UNIVERSIDAD POLITÉCNICA DE CARTAGENA

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Trabajo Fin de Máster

Desarrollo de una interfaz entre el controlador SDN ONOS y Net2Plan para la optimización de redes

Javier López Fernández

Octubre de 2018

Director:
Pablo Pavón Mariño

Autor	Javier López Fernández
E-mail del Autor	jlf2@alu.upct.es
Director(es)	Pablo Pavón Mariño
E-mail del Director	pablo.pavon@upct.es
Título del TFM	Desarrollo de una interfaz entre el controlador SDN ONOS y Net2Plan para la optimización de redes

Resumen

El paradigma *SDN* se ha convertido en la manera más eficiente de gestionar las redes actuales, gracias a la automatización de las funciones de operación y gestión. Debido a esto, han ido surgiendo herramientas para la aplicación de las técnicas *SDN*. Una de ellas es el Controlador *ONOS*, que permite centralizar las decisiones de encaminamiento y liberar a los equipos de red de esta tarea, lo que permite simplificar su hardware. Sin embargo, estas decisiones son inherentes a *ONOS*, por lo que no se pueden modificar. La herramienta de optimización *Net2Plan* permite la ejecución de algoritmos que realicen estas decisiones, proporcionando mayores ventajas a los operadores de red. Este proyecto tiene como objetivo la integración de *Net2Plan* como aplicación *Northbound* de *ONOS*, permitiendo la visualización de la topología, y la ejecución de algoritmos de ingeniería de tráfico.

Titulación	Máster Universitario en Ingeniería de Telecomunicación
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Septiembre 2018

Índice

CAPÍTULO 1

Introducción

1.1. Objetivos

El objetivo principal de este proyecto es desarrollar un *plugin* para *Net2Plan* que ofrezca una comunicación con la interfaz norte del controlador *SDN ONOS*. Esta interfaz se convertirá en un punto de partida que permita a Net2Plan proveer de funcionalidades de optimización para redes controladas con *ONOS*.

1.2. SDN

Las Redes Definidas por Software (Software Defined Networking (SDN)) [1] es un nuevo paradigma de orquestación de redes cuyo objetivo es conseguir que la arquitectura de una red sea más dinámica, fácil de reconfigurar e interoperable, sin importar modelos o fabricantes. Está basado en la idea de permitir a los operadores gestionar todos los servicios de la red de forma externa a los dispositivos, mediante la abstracción de funcionalidades de alto nivel (capa de control).

Para conseguir este objetivo, SDN desacopla el plano de control del plano de datos de

una red. De esta manera, la capa de control es independiente del hardware y la lógica de los dispositivos, y la toma de decisiones respecto al encaminamiento de paquetes y flujos de tráficos se toma de forma externa e independientemente de los sistemas existentes en el plano de datos. Esto implica que las funciones de control y de encaminamiento están separadas (ver figura 1.1).

Entre las beneficios que ofrece la implantación de SDN se encuentran:

- Automatización y agilización a la red, mediante la simplificación de las operaciones a realizar sobre la misma. Con esto se consigue una reducción de la complejidad a través de la separación del plano de control y el plano de datos.
- Construcción de redes programables eliminando la configuración manual. De esta manera se consigue una red más segura y escalable, ya que se reduce la intervención humana.
- Aumento de la velocidad de implementación y despliegue de nuevas aplicaciones y servicios, favoreciendo este desarrollo a través de una Application Programming Interface (API).

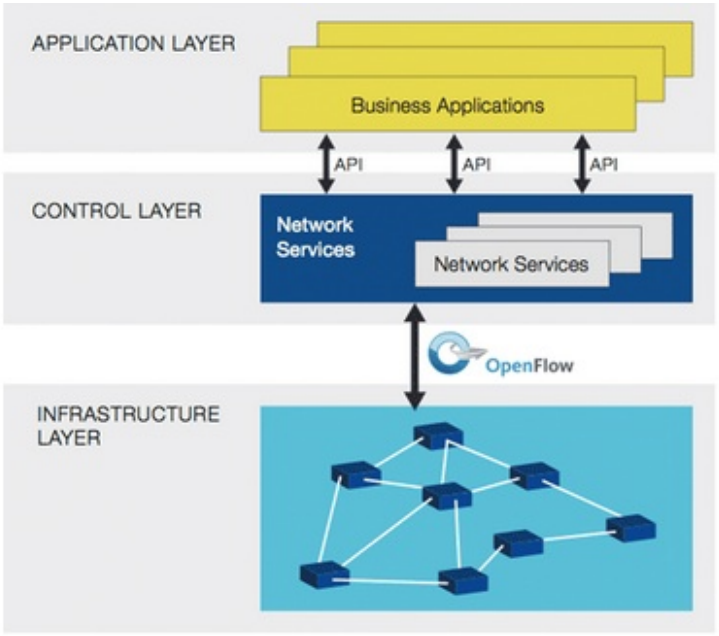

Figura 1.1: Arquitectura de SDN

La arquitectura del paradigma SDN es la siguiente:

