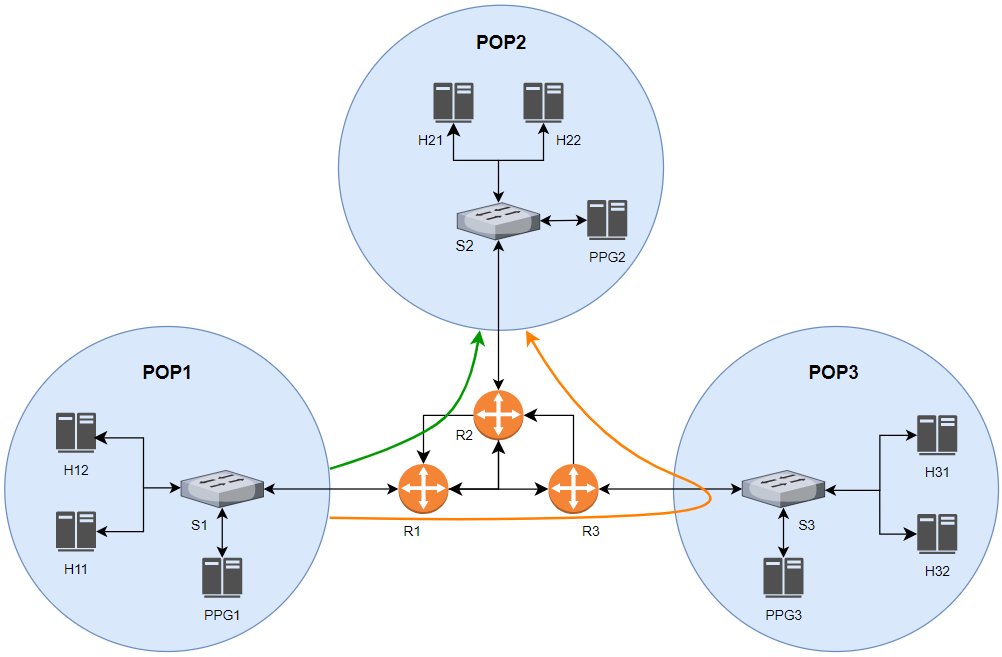


Figura 5.31: Escenario de pruebas de TEApp en Mininet.

Se excluye de esta prueba la validación de *MonApp*, que se utiliza asumiendo su correcto funcionamiento en base a lo desarrollado en [26]. Por ende, para ejecutar esta prueba se fija el costo de medición en cero de forma que en cada iteración *MonApp* solicite medir todas las rutas. Con esto se pretende enfocar la prueba en validar que la medición sea consecuente con los cambios de *delay*, y que se modifique correctamente la ONRP si se produce un cambio en la mejor ruta entre el origen y destino.

Se configura el algoritmo de forma que realice el monitoreo de la red y la decisión de ruteo cada 1 minuto, y que inicie asumiendo que la ruta vigente entre el PoP_1 y PoP_2 es la directa.

Los resultados obtenidos se encuentran en las figuras 5.32 y 5.33. Como puede verse, la ruta elegida en todos los casos es la que tiene el RTT más bajo en ese instante de decisión. En los casos en que la mejor ruta cambia respecto al instante anterior, se realiza una modificación en la política de ruteo que corresponde al flujo de tráfico entre el PoP_1 y PoP_2 para cambiar el *Path*. La primera línea en la Figura 5.32 corresponde a la etapa de inicialización del algoritmo, como fue explicado en la sección 3.3.3, donde se miden las rutas y determina el estado inicial.

Se observa que los resultados de la medición están acorde a lo esperado, obteniendo para la ruta directa un RTT en torno a los $580ms$ cuando el *delay* añadido es de $500ms$, mientras que se encuentra alrededor de los $100ms$ cuando no se añade *delay*.

Para validar la modificación de las rutas se registró la ONRP vigente entre el PoP_1 y el PoP_2 en instantes de tiempo consecutivos. En la Figura 5.33 se observa

Tiempo (min)	RTT ruta directa (seg)	RTT ruta indirecta (seg)	Ruta elegida
0	0.074	0.318	-
1	0.619	**0.351**	Indirecta
2	**0.066**	0.350	Directa
3	**0.087**	0.248	Directa
4	0.598	**0.302**	Indirecta
5	**0.116**	0.351	Directa
6	**0.149**	0.420	Directa
7	0.558	**0.279**	Indirecta
8	**0.122**	0.355	Directa
9	**0.086**	0.261	Directa
10	0.594	**0.325**	Indirecta
11	**0.071**	0.306	Directa
12	**0.108**	0.300	Directa
13	0.527	**0.261**	Indirecta
14	**0.111**	0.279	Directa
15	**0.089**	0.299	Directa
16	**0.071**	0.319	Directa
17	**0.023**	0.278	Directa
18	**0.181**	0.246	Directa
19	0.545	**0.227**	Indirecta
20	**0.081**	0.226	Directa
21	**0.095**	0.264	Directa
22	**0.041**	0.361	Directa
23	**0.033**	0.272	Directa
24	**0.142**	0.300	Directa
25	0.631	**0.231**	Indirecta
26	**0.231**	0.339	Directa
27	**0.165**	0.369	Directa
28	**0.162**	0.292	Directa
29	**0.093**	0.262	Directa
30	**0.057**	0.271	Directa
31	**0.122**	0.366	Directa
32	0.552	**0.338**	Indirecta
33	**0.067**	0.241	Directa
34	0.560	**0.310**	Indirecta
35	**0.124**	0.361	Directa
36	**0.109**	0.237	Directa
37	0.606	**0.276**	Indirecta

Figura 5.32: RTT medido en ruta directa e indirecta junto con la ruta elegida para cada instante de tiempo.

que en el instante de $13min$ se encuentra vigente la ONRP indirecta, y en el siguiente instante se determina que la mejor ruta es la directa por lo que se solicita la modificación a *RouteApp*. En la tabla 5.4, se observan las características de la

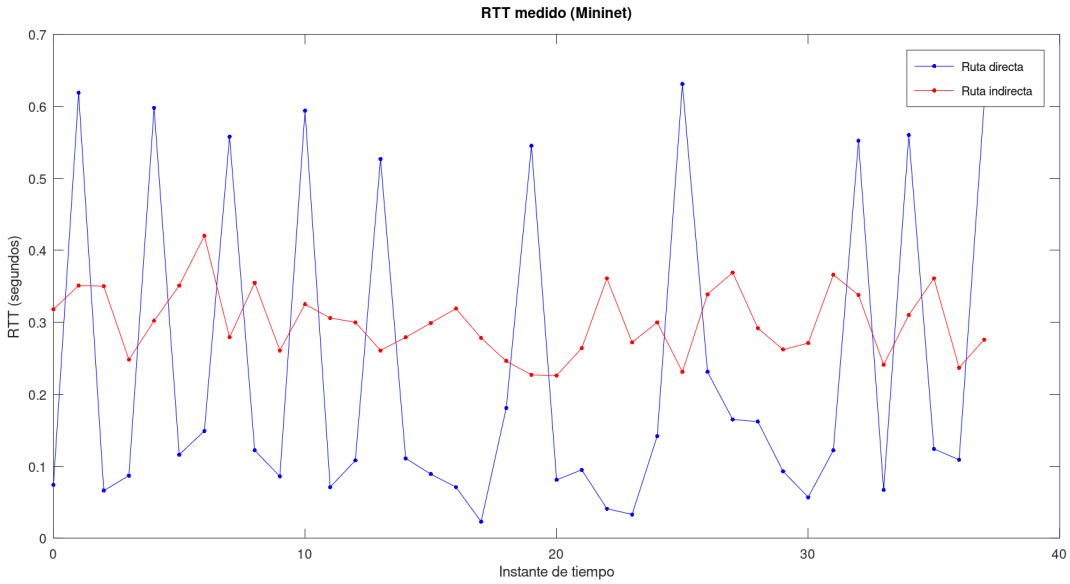


Figura 5.33: RTT medido en ruta directa e indirecta.

Tabla 5.4: ONRP vigente entre el PoP_1 y PoP_2 en distintos instantes de tiempo.

t (min)	ONRP Id	Prioridad	Link Id	IP Ori	IP Dest	Path	Protocolo	Puerto Ori	Puerto Dest
13	68	792	[20, 20]	192.168.1.0/24	192.168.2.0/24	[PoP_1,PoP_3,PoP_2]	-	-	-
14	73	792	[21]	192.168.1.0/24	192.168.2.0/24	[PoP_1,PoP_2]	-	-	-
19	94	792	[27, 27]	192.168.1.0/24	192.168.2.0/24	[PoP_1,PoP_3,PoP_2]	-	-	-

ONRP antes y después de dicha modificación. Se verifica el cambio del *path* al directo y la asignación de nuevos ONRP Id y Link Id. En los siguientes cuatro instantes de tiempo, se mantiene la ruta directa como la más rápida según los resultados de la medición, y en el instante de tiempo de $19min$ la ruta elegida es la indirecta dado que presenta un menor retardo. En la tabla 5.4 se observa que la ONRP nuevamente es modificada para que el tráfico entre el PoP_1 y PoP_2 siga el camino indirecto a través del PoP_3 y se vuelven a asignar los identificadores correspondientes.

5.2. Pruebas en maqueta de laboratorio

5.2.1. RouteApp

Para la validación de *RouteApp* en la maqueta se trabajó con las ONRPs detalladas en la tabla 5.5. Se trata de las rutas directa e indirecta entre el PoP_1 y PoP_2 como se muestra en la Figura 5.34.

En primer lugar, se creó la ONRP correspondiente a la ruta indirecta entre los $PoPs$ mencionados con un salto en el PoP_3. El objetivo es generar un flujo de datos entre el PPG1 y el PPG2, y verificar que sea encaminado por la ruta indirecta configurada. Las características del flujo generado se exhiben en la tabla 5.6.

Tabla 5.5: ONRPs implementadas para las pruebas de *RouteApp* en maqueta de laboratorio.

IP Ori	IP Dest	Path	Protocolo	Puerto Ori	Puerto Dest
172.16.1.0/24	172.16.2.0/24	[1, 2]	-	-	-
172.16.1.0/24	172.16.2.0/24	[1, 3, 2]	-	-	-

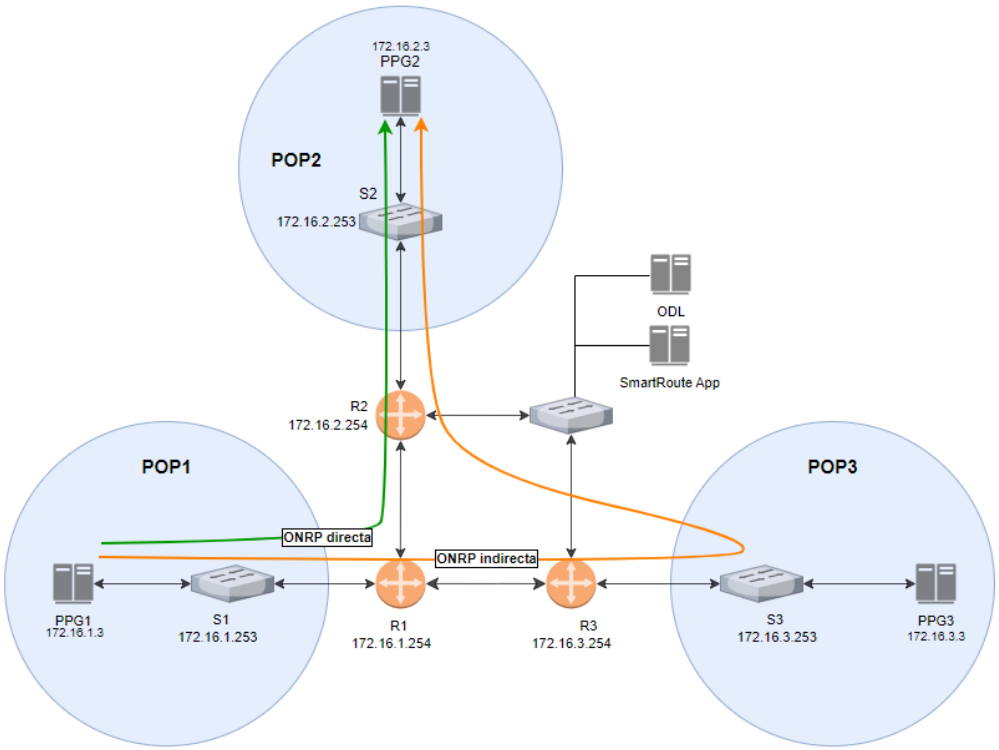


Figura 5.34: Caminos asociados a las ONRPs generadas para las pruebas de *RouteApp* en maqueta de laboratorio.

Tabla 5.6: Flujo de prueba para la validación de *RouteApp* en maqueta de laboratorio.

IP Ori	IP Dest	Path	Puerto Ori	Puerto Dest
172.16.1.3/32	172.16.2.3/32	UDP	45333	2806

En la Figura 5.35 se muestran los paquetes enviados desde el PPG1 con las características mencionadas dirigido hacia el PPG2. Luego, en la Figura 5.36 se observan los paquetes con los campos modificados según indica el algoritmo de encaminamiento. Las direcciones son las asignadas a los switches s1 y s3 respectivamente, por lo que se verifica el salto intermedio a través del PoP_3. Por otro lado, los puertos son modificados por el Link Id y ONAT Id respectivamente. Esta es la primera ONRP configurada por lo que a los links del *path* ([PoP_1,PoP_3] y [PoP_3,PoP_2]) se les asigna el identificador 1. El ONAT Id también vale 1 dado que se trata del primer flujo de la ONRP a ser identificado y encaminado.

Continuando con el camino de la ONRP, en la Figura 5.37 se muestra la captura de tráfico en r3. Allí se observan los paquetes que llegan desde s1 con destino s3, y cómo son reenviados con las direcciones modificadas por las de s3 y s2 respectivamente. Estos paquetes llegan a r2 como se muestra en la Figura 5.38 y finalmente son recibidos por el PPG2. En la Figura 5.39 se observa que los paquetes llegan al destino con la información original del mensaje recuperada.

Aplique un filtro de visualización ... <Ctrl-/>

No.	Time	Source	Destination	Length	Protocol	Info
1	0.000000000	Tp-LinkT_f5:bb:ae	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:bb:ae IN/eth0.5 120 SysD=openvswitch 2.11.6
2	3.779157408	Clevo_2d:40:64	LLDP_Multicast	194	LLDP	MA/80:fa:5b:2d:40:64 MA/80:fa:5b:2d:40:64 120 SysN=ceibal
3	3.963732797	172.16.1.3	172.16.2.3	68	UDP	45333 → 2806 Len=26
4	4.936759952	Tp-LinkT_f5:bb:ae	CayeeCom_00:00:01	101	LLDP	MA/00:00:11:11:11:11 LA/3 4919 SysN=openflow:286331153
5	5.000351482	Tp-LinkT_f5:bb:ae	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:bb:ae IN/eth0.5 120 SysD=openvswitch 2.11.6
6	8.975074937	Clevo_2d:40:64	Normerel_11:11:11	42	ARP	Who has 172.16.1.253? Tell 172.16.1.3
7	9.819297916	Normerel_11:11:11	Clevo_2d:40:64	60	ARP	172.16.1.253 is at 00:00:11:11:11:11
8	9.935674738	Tp-LinkT_f5:bb:ae	CayeeCom_00:00:01	101	LLDP	MA/00:00:11:11:11:11 LA/3 4919 SysN=openflow:286331153
9	10.001063402	Tp-LinkT_f5:bb:ae	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:bb:ae IN/eth0.5 120 SysD=openvswitch 2.11.6
10	14.935596207	Tp-LinkT_f5:bb:ae	CayeeCom_00:00:01	101	LLDP	MA/00:00:11:11:11:11 LA/3 4919 SysN=openflow:286331153
11	15.001195777	Tp-LinkT_f5:bb:ae	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:bb:ae IN/eth0.5 120 SysD=openvswitch 2.11.6
12	18.610929863	172.16.1.3	172.16.2.3	68	UDP	45333 → 2806 Len=26
13	19.935728844	Tp-LinkT_f5:bb:ae	CayeeCom_00:00:01	101	LLDP	MA/00:00:11:11:11:11 LA/3 4919 SysN=openflow:286331153

Figura 5.35: Captura de tráfico en PPG1: flujo generado con las características de la tabla 5.6 a ser encaminado por la ruta indirecta.

udp

No.	Time	Source	Destination	Length	Protocol	Info
37	7.050450	172.16.1.253	172.16.3.253	68	UDP	1 → 1 Len=26
38	7.050521	172.16.1.253	172.16.3.253	68	UDP	1 → 1 Len=26
51	9.779589	172.16.1.253	172.16.3.253	68	UDP	1 → 1 Len=26
52	9.779666	172.16.1.253	172.16.3.253	68	UDP	1 → 1 Len=26

Figura 5.36: Captura de tráfico en r1 asociado a la ruta indirecta.

Siguiendo con las pruebas, se procede a crear la ONRP correspondiente a la ruta directa entre los *PoPs* mencionados y a generar tráfico con las mismas características que en el caso anterior (ver tabla 5.6). El objetivo es verificar que en este caso los paquetes son encaminados por la ruta directa configurada.

udp and !dns

No.	Time	Source	Destination	Length	Protocol	Info
58	5.180700	172.16.1.253	172.16.3.253	68	UDP	1 → 1 Len=26
59	5.180779	172.16.1.253	172.16.3.253	68	UDP	1 → 1 Len=26
60	5.182149	172.16.3.253	172.16.2.253	68	UDP	1 → 1 Len=26
61	5.182209	172.16.3.253	172.16.2.253	68	UDP	1 → 1 Len=26
89	7.909832	172.16.1.253	172.16.3.253	68	UDP	1 → 1 Len=26
90	7.909901	172.16.1.253	172.16.3.253	68	UDP	1 → 1 Len=26
91	7.910093	172.16.3.253	172.16.2.253	68	UDP	1 → 1 Len=26
92	7.910146	172.16.3.253	172.16.2.253	68	UDP	1 → 1 Len=26

Figura 5.37: Captura de tráfico en r3 asociado a la ruta indirecta.

udp

No.	Time	Source	Destination	Length	Protocol	Info
29	3.409701	172.16.3.253	172.16.2.253	68	UDP	1 → 1 Len=26
30	3.409772	172.16.3.253	172.16.2.253	68	UDP	1 → 1 Len=26
48	6.137632	172.16.3.253	172.16.2.253	68	UDP	1 → 1 Len=26
49	6.137702	172.16.3.253	172.16.2.253	68	UDP	1 → 1 Len=26

Figura 5.38: Captura de tráfico en r2 asociado a la ruta indirecta.