- **Southbound API:** permite la lectura y escritura de configuraciones en los dispositivos de encaminamiento (routers y switches). A día de hoy protocolos como *OpenFlow* están

muy de moda, pero los controladores SDN también permiten otras opciones como NETCONF, SNMP, etc.

- **Northbound API:** permite la comunicación con aplicaciones de la capa superior, de forma que otras aplicaciones o software hecho a medida permitan acceder a las funcionalidades que proporciona SDN.

- **Controlador:** Considerado el cerebro de la red, posee una visión global de la misma. Toma las decisiones sobre el encaminamiento de los paquetes y flujos de tráfico, que luego se transmiten a los dispositivos mediante la interfaz southbound.

En la actualidad existen varios controladores SDN, siendo los más utilizados *ODL* (*OpenDaylight*) y *ONOS*. Sus principales características son las siguientes:

1.2.1. OpenDaylight

OpenDaylight (ODL)[2] es un proyecto colaborativo *open source* fundado por *The Linux Foundation*. Su objetivo es facilitar la implantación de las nuevas tecnologías SDN y Network Functions Virtualization (NFV) en las redes actuales. Su principales características son:

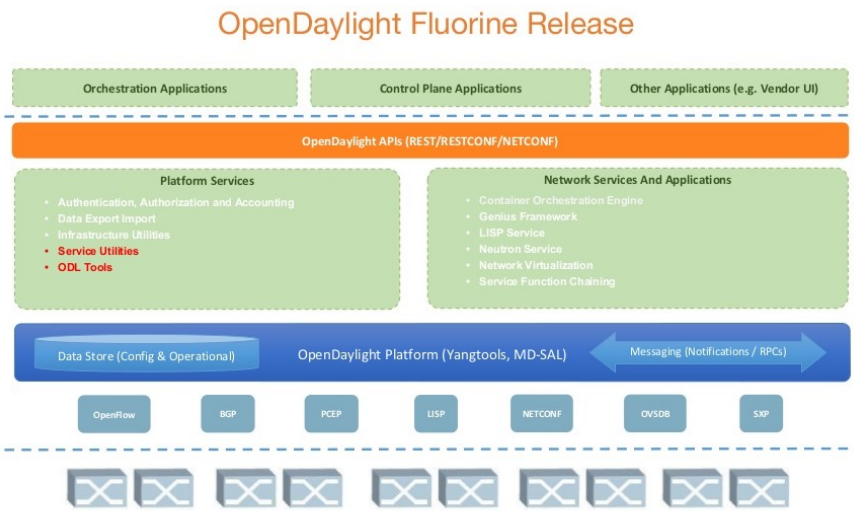

Figura 1.2: Arquitectura de ODL Flourine

- Infraestructura modular, extensible, escalable y multiprotocolo. Además ofrece una alta disponibilidad.

- Soporte de múltiples protocolos *southbound*, como por ejemplo *OpenFlow*, *NETCONF* y *OVSDB*.
- Escrito en *Java*.
- Uso del lenguaje *YANG* para el modelado de datos, mientras que la comunicación entre las diferentes interfaces se realiza mediante *REST API*.
- Recibe el apoyo de importantes empresas dedicadas a las telecomunicaciones. Entre las empresas que forman parte de los miembros de ODL se encuentran Cisco, HP o Intel. Su participación en este proyecto consiste en proporcionar desarrolladores para la evolución y el soporte de esta herramienta.

Su última *release* en el momento en el que ha sido escrito este documento es *Flourine*, publicada en agosto de 2018.

1.2.2. ONOS

Open Network Operating System (ONOS)[3] (*Open Network Operating System*) es un sistema operativo para la gestión de los componentes de red en un entorno SDN lanzado por el *Open Networking Lab (ON. Lab)* en 2014 y actualmente bajo el paraguas de la ONF (*Open Network Foundation*). Su principal objetivo es facilitar el despliegue y desarrollo de soluciones SDN y NFV, al igual que ODL. También es un sistema *open source* apoyado por empresas privadas como AT&T, Huawei o Intel entre otras.