Aplique un filtro de visualización ... <Ctrl-/>

No.	Time	Source	Destination	Length	Protocol	Info
2	1.731907651	Tp-LinkT_f5:d6:b2	CayeeCom_00:00:01	101	LLDP	MA/00:00:22:22:22:22 LA/2 4919 SysN=openflow:572662306
3	5.001251799	Tp-LinkT_f5:d6:b2	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:d6:b2 IN/eth0.5 120 SysD=openvswitch 2.11.6
4	5.765914907	172.16.1.3	172.16.2.3	68	UDP	45333 → 2806 Len=26
5	6.732059380	Tp-LinkT_f5:d6:b2	CayeeCom_00:00:01	101	LLDP	MA/00:00:22:22:22:22 LA/2 4919 SysN=openflow:572662306
6	10.001765464	Tp-LinkT_f5:d6:b2	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:d6:b2 IN/eth0.5 120 SysD=openvswitch 2.11.6
7	11.731965826	Tp-LinkT_f5:d6:b2	CayeeCom_00:00:01	101	LLDP	MA/00:00:22:22:22:22 LA/2 4919 SysN=openflow:572662306
8	15.002875121	Tp-LinkT_f5:d6:b2	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:d6:b2 IN/eth0.5 120 SysD=openvswitch 2.11.6
9	16.732128130	Tp-LinkT_f5:d6:b2	CayeeCom_00:00:01	101	LLDP	MA/00:00:22:22:22:22 LA/2 4919 SysN=openflow:572662306
10	17.464089168	Clevo_2d:39:49	LLDP_Multicast	194	LLDP	MA/80:fa:5b:2d:39:49 MA/80:fa:5b:2d:39:49 120 SysN=ceibal
11	20.003141991	Tp-LinkT_f5:d6:b2	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:d6:b2 IN/eth0.5 120 SysD=openvswitch 2.11.6
12	20.413061317	172.16.1.3	172.16.2.3	68	UDP	45333 → 2806 Len=26
13	21.732060040	Tp-LinkT_f5:d6:b2	CayeeCom_00:00:01	101	LLDP	MA/00:00:22:22:22:22 LA/2 4919 SysN=openflow:572662306

Figura 5.39: Captura de tráfico en PPG2: flujo recibido asociado a la ruta indirecta.

En la Figura 5.40 se muestran los paquetes enviados desde el PPG1 con destino el PPG2, y en la Figura 5.41 se observa que llegan a r1 con los campos modificados adecuadamente. Como origen se asigna la dirección de s1 pero en este caso se verifica que el destino es la dirección asignada a s2, por lo que los paquetes seguirán el camino directo entre el PoP_1 y el PoP_2 como era esperado.

Finalmente, son recibidos por r1 como se muestra en la Figura 5.42 y llegan al PPG2 con la información original inalterada como se muestra en la Figura 5.43.

Aplique un filtro de visualización ... <Ctrl-/>

No.	Time	Source	Destination	Length	Protocol	Info
1	0.000000000	Tp-LinkT_f5:bb:ae	CayeeCom_00:00:01	101	LLDP	MA/00:00:11:11:11:11 LA/3 4919 SysN=openflow:286331153
2	3.971757208	172.16.1.3	172.16.2.3	66	UDP	45333 → 2806 Len=24
3	4.448396621	Tp-LinkT_f5:bb:ae	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:bb:ae IN/eth0.5 120 SysD=openvswitch 2.11.6
4	4.883009529	Clevo_2d:40:64	LLDP_Multicast	194	LLDP	MA/80:fa:5b:2d:40:64 MA/80:fa:5b:2d:40:64 120 SysN=ceibal
5	4.999315311	Tp-LinkT_f5:bb:ae	CayeeCom_00:00:01	101	LLDP	MA/00:00:11:11:11:11 LA/3 4919 SysN=openflow:286331153
6	8.979739739	Clevo_2d:40:64	Normerel_11:11:11	42	ARP	Who has 172.16.1.253? Tell 172.16.1.3
7	9.448582235	Tp-LinkT_f5:bb:ae	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:bb:ae IN/eth0.5 120 SysD=openvswitch 2.11.6
8	9.602417326	Normerel_11:11:11	Clevo_2d:40:64	60	ARP	172.16.1.253 is at 00:00:11:11:11:11
9	9.998812278	Tp-LinkT_f5:bb:ae	CayeeCom_00:00:01	101	LLDP	MA/00:00:11:11:11:11 LA/3 4919 SysN=openflow:286331153
10	10.169511867	172.16.1.3	172.16.2.3	66	UDP	45333 → 2806 Len=24
11	14.448801989	Tp-LinkT_f5:bb:ae	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:bb:ae IN/eth0.5 120 SysD=openvswitch 2.11.6

Figura 5.40: Captura de tráfico en PPG1: flujo generado con las características de la tabla 5.6 a ser encaminado por la ruta directa.

udp

No.	Time	Source	Destination	Length	Protocol	Info
13	13.318887	172.16.1.253	172.16.2.253	66	UDP	8 → 1 Len=24
14	13.318957	172.16.1.253	172.16.2.253	66	UDP	8 → 1 Len=24
17	17.821104	172.16.1.253	172.16.2.253	66	UDP	8 → 1 Len=24
18	17.821175	172.16.1.253	172.16.2.253	66	UDP	8 → 1 Len=24

Figura 5.41: Captura de tráfico en r1 asociado a la ruta directa.

udp

No.	Time	Source	Destination	Length	Protocol	Info
15	11.102286	172.16.1.253	172.16.2.253	66	UDP	8 → 1 Len=24
16	11.102363	172.16.1.253	172.16.2.253	66	UDP	8 → 1 Len=24
20	15.604474	172.16.1.253	172.16.2.253	66	UDP	8 → 1 Len=24
21	15.604550	172.16.1.253	172.16.2.253	66	UDP	8 → 1 Len=24

Figura 5.42: Captura de tráfico en r2 asociado a la ruta directa.

Aplique un filtro de visualización ... <Ctrl-/>

No.	Time	Source	Destination	Length	Protocol	Info
3	5.000945927	Tp-LinkT_f5:d6:b2	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:d6:b2 IN/eth0.5 120 SysD=openvswitch 2.11.6
4	6.103373644	Tp-LinkT_f5:d6:b2	CayeeCom_00:00:01	101	LLDP	MA/00:00:22:22:22:22 LA/2 4919 SysN=openflow:572662306
5	6.775575887	172.16.1.3	172.16.2.3	66	UDP	45333 → 2806 Len=24
6	10.000443236	Tp-LinkT_f5:d6:b2	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:d6:b2 IN/eth0.5 120 SysD=openvswitch 2.11.6
7	11.103026945	Tp-LinkT_f5:d6:b2	CayeeCom_00:00:01	101	LLDP	MA/00:00:22:22:22:22 LA/2 4919 SysN=openflow:572662306
8	11.783077849	Clevo_2d:39:49	VisualTe_22:22:22	42	ARP	Who has 172.16.2.253? Tell 172.16.2.3
9	12.687066680	VisualTe_22:22:22	Clevo_2d:39:49	60	ARP	172.16.2.253 is at 00:00:22:22:22:22
10	15.001169482	Tp-LinkT_f5:d6:b2	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:d6:b2 IN/eth0.5 120 SysD=openvswitch 2.11.6
11	16.103187412	Tp-LinkT_f5:d6:b2	CayeeCom_00:00:01	101	LLDP	MA/00:00:22:22:22:22 LA/2 4919 SysN=openflow:572662306
12	17.742308876	Clevo_2d:39:49	LLDP_Multicast	194	LLDP	MA/80:fa:5b:2d:39:49 MA/80:fa:5b:2d:39:49 120 SysN=ceibal
13	18.971243368	172.16.1.3	172.16.2.3	66	UDP	45333 → 2806 Len=24
14	20.000744156	Tp-LinkT_f5:d6:b2	LLDP_Multicast	116	LLDP	MA/90:f6:52:f5:d6:b2 IN/eth0.5 120 SysD=openvswitch 2.11.6

Figura 5.43: Captura de tráfico en PPG2: flujo recibido asociado a la ruta directa.

5.2.2. MeasureApp

Las capturas realizadas para validar el funcionamiento se realizaron en el PPG1 (172.16.1.3) perteneciente al PoP_1, PPG2 (172.16.2.3) perteneciente al PoP_2 y el equipo donde se aloja la aplicación (10.10.0.20). Además, en las capturas se puede visualizar la dirección IP 172.16.1.253 que es la del switch perteneciente al PoP_1.

Primera Prueba: Validación de medida usando *throughput* 1 y *average 1*

Esta prueba se realiza fijando los valores de *throughtput* y *average* en 1, y se ejecuta entre el PoP 1 y 2 utilizando el camino directo.

Captura realizada en la aplicación

En esta captura y como se puede observar en la Figura 5.44 se obtienen dos mensajes, uno hacia el PoP_1 y otro desde el PoP_1. A continuación se analizará cada uno de ellos.

No.	Time	Source	Destination	Protocol	Length	Info
55	24.718596	10.10.0.20	172.16.1.253	UDP	76	59862 → 2803 Len=34
64	31.507981	172.16.1.253	10.10.0.20	UDP	60	41562 → 2802 Len=15

Figura 5.44: Captura en la aplicación de la primera prueba.

- Paquete Número 55: Este es el paquete de inicialización de medida. El mensaje es "Measure/2/41933/172.16.2.3/1/1/120" por lo que el ID de medida es 1, throughtput y average en 1, tamaño del paque en 120.
- Paquete Número 64: Su carga útil es "Measure/2/0.517", entonces se puede afirmar que es el resultado de la medida con el ID 2.

Captura realizada en el PPG1

Los paquetes capturados se pueden visualizar en la Figura 5.45, y a continuación se va a analizar cada uno de ellos.

No.	Time	Source	Destination	Protocol	Length	Info
12	20.835121054	10.10.0.20	172.16.1.3	UDP	76	59862 → 2803 Len=34
14	24.049921735	172.16.1.3	172.16.2.3	UDP	68	41933 → 2801 Len=26
16	24.913929532	172.16.2.3	172.16.1.3	UDP	60	41933 → 2801 Len=9
22	27.076049042	172.16.1.3	172.16.2.3	UDP	72	41933 → 2801 Len=30
24	27.521720347	172.16.2.3	172.16.1.3	UDP	72	41933 → 2801 Len=30
25	27.621088104	172.16.1.3	10.10.0.20	UDP	57	41562 → 2802 Len=15

Figura 5.45: Captura en el PPG1 de la prueba 1.