Además, ambos controladores comparten algunas características fundamentales. Las peculiaridades que ofrece ONOS con respecto a ODL son:

- **Código modular:** El proyecto está dividido en una serie de subproyectos independientes entre sí. Esto facilita a los desarrolladores la implementación de nuevas funcionalidades en un módulo sin afectar al resto.
- **Configurabilidad:** Cada una de las funciones que ofrece ONOS puede ser activada o desactivada, incluso en tiempo de ejecución, gracias al uso de *Apache Karaf*. Esto es muy útil a la hora de ampliar las características del controlador sin necesidad de tener que parar y volver a lanzar el servicio.
- **Abstracción y simplicidad:** Estas características favorecen el desarrollo de nuevas aplicaciones y soluciones.
- **Compatibilidad con cualquier equipo de red:** Esta característica hace de ONOS un entorno altamente versátil e interoperable, compatible con diferentes vendedores o configuraciones de equipos de red.

Su última *release* conocida es *Nightingale*, publicada en Marzo de 2018.

1.3. Net2Plan

Net2Plan[4] es una herramienta de software libre (open source) escrita en Java y desarrollada por el grupo GIRTEL[5] de la Universidad Politécnica de Cartagena. Esta herramienta entre otras funcionalidades permite planificar, optimizar y evaluar las redes de comunicaciones. La primera versión data de septiembre de 2011, y fue planteada inicialmente como una herramienta de soporte a la docencia en cursos de planificación y gestión de redes en la UPCT. Con el tiempo, se ha convertido en una poderosa herramienta de optimización y planificación, tanto para la docencia e investigación como para la industria. Además cuenta con un creciente repositorio de recursos disponibles online. La versión más reciente es la 0.6.0, lanzada en julio de 2018.

Net2Plan está construido sobre una representación tecnológicamente agnóstica de la red (llamada *Network plan*), basada en componentes abstractos: nodos, enlaces, demandas, rutas, segmentos de protección (SRG), árboles multicast, capas de red, etc. La representación de la red no depende de una tecnología concreta, por tanto Net2Plan puede adaptarse para planificar todo tipo de redes. Para que la adaptación sea efectiva, la información específica de la tecnología puede ser introducida a través de atributos (definidos por el usuario), unidos a cualquiera de los componentes mencionados anteriormente. Algunos nombres de atributos han sido fijados para facilitar la adaptación de algunas tecnologías conocidas (como por ejemplo, redes IP).

Net2Plan permite su ejecución tanto mediante una interfaz gráfica (Graphical User Interface (GUI)) como por línea de comandos (Command Line Interface (CLI)). La interfaz gráfica es especialmente útil como recurso docente para sesiones de laboratorio, o para la inspección visual de la red. Por otro lado, la interfaz de línea de comandos está dedicada a estudios de investigación en profundidad, haciendo uso de procesamiento por lotes o simulación a gran escala. Por lo tanto, Net2Plan es una herramienta destinada a un amplio espectro de usuarios: industria, investigación y el campo académico.

Sin tener en cuenta la interfaz seleccionada por el usuario (CLI o GUI), Net2Plan proporciona actualmente cuatro herramientas diferentes:

- **Diseño de red offline:** Enfocada a la evaluación de los diseños de red mediante algoritmos incorporados en la herramienta o definidos por el usuario, decidiendo aspectos como la topología de red, el enrutamiento del tráfico, la capacidad de los enlaces, las rutas de protección, etc. Si fuera necesario, aquellos algoritmos de optimización basados en programación matemática (ILPs), pueden ser resueltos usando la librería *open*

source Java Optimization Modeler (JOM)[6] (también desarrollada por el grupo GIRTEL), que permite el uso de una interfaz común para conectar con una serie de *solvers* externos, como *GPLK*, *CPLEX* o *IPOPT*.

- **Generación de matrices de tráfico:** Asiste a los usuarios en el proceso de generación y normalización de matrices de tráfico, por ejemplo siguiendo un modelo aleatorio, gaussianos, etc.

- **Simulación online:** Permite la evaluación de esquemas de recuperación de red, sistemas CAC (connection-admission-control), o algoritmos dinámicos de aprovisionamiento para tráfico variante en el tiempo. Al final de la simulación, algunas métricas (incluidas métricas predefinidas o personalizadas) son presentadas.

- **Informes:** Ne2Plan permite la generación de informes (incluidos en la herramienta, o definidos por el usuario) para cualquier diseño de red. La herramienta de generación de informes está integrada dentro junto con todas las funcionalidades previas, de modo que es posible crear informes recopilando medidas de rendimiento en cualquiera de estos aspectos.