- Paquete Número 12: Es el paquete de inicialización recibido desde la aplicación, su contenido útil para la implementación es "Measure/2/41933/172.16.2.3/1/1/120". Esto coincide con lo visto en el paquete numero 55 en la subsección anterior.
- Paquete Número 14: Es el generado para la creación de las entradas dinámicas de los switches OpenFlow. El mensaje que contiene es "MensIni/1/41933/172.16.1.3".
- Paquete Número 16: Es la respuesta al paquete descrito en el punto anterior, su carga útil es "MensIni/2".
- Paquete Número 22: El mensaje que contiene es "RTT/18:03:35.565/2/1/1/1/1/120" y es utilizado para la realización de la medida.
- Paquete Número 24: Es la respuesta al paquete número 22, en el se puede ver el mensaje "RTT/18:03:35.565/2/1/2/1/1/120". El valor de NEnvios es 2 por lo que se deja de enviar paquetes y se comienza a realizar el cálculo de la medida.
- Paquete Número 25: En él se puede ver "Measure/2/0.517", por lo que es la respuesta a la solicitud de medida con el ID 2. Esto se corrobora en el paquete número 64 capturado en la aplicación.

Captura realizada en el PPG2

Esta captura se puede observar en la Figura 5.46 realizada con el fin de validar el intercambio de mensajes entre los PPG. Debido a que en el PPG1 ya se analizaron los mensajes intercambiados en esta subsección solamente se menciona el contenido de los paquetes.

No.	Time	Source	Destination	Protocol	Length	Info
15	31.567685675	172.16.1.3	172.16.2.3	UDP	68	41933 → 2801 Len=26
16	32.785874695	172.16.2.3	172.16.1.3	UDP	51	41933 → 2801 Len=9
22	36.399328967	172.16.1.3	172.16.2.3	UDP	72	41933 → 2801 Len=30
24	36.843493197	172.16.2.3	172.16.1.3	UDP	72	41933 → 2801 Len=30

Figura 5.46: Captura en el PPG2 de la prueba 1.

- Paquete Número 15: "MensIni/1/41933/172.16.1.3".
- Paquete Número 16: "MensIni/2".
- Paquete Número 22: "RTT/18:03:35.565/2/1/1/1/1/120".
- Paquete Número 24: "RTT/18:03:35.565/2/1/2/1/1/120".

Segunda Prueba: Validación de medida usando *throughput* 2 y *average* 1

La segunda prueba consiste en cambiar el valor de *throughput* a 2 y mantener el *average* en 1. Se realizaron capturas de paquetes en los PPG involucrados en la medida y la aplicación.

Captura realizada en la aplicación

En la Figura 5.48 se pueden ver los paquetes capturados en la aplicación. El paquete Número 81 es enviado al PoP_1 para iniciar la medida, mientras que el 97 es la respuesta y contiene el valor de la medida. A continuación se indica los mensajes que contienen cada uno de los paquetes.

No.	Time	Source	Destination	Protocol	Length	Info
23	45.769438249	10.10.0.20	172.16.1.3	UDP	76	52209 → 2803 Len=34
24	45.817225003	172.16.1.3	172.16.2.3	UDP	68	44155 → 2801 Len=26
25	48.638897599	172.16.2.3	172.16.1.3	UDP	60	44155 → 2801 Len=9
36	52.457832713	172.16.1.3	172.16.2.3	UDP	72	44155 → 2801 Len=30
37	52.458167187	172.16.1.3	172.16.2.3	UDP	72	44155 → 2801 Len=30
38	52.569661112	172.16.2.3	172.16.1.3	UDP	72	44155 → 2801 Len=30
39	52.579855681	172.16.2.3	172.16.1.3	UDP	72	44155 → 2801 Len=30
40	52.615739973	172.16.1.3	10.10.0.20	UDP	57	36364 → 2802 Len=15

Figura 5.47: Captura en la aplicación de la segunda prueba.

- Paquete Número 81: "Measure/3/44155/172.16.2.3/2/1/120".
- Paquete Número 92: "Measure/3/0.058".

Captura realizada en el PPG1

En esta subsección se analizan los mensajes enviados y recibidos por el PPG1, en la Figura 5.48 se puede observar la secuencia de paquetes enviados que se analizará a continuación. Los paquetes 24 y 25 son los primeros mensajes intercambiados por los PPG, generados para crear las entradas dinámicas en los switches

No.	Time	Source	Destination	Protocol	Length	Info
23	45.769438249	10.10.0.20	172.16.1.3	UDP	76	52209 → 2803 Len=34
24	45.817225003	172.16.1.3	172.16.2.3	UDP	68	44155 → 2801 Len=26
25	48.638897599	172.16.2.3	172.16.1.3	UDP	60	44155 → 2801 Len=9
36	52.457832713	172.16.1.3	172.16.2.3	UDP	72	44155 → 2801 Len=30
37	52.458167187	172.16.1.3	172.16.2.3	UDP	72	44155 → 2801 Len=30
38	52.569661112	172.16.2.3	172.16.1.3	UDP	72	44155 → 2801 Len=30
39	52.579855681	172.16.2.3	172.16.1.3	UDP	72	44155 → 2801 Len=30
40	52.615739973	172.16.1.3	10.10.0.20	UDP	57	36364 → 2802 Len=15

Figura 5.48: Captura en el PPG1 de la segunda prueba.

y así evitar que ese tiempo se refleje en la medida. Si bien aparecen en la captura para evitar mencionar nuevamente lo mismo se van a excluir del análisis.

- Paquete Número 23: Es el mensaje de inicialización enviado por la aplicación, su carga útil es "Measure/3/44155/172.16.2.3/2/1/120".
- Paquete Número 36: El contenido es "RTT/18:06:55.958/3/1/1/2/1/120". En él se puede observar que el valor de NPaquete es 1, NEnvios es 1 y el Throughput es 2.
- Paquete Número 37: Se envía el mensaje "RTT/18:06:55.958/3/1/1/2/2/120". Este contiene los valores de NPaquete, NEnvios y Throughput igual a 2, 1 y 2 respectivamente.

Los paquetes del 36 y 37 son enviados en ráfagas. A continuación se hace mención a la respuesta de estos mensajes en donde se puede ver que el valor de NPaquete se mantiene inalterado y el de NEnvios aumenta.

- Paquete Número 38: El contenido es "RTT/18:06:55.958/3/1/2/2/1/120". En él se puede observar que el valor de NPaquete es 1, el de NEnvios es 2 y el Throughput es 2.
- Paquete Número 39: Se envía el mensaje "RTT/18:06:55.958/3/1/2/2/2/120". Este contiene los valores de NPaquete, NEnvios y Throughput igual a 2, 2 y 2 respectivamente.

El último paquete es el enviado a la aplicación, y como se exhibe a continuación posee los mismos datos que el paquete número 92 recibido por la aplicación.

- Paquete Número 30: El contenido útil para la medida es "Measure/3/0.058".

Captura realizada en el PPG2

Por último, para la validación de esta prueba se muestra en la Figura 5.49 los mensajes captados en el PPG2. Como se muestra a continuación se trata de los mensajes enviados por el PPG1 y la respuesta correspondiente. Al igual que en el PPG1 no serán mostrados los paquetes de creación de dinámicas.

- Paquete Número 36: "RTT/18:06:55.958/3/1/1/2/1/120".

No.	Time	Source	Destination	Protocol	Length	Info
33	61.628722279	172.16.1.3	172.16.2.3	UDP	68	44155 → 2801 Len=26
34	61.659393613	172.16.2.3	172.16.1.3	UDP	51	44155 → 2801 Len=9
35	63.020070475	172.16.1.3	172.16.2.3	UDP	72	44155 → 2801 Len=30
36	63.020133090	172.16.1.3	172.16.2.3	UDP	72	44155 → 2801 Len=30
38	63.130353326	172.16.2.3	172.16.1.3	UDP	72	44155 → 2801 Len=30
39	63.140886367	172.16.2.3	172.16.1.3	UDP	72	44155 → 2801 Len=30

Figura 5.49: Captura en el PPG2 de la segunda prueba.

- Paquete Número 37: "RTT/18:06:55.958/3/1/1/2/2/120".
- Paquete Número 38: "RTT/18:06:55.958/3/1/2/2/1/120".
- Paquete Número 39: "RTT/18:06:55.958/3/1/2/2/2/120".

Tercera Prueba: Validación de medida usando *throughput* 1 y *average* 2

La tercera prueba consiste en fijar el *throughput* 1 y *average* en 2, por lo que el paquete de medida va a recorrer dos veces el camino directo entre el PoP_1 y el PoP_2.

Captura realizada en la aplicación

En la Figura 5.50 se muestra la secuencia de paquetes capturados en la aplicación y a continuación se exhiben los mensajes contenidos en ellos.

No.	Time	Source	Destination	Protocol	Length	Info
56	42.103140	10.10.0.20	172.16.1.253	UDP	76	49195 → 2803 Len=34
68	50.008856	172.16.1.253	10.10.0.20	UDP	60	34626 → 2802 Len=15

Figura 5.50: Captura en la aplicación de la tercera prueba.

- Paquete Número 56: "Measure/7/40358/172.16.2.3/1/2/120"
- Paquete Número 68: "Measure/7/0.277"

El paquete 56 hace referencia a que la medida con el ID 7 se va a realizar al PPG2, con un throughput 1 y average 2. Mientras que el paquete 68 corresponde al resultado de la medida solicitada.

Captura realizada en el PPG1

En este apartado se analizarán los paquetes capturados en el PPG1. En la Figura 5.51 se visualiza la captura y a continuación se muestran los mensajes contenidos en dichos paquetes.