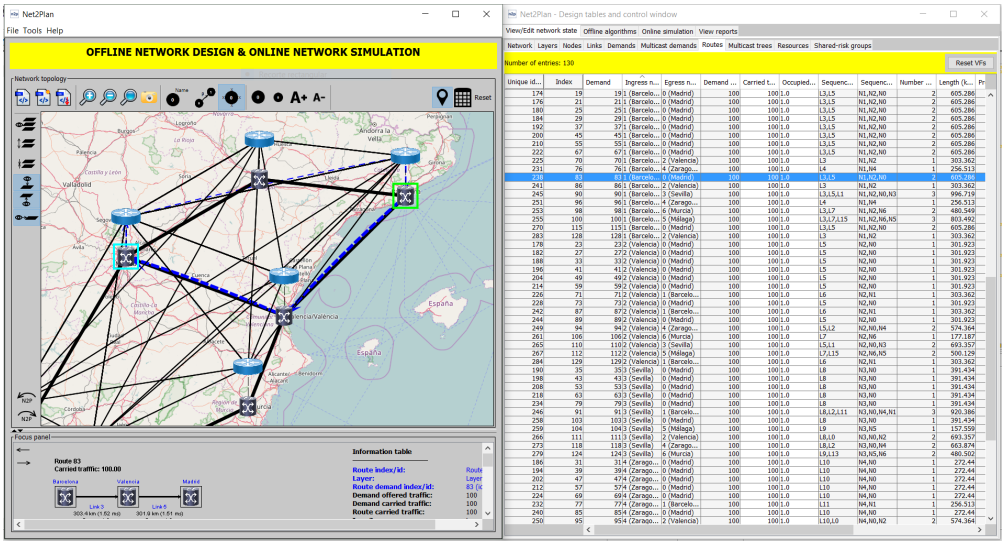

Figura 1.3: Interfaz gráfica de Net2Plan

Los algoritmos que pueden ejecutarse en los modos *Offline Network Design* y *Online Simulation* son clases de *Java* que implementan una interfaz pública definida en la documentación. Para crear un nuevo algoritmo para una funcionalidad particular de Net2Plan, sólo se requiere la programación de una clase en *Java* que implemente la interfaz antes mencionada.

En cuanto al modelo de red usado por *Net2Plan*, destacan las siguientes entidades:

- **Network:** Es el objeto genérico del que cuelgan todos los componentes que definen la red. Este objeto será con el que trabajaran los diferentes algoritmos o informes

ejecutados en *Net2Plan*, ya que proporciona acceso a todos los elementos de la red (nodos, links, demandas, etc.)

- **Node:** Es una entidad básica que define un nodo de la red, que será la entidad que defina los extremos de un componente que represente un enlace punto a punto o punto a multipunto. Posee información que caracteriza al nodo, como por ejemplo sus coordenadas (X-Y o Longitud-Latitud), su estado (si está activo o caído) o el número de usuarios a los que da acceso.

- **Link:** Este objeto representa la conexión entre dos nodos mediante un enlace. Lo definen una serie de atributos, como por ejemplo su capacidad, su longitud, el tráfico que es capaz de cursar, y la capa en la que se encuentra. Los enlaces de *Net2Plan* son siempre unidireccionales, y no están permitidos los autoenlaces.

- **Demand:** Define demandas de tráfico *unicast* entre dos nodos. Su principal atributo es el tráfico inyectado por el nodo origen que va hacia el nodo destino. Además, en las últimas versiones se han incluido funcionalidades de calidad de servicio (QoS) aplicables a este objeto.

- **Route:** Mediante este objeto se define el encaminamiento de la red. Determina cómo una demanda de tráfico es cursada, y a través de que secuencia de enlaces debe serlo desde el origen de la demanda hasta su destino.

- **Shared-risk groups:** Representa un riesgo de fallo en la red dado que si ocurriera, produciría una serie de caídas simultáneas en los diferentes enlaces y nodos incluidos en cada *SRG*. De esta forma se puede observar el comportamiento de la red en casos extremos, y si la red ha sido correctamente planificada para soportar caídas simultáneas en varios puntos de la misma. Está definido por valores estadísticos, como por ejemplo el tiempo medio entre fallos ($MTTF$), el tiempo medio de reparación ($MTTR$) y la disponibilidad (A).

Por último, cabe destacar que Net2Plan facilita el desarrollo de extensiones (o *plugins*), que permiten añadir funcionalidades y características a la herramienta, ajustándose a las necesidades del usuario.