- Paquete Número 10: "Measure/7/40358/172.16.2.3/1/2/120". Es el paquete enviado por la aplicación para la inicialización de la medida.

No.	Time	Source	Destination	Protocol	Length	Info
10	16.979920786	10.10.0.20	172.16.1.3	UDP	76	49195 → 2803 Len=34
19	23.300877826	172.16.1.3	172.16.2.3	UDP	68	40358 → 2801 Len=26
20	23.417443517	172.16.2.3	172.16.1.3	UDP	60	40358 → 2801 Len=9
21	24.317165263	172.16.1.3	172.16.2.3	UDP	72	40358 → 2801 Len=30
22	24.771193740	172.16.2.3	172.16.1.3	UDP	72	40358 → 2801 Len=30
23	24.824257303	172.16.1.3	172.16.2.3	UDP	72	40358 → 2801 Len=30
24	24.853475441	172.16.2.3	172.16.1.3	UDP	72	40358 → 2801 Len=30
25	24.882253889	172.16.1.3	10.10.0.20	UDP	57	34626 → 2802 Len=15

Figura 5.51: Captura en el PPG1 de la tercera prueba.

- Paquete Número 19: “MeasureIni/1/40358/172.16.1.3”.
- Paquete Número 20: “MensIni/2”.

Los paquetes 19 y 20 son para que la aplicación genere las entradas dinámicas en los switches OpenFlow y así evitar que este tiempo impacte en la medida.

- Paquete Número 21: “RTT/18:12:47.835/7/2/1/1/1/120”.
- Paquete Número 22: “RTT/18:12:47.835/7/2/2/1/1/120”.
- Paquete Número 23: “RTT/18:12:47.835/7/2/3/1/1/120”.
- Paquete Número 24: “RTT/18:12:47.835/7/2/4/1/1/120”.

Los paquetes 21, 22, 23 y 24 representan los dos pasajes que hace la medida entre el PoP_1 y el PoP_2. Se puede observar que el valor de NEnvios sube hasta 4 lo que afirma lo mencionado.

- Paquete Número 25: “Measure/7/0.277”

Captura realizada en el PPG2

A fin de validar lo recibido en el PPG2, en la Figura 5.52 se muestra la secuencia de paquetes recibida y a continuación se describe el mensaje que contiene cada uno de ellos. No serán analizados a causa de que se realizó en el PPG1.

No.	Time	Source	Destination	Protocol	Length	Info
22	38.870857437	172.16.1.3	172.16.2.3	UDP	68	40358 → 2801 Len=26
23	38.985966057	172.16.2.3	172.16.1.3	UDP	51	40358 → 2801 Len=9
24	39.887298696	172.16.1.3	172.16.2.3	UDP	72	40358 → 2801 Len=30
26	40.339526756	172.16.2.3	172.16.1.3	UDP	72	40358 → 2801 Len=30
27	40.394168460	172.16.1.3	172.16.2.3	UDP	72	40358 → 2801 Len=30
28	40.421884867	172.16.2.3	172.16.1.3	UDP	72	40358 → 2801 Len=30

Figura 5.52: Captura en el PPG2 de la tercera prueba.

- Paquete Número 22: “MeasureIni/1/40358/172.16.1.3”.
- Paquete Número 23: “MensIni/2”.

- Paquete Número 24: “RTT/18:12:47.835/7/2/1/1/1/120”.
- Paquete Número 26: “RTT/18:12:47.835/7/2/2/1/1/120”.
- Paquete Número 27: “RTT/18:12:47.835/7/2/3/1/1/120”.
- Paquete Número 28: “RTT/18:12:47.835/7/2/4/1/1/120”.

5.2.3. TEApp

Para esta prueba, se parte de la base de que el algoritmo utilizado para decidir cuándo medir y cómo estimar ya fue probado y funciona de forma aceptable para el escenario que se considera (dos rutas con dos niveles posibles de RTT cada una).

Se activa TEApp indicándole que debe tomar una decisión de ruteo cada 60 segundos y se le asigna a la medida un costo cero ya que se desea que siempre se midan ambas rutas, para comprobar que las medidas sean estables y la elección sea correcta. Las medidas son realizadas entre el PoP_1 y el PoP_2, y se tiene una ruta directa (PoP_1-PoP_2) y una indirecta (PoP_1-PoP_3-PoP_2) que además pasa por un switch. Para la prueba, el parámetro *average* se configuró en 1, y *throughput* en 4. Se le indica al sistema que al momento del comienzo la ruta por la que se están enviando los paquetes es la indirecta porque se conoce que es la más lenta y se desea observar que al medir y encontrar que la directa es más rápida se elija realizar el cambio, lo cual sucede.

En la Figura 5.53 se observa en cada instante de decisión los valores de RTT medidos para cada ruta, y en la Figura 5.54 puede verse la misma información gráficamente. Como puede verse, para este caso particular la ruta directa siempre es más rápida, y se encontró que siempre fue la elegida por el sistema para realizar el ruteo.

Tiempo (min)	RTT ruta directa (seg)	RTT ruta indirecta (seg)
0	0.010	0.470
1	0.034	0.252
2	0.069	0.518
3	0.058	0.489
4	0.063	0.298
5	0.127	0.528
6	0.151	0.313
7	0.031	0.326
8	0.057	0.264
9	0.139	0.578
10	0.017	0.428
11	0.058	0.527
12	0.057	0.540
13	0.137	0.390
14	0.062	0.470
15	0.017	0.382
16	0.043	0.244
17	0.195	0.480
18	0.156	0.528
19	0.117	0.225
20	0.038	0.400
21	0.026	0.285
22	0.051	0.410

Figura 5.53: Medidas de RTT para dos rutas.

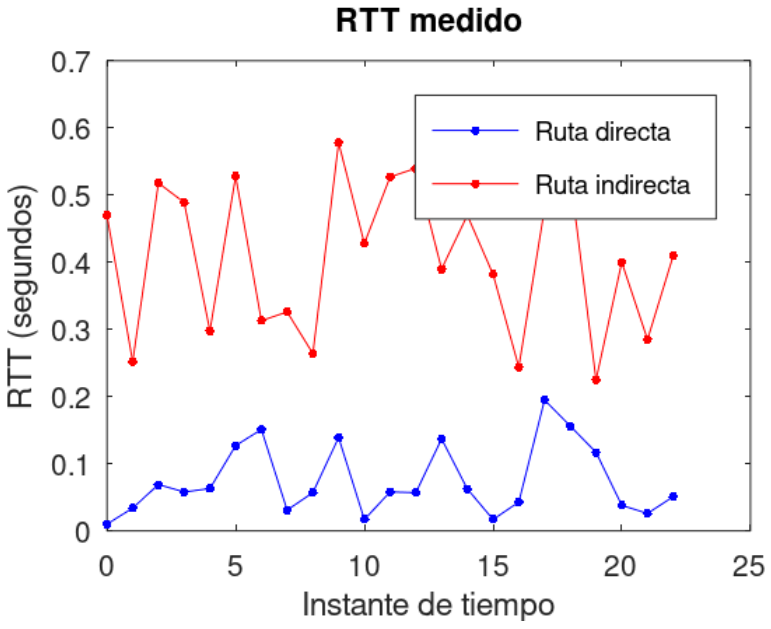


Figura 5.54: RTT de ruta directa e indirecta.

En el presente capitulo se exhibieron las pruebas operativas realizadas para la validación del funcionamiento de SmartRoute y los resultados obtenidos. En el siguiente capitulo se presentarán las conclusiones y se plantearán los trabajos a futuro.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Conclusiones y trabajo a futuro

6.1. Conclusiones

En este trabajo se presentó el estudio realizado para la implementación de una arquitectura basada en SDN que optimiza el ruteo de paquetes, como una alternativa a las redes tradicionales que realizan sus decisiones de ruteo utilizando BGP como se explicó en el capítulo 1. Para ello, se desarrolló la aplicación *SmartRoute* compuesta por módulos encargados de llevar a cabo las diferentes tareas necesarias. Se trabajó en base a la solución del proyecto de fin de carrera [21] que está basada en el controlador ONOS, y se integró con lo desarrollado en [26] para implementar la arquitectura completa usando el controlador OpenDaylight. La arquitectura completa refiere a los módulos *RouteApp*, *MeasureApp*, y *TEApp* junto con *MonApp*.

El módulo *MeasureApp* es encargado de recibir solicitudes de medidas de RTT para diferentes rutas, realizar la medición con los parámetros solicitados (*average, throughput* y *packet size*) y devolver el cálculo de la medida correspondiente. Además, durante la medición, mantiene actualizada la información sobre el estado de la medida de forma que el ente externo que la solicita sea capaz de consultarlo.

El módulo *RouteApp* es el responsable de la gestión de los flujos de datos, permitiendo establecer políticas de encaminamiento personalizadas en el plano de datos. Recibe solicitudes de creación de políticas de ruteo según un formato establecido, y configura como corresponde los switches OpenFlow involucrados. Permite al ente externo, ya sea un usuario u otra aplicación, modificar políticas de ruteo existentes lo que optimiza el proceso de creación de la política y gestión de las *flow entries* asociadas, dado que puede crearse un número acotado de entradas en las *flow tables* de los switches.

TEApp es responsable de la ingeniería de tráfico, trabajando en conjunto con *MonApp* para el monitoreo de la red y tomando las decisiones de ruteo. Esto implica decidir en qué momento medir cada ruta, y realizar las estimaciones correspondientes cuando se decide no medir, intentando minimizar la cantidad de medidas sin que la incertidumbre sobre el estado de las rutas crezca de forma que pueda afectar la calidad de servicio. Interactúa con los dos módulos anteriores,

solicitando en cada momento las medidas necesarias, y luego informando a *RouteApp* sobre las decisiones tomadas para que pueda mantener las políticas de ruteo actualizadas según el estado de la red.