1.4. Swagger

Swagger[7] es una herramienta *online open source* que ofrece herramientas a los desarrolladores para la generación automática de APIs. De entre las múltiples herramientas que ofrece destaca la herramienta *Swagger Editor*, que permite diseñar, describir y documentar desde cero una API.

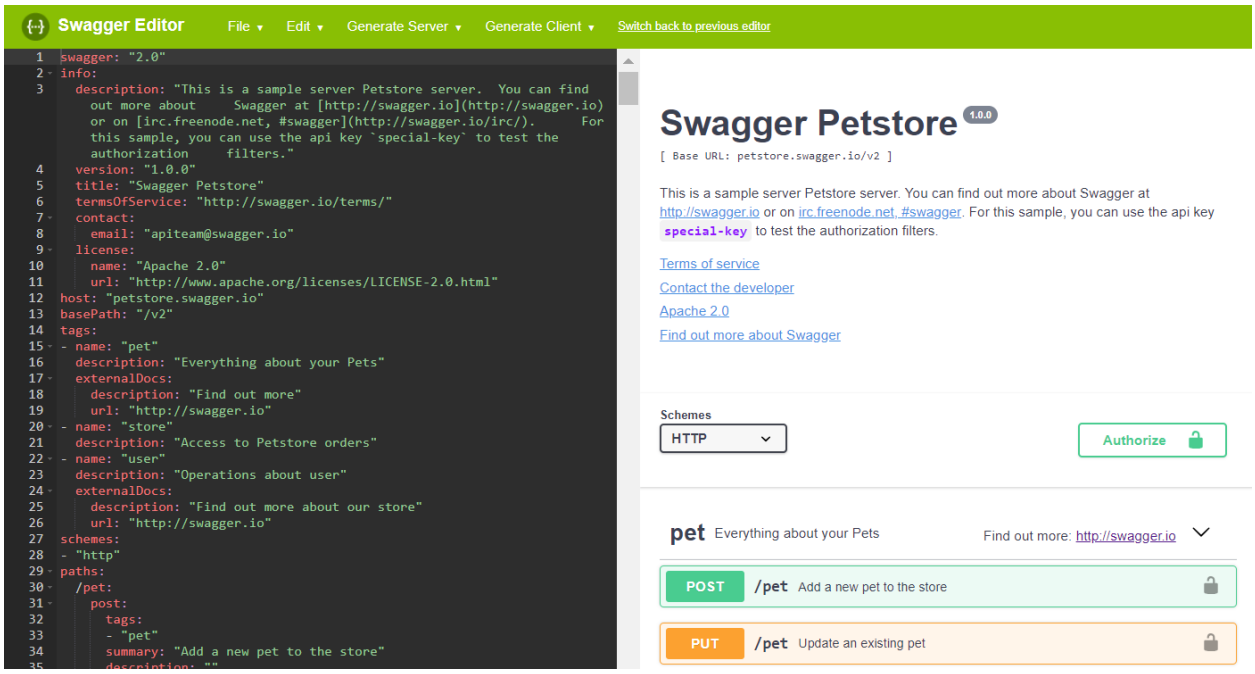

Figura 1.4: Swagger Editor

El proceso para generar una API en *Swagger Editor* consiste en introducir la definición del servidor en formato *YAML*. Una vez definido esta herramienta da la oportunidad de generar el código que implementa tanto el servidor que ofrece la API como el cliente para comunicarse con dicho servidor, en cualquiera de los lenguajes de programación disponibles.

Gracias a esto, cualquier servidor generado por *Swagger* ofrecerá la facilidad de generar automáticamente un cliente mediante *Swagger Editor*, siempre y cuando su definición sea accesible.

1.5. Partes del documento

En los capítulos posteriores se detallará el proceso que se ha llevado para la confección de este proyecto.

En primer lugar, se entrará en detalle del controlador ONOS en el capítulo 2. A continuación, el capítulo 3 contendrá la arquitectura del software desarrollado. En el capítulo 4 aparecerá un caso de uso de la herramienta, y por último el capítulo final albergará las conclusiones y los posibles pasos a seguir tras la finalización de este proyecto.

CAPÍTULO 2

ONOS

ONOS (*Open Networking Operating System*) es un controlador SDN cuyo principal objetivo es permitir que los proveedores de servicios construyan soluciones *SDN/NFV* reales.

En este capítulo se detallará la arquitectura del controlador ONOS, y las diferentes vías para interaccionar con él a nivel de usuario.

2.1. Arquitectura

Los principios de su arquitectura son los siguientes:

- Alta disponibilidad, escalabilidad y rendimiento.
- Fuerte abstracción y simplicidad.
- Independencia entre el protocolo y el comportamiento de los equipos.
- Modularidad.

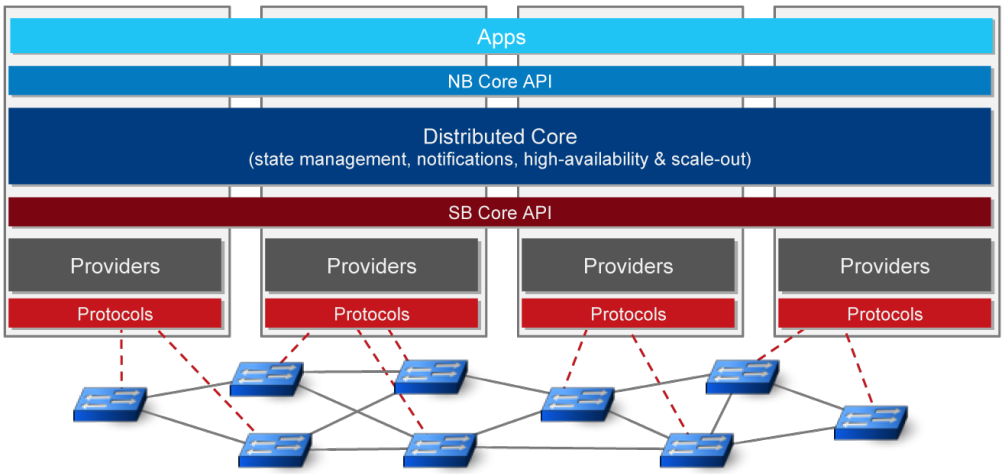

Figura 2.1: Arquitectura de ONOS

La arquitectura de ONOS se divide en tres capas: *Interfaz Southbound*, *Core* y *Interfaz Northbound*.

2.1.1. Interfaz Southbound

La *Interfaz Southbound*[8] de ONOS es la encargada de la comunicación entre el *core* y los módulos que interactúan directamente con la red. Ofrece las siguientes características:

- Abstracción, modularidad e interoperabilidad.
- Uso de los equipos de red en vivo. De esta manera no es necesario la desconexión de todo el sistema para añadir nuevos equipos de red al controlador.
- Ocultar la complejidad de esta capa a las capas superiores.

Para ello, utiliza las siguientes entidades:

- ***Providers***: Entidad que contiene el resumen de la configuración de bajo nivel. También realiza el control y la gestión de las operaciones asociadas a los equipos de red. Ejecuta además peticiones originadas en el *core*, y procesa y notifica al *core* eventos originados en los equipos.
- ***Protocols***: Contiene todas las características necesarias para la comunicación entre ONOS y los equipos de red. En esta entidad se encuentra la implementación de los

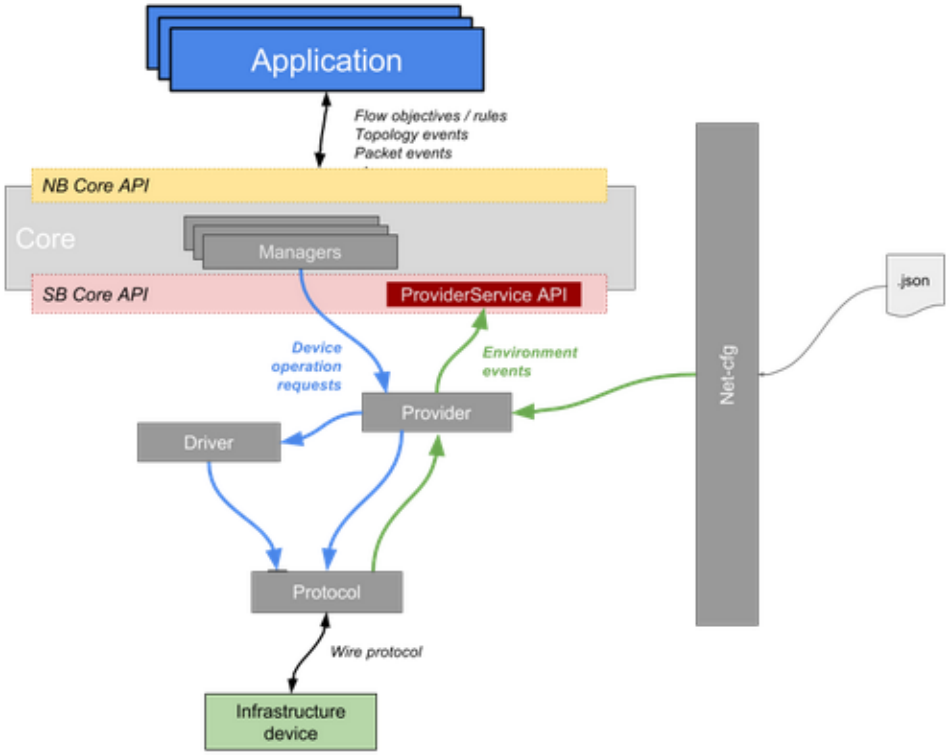

Figura 2.2: Arquitectura de la Interfaz Southbound de ONOS

protocolos específicos de control y gestión de red, como pueden ser *OpenFlow*, *NETCONF*, *SNMP*, etc.