Respecto al uso del controlador OpenDaylight, se detectó una limitante al momento de realizar el procesamiento de paquetes, en particular la detección de mensajes del tipo $PACKET\_IN$. El controlador no brinda la posibilidad de consultar estos mensajes a través de su API, lo que representa un problema significante dado que la detección y procesamiento de este tipo de paquetes es esencial para el correcto funcionamiento de la aplicación tal como fue diseñada. Esto motivó la creación e integración al controlador de dos módulos relacionados al procesamiento de paquetes. En primer lugar, el módulo *PacketinListener* encargado de la detección de mensajes $PACKET\_IN$ en el controlador, y el módulo *MyPacket* que permite enviar un paquete desde el controlador a través de los switches OpenFlow. En conjunto, dichos módulos permiten que la clasificación de las *flow tables* en los switches presentada en la sección 3.5 cumpla su objetivo, y que el tratamiento de paquetes descrito en la sección 3.6 pueda realizarse.

En las pruebas de validación realizadas se observó el correcto funcionamiento de los módulos desarrollados. Dependiendo de las necesidades de la red, el administrador es capaz de establecer distintas ONRPs y realizar medidas de caminos cuando lo crea conveniente.

Se logró implementar una herramienta capaz de otorgarle al usuario la capacidad de aplicar ingeniería de tráfico. Las pruebas de funcionamiento de la herramienta implicaron realizar medidas, modificar y crear ONRPs, por lo que resultaron en la validación la aplicación *SmartRoute* completa. La aplicación desarrollada se encuentra disponible en el repositorio [15].

6.2. Trabajo a futuro

En esta sección se detallan las potenciales mejoras y líneas de investigación que podrían incorporarse al proyecto presentado en este documento.

6.2.1. *SmartRoute del futuro*

En caso de que se genere una nueva versión de la aplicación se cree que es importante tener en cuenta los siguientes aspectos.

Optimización

Teniendo en cuenta que nuestra rama de estudio no se centra en la programación, se cree que existe margen para la optimización del código respecto a la velocidad de acceso a los datos y el consumo de memoria y CPU.

Seguridad

En el desarrollo de la aplicación no se consideró ningún aspecto de seguridad informática, por lo que es indispensable realizarlo en futuras implementaciones. En particular se deberá emplear métodos de cifrado y autenticación para la comunicación entre la aplicación y los PPG y entre PPG distintos.

Interfaz de comunicación con el usuario

La interfaz de comunicación que tiene el usuario para interactuar con la aplicación es mediante línea de comandos. Además de esto, para obtener una mayor escalabilidad sería interesante desarrollar una API para el manejo de la aplicación.

Procesamiento de mensajes ARP

En la versión probada en maqueta, los switches OpenFlow pertenecientes a cada PoP cuentan con una sola IP. Esto se debe a que cuando se asignan dos IP las respuestas a los mensajes ARP no llegan al host que realizó la solicitud. Se diseñaron pruebas y se pudo observar que los mensajes son procesados por la aplicación y enviados al switch correspondiente. Como trabajo a futuro se plantea lograr el funcionamiento del procesamiento de mensajes ARP con dos IP.

6.2.2. Futuras líneas de investigación

A continuación se presentan las posibles líneas a investigar en un futuro, tomando como partida la solución propuesta en este trabajo.

- Probar la solución completa en una red real:
  Se propone realizar pruebas en una red donde intervenga Internet. Se podrán evaluar soluciones como Google Cloud, Amazon Cloud o eventualmente tener puntos con absoluto control, por ejemplo puntos ubicados en diferentes universidades.

- Explorar algoritmos de aprendizaje automático:
  Para las implementaciones desarrolladas en [26], se podrá complementar el trabajo explorando otros algoritmos de aprendizaje automático.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice A

Instalación y comandos principales de Mininet y ODL

A.1. ODL

En la presente subsección se exhibe la forma de instalar ODL y los comandos principales.

A.1.1. Instalación de ODL

A continuación se presenta el paso a paso para la instalación de ODL utilizando el sistema operativo Ubuntu.

- Se descarga la versión que se requiera de ODL del enlace [6].
- Se abre la terminal y se navega hasta el directorio donde se encuentra la versión descargada.
- Con el comando "unzip <archivo.zip>"se descomprime el archivo descargado.
- Se ingresa al directorio descomprimido y luego se escribe el comando "./bin/karaf"para ejecutar ODL.

A.1.2. Comandos

En este apartado se describen los comandos utilizados con frecuencia durante el proyecto en la consola de ODL.

- "feauture:install <feature>": Se ejecuta para instalar una feature.
- "feature:list -i": Con este comando se listan todas las features instaladas.
- "logout": Se ejecuta para dejar de correr el controladro ODL.

A.2. Mininet

En esta subsección se describe el método para instalar Mininet y los comandos básicos necesarios al utilizar Ubuntu como sistema operativo.

A.2.1. Instalación

En Ubuntu 20.04 la forma de instalarlo y poder elegir la versión es ejecutando los siguientes comandos [2].

- "sudo git clone `https://github.com/mininet/mininet`": Se descarga el repositorio de Mininet.
- "cd mininet": Se ingresa al directorio mininet.
- "git tag": Se lista las versiones disponibles de Mininet.
- "sudo git checkout -b <Mininet Version>": Con este comando se selecciona la versión de Mininet que se quiere instalar.
- "cd ..": Se vuelve al escritorio raíz.
- "mininet/util/install.sh [options]": Este comando se ejecuta para instalar la versión de Mininet elegida. En [options] se utiliza -a para instalar el paquete completo.

A.2.2. Comandos

En la siguiente lista se presentan los comandos utilizados frecuentemente en este proyecto.

- "sudo -E python3 <ArchivoPython>": Al ejecutar este comando en Ubuntu se genera la topología programada en python (<ArchivoPython>). Con la -E se permite utilizar X11 forwarding para abrir xterm en los dispositivos pertenecientes a la topología.
- "sudo mn -c": Es ejecutado en la terminal de Ubuntu para finalizar y liberar los recursos utilizados por topologías que estan en ejecución.
- "pingall": Se ejecuta desde la consola de Mininet para realizar ping entre todos los equipos de la red.
- "xterm <dispositivo>": Se abre la terminal del dispositivo, es ejecutado dentro de la CLI de Mininet.

Apéndice B

Desarrollo de módulos en ODL

El presente apéndice es una guía para la creación y despliegue de módulos sobre el controlador OpenDaylight. Es de utilidad para añadir funcionalidades específicas al controlador. En este trabajo se desplegaron los módulos *PacketinListener* y *MyPacket* para la detección de $PACKET\_IN$ y envío de paquetes respectivamente. El siguiente desarrollo está hecho en base a la versión Oxygen de OpenDaylight, que fue la elegida para trabajar en este proyecto.

B.1. Aspectos generales

A continuación se describen los requisitos previos y puesta a punto del entorno de desarrollo para la creación de módulos en ODL.
Lo primero es instalar JDK8, para lo que se usa el siguiente comando:

```
sudo apt-get install openjdk-8-jdk
```

La siguiente herramienta necesaria es Maven, la cual permite generar la estructura de proyecto que conformará un módulo y simplifica el proceso de build. Para la instalación de Maven se ejecuta el siguiente comando:

```
sudo apt install maven
```

Finalizada la instalación de Maven, es necesario configurar adecuadamente un archivo settings.xml. Una forma sencilla de obtener el archivo predeterminado de OpenDaylight es:

```
cp -n ~/.m2/settings.xml{,.orig}

wget -q -O - https://raw.githubusercontent.com/opendaylight/odlparent/
master/settings.xml > ~/.m2/settings.xml
```

Para generar la estructura del proyecto Maven se usa el arquetipo *opendaylight-startup-archetype* con -DarchetypeVersion acorde a la versión de OpenDaylight utilizada. Se ejecuta el siguiente comando:

```
mvn archetype:generate -DarchetypeGroupId=org.opendaylight.controller
-DarchetypeArtifactId=opendaylight-startup-archetype -DarchetypeRepository
=http://nexus.opendaylight.org/content/repositories/opendaylight.release/
-DarchetypeCatalog=remote -DarchetypeVersion=1.5.0
```

Se cargan correctamente los valores de las propiedades asumiendo de ahora en adelante que el nombre del proyecto es *example* a modo ilustrativo.

```
'groupId': org.opendaylight.example
'artifactId': example
'package' org.opendaylight.example: :
'classPrefix' Example: :
'copyright': example inc.
```

Luego, dentro del directorio del proyecto, se procede a compilar usando el siguiente comando:

```
mvn clean install -DskipTests
```

Una vez creado correctamente el proyecto, el directorio contiene tres sub módulos de importancia: api, impl y features. La estructura del proyecto Maven se ilustra en la Figura B.1.

En este punto se procede a realizar los cambios necesarios para obtener la funcionalidad requerida. Se crea la lógica de programación adecuada en los archivos ubicados dentro de la carpeta *impl/src/main/java/org/opendaylight/example/impl* y se añaden las dependencias correspondientes en los archivos *pom*.