- ***Drivers***: Un *driver* en ONOS es una representación específica de una familia de *devices*. Define el comportamiento del *device* al que representa, por lo que es el componente que ofrece la característica de interoperabilidad en ONOS. Sin el *driver* necesario habrá *devices* que no sean compatibles con ONOS. Además, gracias a esta entidad se pueden definir *devices* virtuales, es decir, *devices* útiles para la investigación con un comportamiento definido que físicamente no son reales.

2.1.2. Core

El *core*[9] de ONOS compone el núcleo de la arquitectura, encargado de rastrear y presentar la información recibida sobre el estado de la red a la capa de aplicaciones por medio de servicios. Un servicio está compuesto por múltiples componentes a lo largo de diferentes capas. Los servicios primarios que ofrece el *core* son:

- ***Device Subsystem*:** Gestiona los nodos de la red involucrados en tareas de encaminamiento. Existen varios tipos de *devices*, como por ejemplo *switches*, *routers* o puntos

de acceso. Un *device* está represetnado por un conjunto de interfaces/puertos y un *DeviceId*.

- ***Host Subsystem*:** Gestiona los nodos que actúan como origen y destino del tráfico llamados *hosts*. Un *host* puede ser un PC, un servidor o cualquier otro equipo que inyecta tráfico a la red. Contiene una dirección *IP*, una dirección *MAC*, un *VLAN ID* y un *ConnectPoint*.

- ***Link Subsystem*:** Gestiona los enlaces directos entre nodos. Los enlaces permitidos en ONOS son aquellos que conectan dos *devices* o un *device* con un *host*. Los enlaces que conectan directamente dos *hosts* no están permitidos.

- ***Topology Subsystem*:** Ofrece una instantánea del grafo que representa la red. Este servicio es utilizado por los algoritmos que computan el encaminamiento, como por ejemplo el algoritmo de *Dijkstra*.

- ***Path Subsystem*:** Computa las ruta (o *paths*), que consisten en un conjunto de uno o más enlaces (que deben de ser adyacentes) entre dos *hosts* de la red. Para ello utiliza la instantánea ofrecida por *Topology Subsystem*.

- ***FlowRule Subsystem*:** Gestiona el inventario de las reglas de flujo instaladas en los *devices*, y además proporciona métricas de flujo.

- ***Packet Subsystem*:** Permite a las aplicaciones observar los paquetes de tráfico recibidos por los *devices*, y además permite inyectar tráfico a la red.

Los servicios están compuestos por dos entidades:

- ***Manager*:** Encargado de la mediación entre los *providers* de la *Interfaz Southbound* y las aplicaciones. Para ello ofrece varias interfaces que se encargan de ofrecer a las aplicaciones el estado de la red en un momento determinado, ejecutar comandos de administración en los equipos, y establecer la comunicación entre el *manager* y las aplicaciones.

- ***Listener*:** Ofrece la posibilidad a las aplicaciones de registrarse para recibir notificaciones de eventos producidos en la red.

2.1.3. Interfaz Northbound

En la *Interfaz Northbound*[10] se encuentran los componentes encargados de comunicar el *core* de ONOS con los servicios y aplicaciones ejecutadas por encima de la red.