B.2. Despliegue sobre ODL

Una vez que el proyecto ha compilado exitosamente, se procede a integrarlo al controlador. Apache Karaf permite el despliegue y ejecución de módulos que carga desde el directorio *system* de ODL. Por lo tanto, es necesario copiar los bundles generados desde el directorio del proyecto al directorio *system* de la distribución de ODL.

```
cp -r karaf/target/assembly/system/org/opendaylight/example
~/ODL/karaf-0.8.4/system/org/opendaylight
```

Se procede a agregar las features del proyecto al archivo features-index-0.8.4-features.xml de ODL ubicado en *ODL/karaf-0.8.4/system/org/opendaylight/integration/features-index/0.8.4*. Para ello es necesario agregar una línea por feature al archivo mencionado, de la siguiente forma:

```
<repository>mvn:GROUP-ID/FEATURE-ID/PROJECT-VERSION/xml/features</repository>
```

En el ejemplo en cuestión, se agregan las siguientes líneas:

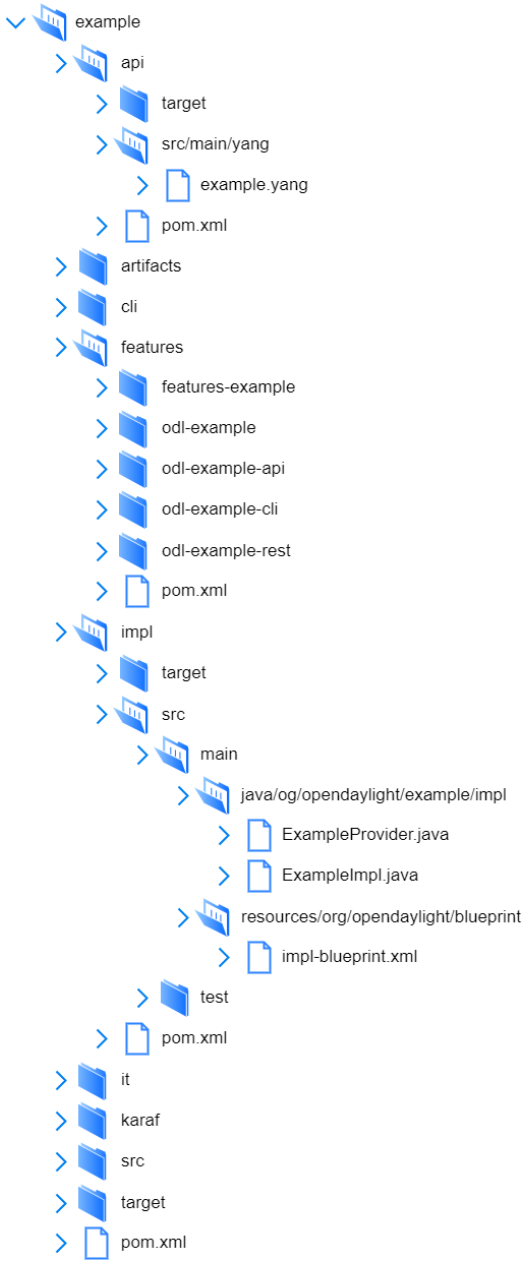


Figura B.1: Estructura de proyecto Maven.

```
<repository>mvn:org.opendaylight.example/features-example/0.1.0/xml/features
</repository>
<repository>mvn:org.opendaylight.example/odl-example/0.1.0/xml/features
</repository>
<repository>mvn:org.opendaylight.example/odl-example-cli/0.1.0/xml/
features</repository>
<repository>mvn:org.opendaylight.example/odl-example-api/0.1.0/xml/
features</repository>
<repository>mvn:org.opendaylight.example/odl-example-rest/0.1.0/xml/
features</repository>
```

Finalizado esto, se puede proceder con la instalación de las features. Para ello, se ejecuta el controlador y se registran los repositorios de cada feature añadida en el paso anterior.

```
opendaylight@root> feature:repo-add
mvn:GROUPID/FEATURE-ID/PROJECT-VERSION/xml/features
```

Se verifica que las features están disponibles para ser instaladas y finalmente se instalan:

```
opendaylight@root> feature:list | grep example

opendaylight@root> feature:install FEATURE-ID
```

Finalmente se corrobora que el módulo fue instalado correctamente y se encuentra activo:

```
opendaylight@root> bundle:list | grep example
```

B.3. Módulos desarrollados

A continuación se describen los módulos generados a lo largo de este trabajo y los complementos disponibles en OpenDaylight necesarios para su desarrollo [10]. Como se mencionó previamente, para este proyecto se crearon dos módulos que son de utilidad para el envío y recepción de paquetes desde el controlador. Estos son respectivamente *Mypacket* y *PacketinListener*. Para el desarrollo de ambos módulos se utilizaron funcionalidades disponibles en el plugin Openflow del controlador.

OpenflowPlugin contiene el módulo YANG *packet-processing* que, como el nombre lo indica, define modelos de datos, notificaciones y RPCs relacionados con el procesamiento de paquetes. Se destacan los siguientes modelos de RPC y Notificación respectivamente que son de utilidad:

- transmit-packet: envío de paquetes a través de un dispositivo Openflow desde el controlador.
- packet-received: entrega de paquetes entrantes envueltos en estrucutra Openflow (Packet-in).

A continuación se detallan las interfaces disponibles que implementan los modelos mencionados [9]:

- PacketProcessingService: interfaz generada a partir de la definición de RPCs en el módulo *packet-processing*. Dispone de un método llamado *transmitPacket* que coincide con el RPC definido en dicho módulo.
- PacketProcessingListener: interfaz generada a partir de las notificaciones definidas en el módulo *packet-processing*. La notificación *packet-received* se traduce a un método específico en la interfaz de la forma *onPacketReceived*.

B.3.1. Módulo Mypacket

El objetivo en este caso es implementar y registrar un servicio RPC en el controlador que permita solicitar el envío de mensajes desde un dispositivo Openflow. El primer paso es generar el modelo YANG que defina una estructura para el envío de dichos mensajes a través de la operación RPC *send-packet*. A continuación, se muestra el modelo definido en el archivo *mypacket.yang*:

```
rpc send-packet {

        input {
            leaf egress {
                type inv:node-connector-ref;
            }
            leaf node {
                type inv:node-ref;
            }
            leaf rawpacket {
                type binary;
            }
        }
        output {
            leaf result {
                type string;
            }
        }
}
```

Luego se procede a implementar la lógica para el servicio RPC definido en el módulo YANG. Si el proceso de implementación y registro del servicio se realiza correctamente, se puede usar la REST API para enviar mensajes a través del módulo *MyPacket* en el controlador.

A continuación se presenta el formato de una solicitud *send-packet*, junto con su contenido y el URL correspondiente. En este ejemplo, se genera una solicitud de tipo POST al módulo diseñado para transmitir un mensaje desde el nodo *openflow:1* a través de la interfaz *openflow:1:2*.

```
URL:
http://ControllerIP:Port/restconf/operations/mypacket:send-packet

{
"input": {
    "egress": "/opendaylight-inventory:nodes/opendaylight-
    inventory:node[opendaylight-inventory:id='openflow:1']/
    opendaylight-inventory:node-connector[opendaylight-
    inventory:id='openflow:1:2']",

    "node": "/opendaylight-inventory:nodes/opendaylight-
    inventory:node[opendaylight-inventory:id='openflow:1']",

    "rawpacket": "kPZS9daykPZS9buuCABFAABBkVhAAEARTDmsEAH9rB
    AC/Q+hE4kALVsiTWVhc3VyZQ=="
    }
}
```

B.3.2. Módulo PacketinListener

El propósito de este módulo consiste en detectar los mensajes *packet-in* en el controlador, a fin de procesarlos en la aplicación según corresponda.
Para llevar a cabo esto, se emplea el método *onPacketReceived* que se encuentra disponible en la interfaz *PacketProcessingListener*. Específicamente, se procede a sobrescribir el método mencionado con el fin de implementar la lógica deseada al recibir un *packet-in*. Cuando esto ocurre, se generan *logs* que contienen la notificación correspondiente, que sigue el siguiente formato:

```
PacketReceived [_flowCookie=FlowCookie [_value=24], _ingress=
NodeConnectorRef[_value=KeyedInstanceIdentifier{targetType=interface
org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.
node.NodeConnector, path=[org.opendaylight.yang.gen.v1.urn.opendaylight.
inventory.rev130819.Nodes, org.opendaylight.yang.gen.v1.urn.opendaylight.
inventory.rev130819.nodes.Node[key=NodeKey [_id=Uri [_value=openflow:1]]],
org.opendaylight.yang.gen.v1.urn.opendaylight.inventory.rev130819.node.
NodeConnector[key=NodeConnectorKey [_id=Uri [_value=openflow:1:1]]]]}],
_match=Match [_inPort=Uri [_value=1], _metadata=Metadata [_metadata=1,
augmentation=[]], augmentation=[]], _packetInReason=class org.opendaylight.
yang.gen.v1.urn.opendaylight.packet.service.rev130709.SendToController,
_payload=[0, 0, 0, 0, 0, 16, 0, 0, 0, 0, 0, 1, 8, 0, 69, 0, 0, 38, 28, -104,
64, 0, 64, 17, -61, 12, -84, 16, 1, 1, -84, 16, 2, 1, 19, -117, 19, -119, 0,
18, -50, -105, 72, 111, 108, 97, 32, 77, 117, 110, 100, 111],
_tableId=TableId [_value=178], augmentation=[]]
```

A partir de la notificación recibida al detectar un *packet-in*, se puede obtener información de interés para el procesamiento del mismo. Principalmente, se utilizarán los campos *payload* y *tableId*, así como el valor de *metadata* si corresponde.

Apéndice C

Features OpenDaylight

Este Apéndice presenta las principales *features* desarrolladas en OpenDaylight y describe brevemente alguna de ellas [7]. A su vez, muestra las *features* instaladas para el desarrollo del presente proyecto y los comandos importantes para su instalación.

C.1. Descripción de *features*

Authentication, Authorization and Accounting (AAA) Services:
Brinda servicios de autenticación y seguridad.

ALTO:
Implementa el protocolo IETF básico de optimización de tráfico de capa de aplicación (ALTO) para proporcionar información de red a las aplicaciones. Define abstracciones y servicios para permitir vistas de red simplificadas y servicios de red para guiar el uso de recursos por parte de las aplicaciones.

BGP:
Es un complemento southbound que brinda soporte para el Protocolo Border Gateway.

DLUX:
Provee una interfaz gráfica de usuario basada en la web que incluye:

- Un visor de flujo MD-SAL
- Visualizador de topología de red
- Una caja de herramientas y un modelo YANG que ejecutan consultas y visualizan el árbol YANG

Group Based Policy (GBP):
Define un modelo de política centrado en la aplicación para OpenDaylight que separa la información sobre los requisitos de conectividad de la aplicación, de la

Tabla C.1: Principales features en OpenDaylight Oxygen.

Authentication, Authorization and Accounting (AAA) Services
Application-Layer Traffic Optimization (ALTO)
Border Gateway Protocol (BGP)
Border Gateway Monitoring Protocol (BMP)
DLUX
Group Based Policy (GBP)
L2switch
Location Identifier Separation Protocol (LISP) Flow Mapping
NEtwork MOdeling (NEMO)
NETCONF
OpenFlow Configuration Protocol (OF-CONFIG)
OpenFlowPlugin Project
OVSDB Project
Path Computation Element Protocol (PCEP)
Service Function Chaining (SFC)
SNMP Plugin
SNMP4SDN
Source-Group Tag Exchange Protocol (SXP)
Time Series Data Repository (TSDR)
Unified Secure Channel (USC)
Virtual Tenant Network (VTN)

información sobre los detalles subyacentes de la infraestructura de la red.

L2Switch:
Proporciona la funcionalidad de switch L2 cuya arquitectura se conforma de:

- Controlador de paquetes: Decodifica los paquetes que llegan al controlador y los envía adecuadamente.
- Eliminador de bucles: Elimina bucles en la red.
- Controlador de ARP: Maneja los paquetes ARP decodificados.
- Rastreador de direcciones: Aprende las direcciones (MAC e IP) de los elementos en la red.
- Rastreador de host: Rastrea las ubicaciones de los hosts en la red.
- Switch L2 principal: Instala flujos en cada switch en función del tráfico de red.

OpenFlow plugin:
Admite la conexión a dispositivos de red habilitados para OpenFlow a través de la especificación OpenFlow. Actualmente es compatible con las versiones 1.0 y 1.3.2 de OpenFlow.

C.2. Instalación de *features*

Es necesario instalar *features* al controlador, que por defecto no trae incorporadas, para cumplir con tareas específicas según la funcionalidad requerida. Para el desarrollo del presente trabajo fue necesario instalar las siguientes *features*:

- odl-l2switch-arphandler, odl-l2switch-addresstracker, odl-l2switch-hosttracker, odl-l2switch-loopremover, odl-l2switch-packethandler, odl-l2switch-switch
- odl-mdsal-apidocs, odl-mdsal-binding-api, odl-mdsal-binding-base, odl-mdsal-binding-dom-adapter, odl-mdsal-binding-runtime, odl-mdsal-broker, odl-mdsal-broker-local, odl-mdsal-clustering-commons, odl-mdsal-common, odl-mdsal-distributed-datastore, odl-mdsal-dom, odl-mdsal-dom-api, odl-mdsal-dom-broker, odl-mdsal-eos-binding, odl-mdsal-eos-common, odl-mdsal-eos-dom, odl-mdsal-models, odl-mdsal-model-inventory, odl-mdsal-remoterpc-connector, odl-mdsal-singleton-common, odl-mdsal-singleton-dom
- odl-openflowplugin-app-arbitratorreconciliation, odl-openflowplugin-app-config-pusher, odl-openflowplugin-app-forwardingrules-manager, odl-openflowplugin-app-topology, odl-openflowplugin-app-reconciliation-framework, odl-openflowplugin-flow-services, odl-openflowplugin-libraries, odl-openflowplugin-nsf-model, odl-openflowplugin-southbound
- odl-restconf, odl-restconf-common, odl-restconf-noauth
- odl-yangtools-codec, odl-yangtools-common, odl-yangtools-data, odl-yangtools-data-api, odl-yangtools-export, odl-yangtools-parser, odl-yangtools-parser-api, odl-yangtools-util

A continuación se presentan comandos importantes para la gestión de *features* en OpenDaylight [3]:

Para instalar *features* se ejecuta el siguiente comando:

```
opendaylight-user@root> feature:install <feature-name>
```

También es posible instalar más de una como se muestra a continuación:

```
opendaylight-user@root> feature:install <feature1> <feature2> ...
```

Se puede consultar un listado de *features* disponibles para instalar con el siguiente comando:

```
opendaylight-user@root> feature:list [-i] [| grep <feature-name>]
```

Con la opción *-i* se verifican las *features* instaladas, y para especificar la búsqueda se usa el comando *grep*.

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] 7 sdn benefits: Software defined networking. https://ipwithease.com/sdn-benefits/. (Online; Fecha de acceso 30-10-2023).

[2] Download/get started with mininet. http://mininet.org/download/. (Online; Fecha de acceso 30-10-2023).

[3] Install the Karaf features. https://test-odl-docs.readthedocs.io/en/latest/getting-started-guide/installing_opendaylight.html#install-the-karaf-features. (Online; Fecha de acceso 30-10-2023).

[4] Mininet Overview. https://mininet.org/overview/. (Online; Fecha de acceso 30-10-2023).

[5] Opendaylight Controller overview. https://docs.opendaylight.org/projects/controller/en/latest/dev-guide.html#. (Online; Fecha de acceso 30-10-2023).

[6] Opendaylight downloads. http://www.opendaylight.org/software/downloads. (Online; Fecha de acceso 30-10-2023).

[7] Opendaylight Karaf Features. https://opendaylight-documentation.readthedocs.io/en/stable/getting-started-guide/karaf_features.html. (Online; Fecha de acceso 30-10-2023).

[8] Opendaylight Oxygen Release. https://www.opendaylight.org/what-we-do/current-release/oxygen,. (Online; Fecha de acceso 30-10-2023).

[9] Opendaylight Packet Service. https://javadocs.opendaylight.org/org.opendaylight.openflowplugin/oxygen/org/opendaylight/yang/gen/v1/urn/opendaylight/packet/service/rev130709/package-summary.html. (Online; Fecha de acceso 30-10-2023).

[10] The OpenDaylight Project. https://github.com/opendaylight. (Online; Fecha de acceso 30-10-2023).

[11] Opendaylight "ODL OpenFlowPlugin Release master". https://docs.opendaylight.org/_/downloads/openflowplugin/en/latest/pdf/. (Online; Fecha de acceso 30-10-2023).

[12] Openflow Plugin Operation. https://docs.opendaylight.org/projects/openflowplugin/en/latest/users/operation.html#openflow-programming. (Online; Fecha de acceso 30-10-2023).

[13] Openflow switch specification, version 1.5.1. https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. (Online; Fecha de acceso 30-10-2023).

[14] Openwrt project. https://openwrt.org/. (Online; Fecha de acceso 30-10-2023).

[15] Smartroute. https://gitlab.fing.edu.uy/smartroute/smartroute.

[16] Pascal Berthoua Douglas C. Schmidtc Gayraud Thierrya Akram Hakiria, Aniruddha Gokhalec. Software-defined networking: Challenges and research opportunities for future internet. *Computer Networks*, 2014.

[17] Isabel Amigo, Gabriel Gómez, Marwa Chami, and Pablo Belzarena. An sdn-based approach for qos and reliability in overlay networks. In *TMA Conference 2018 : Network Traffic Measurement and Analysis Conference, Vienna, Austria, 26-29 jun*, pages 1–2, 2018.

[18] Pablo Belzarena, Gabriel Gómez Sena, Isabel Amigo, and Sandrine Vaton. Sdn-based overlay networks for qos-aware routing. In *Proceedings of the 2016 Workshop on Fostering Latin-American Research in Data Communication Networks*, page 19–21, 2016.

[19] Santiago Bentancur, Martín Fernández Bon, Gabriel Gómez, Claudina Rattaro, and Ignacio Brugnoli. Flow-based qos forwarding strategy: a practical implementation and evaluation. In *2020 Seventh International Conference on Software Defined Systems (SDS), Paris, France, 20-23 apr*, pages 274–279. IEEE, 2020.

[20] Paul Göransson; Chuck Black. *Software Defined Network, A Comprehensive Approach*. Morgan Kaufmann, 2014.

[21] Ignacio Brugnoli, Martín Fernández, and Diego Mazzuco. "Overlay Network Routing Application (ONRApp)", jun 2019. Available: https://iie.fing.edu.uy/publicaciones/2019/BFM19.

[22] Ignacio Brugnoli, Martín Fernández Bon, Diego Mazzuco, Gabriel Gómez, Claudina Rattaro, and Santiago Bentancur. Tunnelless sdn overlay architecture for flow based qos management. In *24th Conference on Innovation in Clouds, Internet and Networks (ICIN 2021), Paris, France, 1-4 mar*, pages 1–8. IEEE, 2021.

[23] A. Bose C. Labovitz, A. Ahuja and F. Jahanian. Delayed internet routing convergence. *ACM SIGCOMM Computer Communication Review*, 2000.

[24] C. Labovitz F. Jahanian and A. Ahuja. Experimental study of internet stability and wide-area backbone failures. *U. of Michigan, Tech. Rep. CSETR-382-98,*, 1998.

[25] V. Paxson. End-to-end routing behavior in the internet. *IEEE/ACM transactions on Networking*, 1997.

[26] Martín Randall. Optimización del ruteo en redes sobrepuestas con sistemas de decisión en base a medidas. Master's thesis, Universidad de la República (Uruguay). Facultad de Ingeniería. IIE, sep 2020. Available: `https://iie.fing.edu.uy//publicaciones/2020/Ran20`.

Esta página ha sido intencionalmente dejada en blanco.

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.
Compilado el domingo 28 enero, 2024.
http://iie.fing.edu.uy/