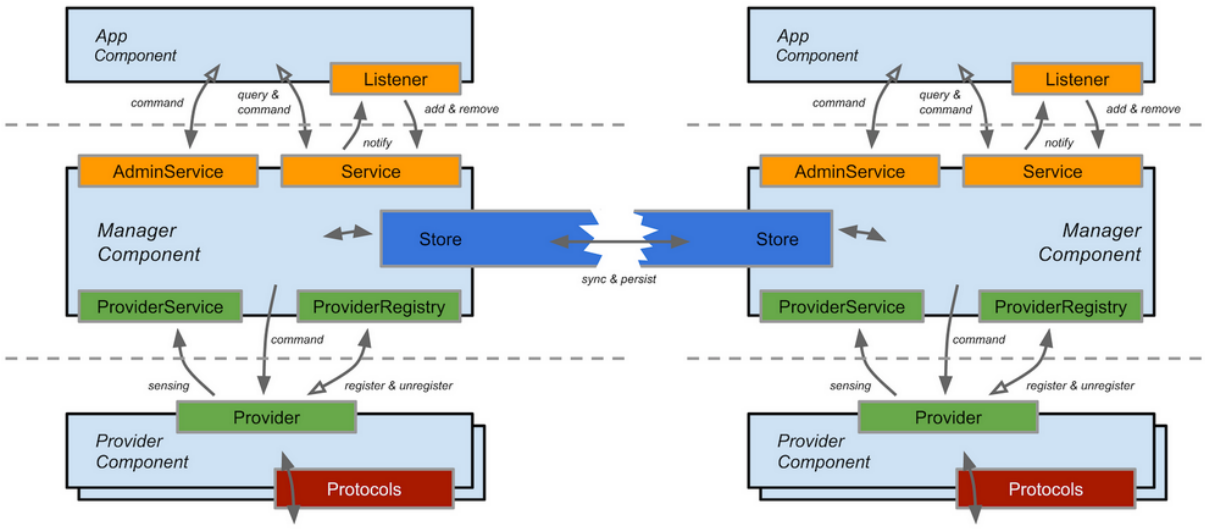

Figura 2.3: Estrucutura de un servicio en ONOS

ONOS ofrece una extensa API con múltiples capas de abstracción que permiten la eficiente automatización y configuración de la red para satisfacer las necesidades de las diferentes aplicaciones que se comunican con la *Interfaz Northbound*.

Entre las abstracciones *Northbound* implementadas por ONOS destacan:

- ***Intents***: Un *intent* se define como un conjunto de reglas de flujo. Es el mecanismo que ofrece ONOS a las aplicaciones para que expresen qué es lo que quieren que haga la red sin importar cómo lo debe hacer. Una vez creado un *intent*, el *core* realiza las acciones necesarias para llevarlo a cabo, de manera que las aplicaciones estén abstraídas del proceso realizado para este fin.

- ***Flow Objectives***: Define cómo los *devices* deben manejar el tráfico. Para ello, se realiza una abstracción del plano de datos específico de cada tipo de *device*, lo que permite a las aplicaciones definir un *flow objective* que sea ejecutado por todos y cada uno de los *devices* de la red. Existen tres tipos de *flow objective*:

  - ***Filtering Objective***: Permite o deniega el encaminamiento de un tipo determinado de paquetes.
  - ***Forwarding Objective***: Representa una descripción de los tipos de tráfico que deben ser reenviados a través del *device*.
  - ***Next Objective***: Indica el siguiente *device* al que debe de ser enviado el paquete para llegar a su destino siguiendo la ruta especificada.

- ***Network Graph***: Esta abstracción proporciona a las aplicaciones una vista del estado actual de la red, permitiendo el acceso a todos los objetos que la modelan (*hosts*, *links*, *devices*, etc.) Además ofrece diversas funcionalidades, como por ejemplo calcular el camino más corto dados dos nodos, maximizar la utilización de la red gracias a los datos proporcionados por la monitorización de la misma, y aislar una parte de la red para que no reciba tráfico en el caso de que haya que desconectar los nodos por algún motivo.

2.2. Interacción con ONOS

ONOS ofrece tres vías para que un usuario (comúnmente un operador de red) pueda interaccionar con el servidor:

- ***GUI***: Interfaz gráfica vía *web* que recoje la visualización de la red controlada por ONOS.
- ***CLI***: Consola para introducir comandos directamente en la máquina en la que está corriendo ONOS.
- ***REST API***: Interfaz proporcionada por ONOS para la interacción mediante el protocolo *REST*.

2.2.1. Interfaz gráfica (GUI)

La interfaz gráfica de ONOS[11] es una página web que proporciona una interfaz visual del controlador ONOS. Para acceder a la *GUI* de ONOS habrá que acceder al siguiente enlace:

```
http://DIRECCION-IP-SERVIDOR-ONOS:PUERTO-ONOS/onos/ui
```

Lo primero que se debe de hacer es iniciar sesión con un usuario y contraseña válidos (por defecto es *onos/rocks*). Una vez iniciada sesión, aparece la ventana principal de la *GUI* de ONOS. Esta ventana está dividida en las siguientes secciones:

- Botón para acceder al menú de navegación. Este menú contiene las siguientes opciones:
  - Opción para acceder a las aplicaciones instaladas en el servidor ONOS.

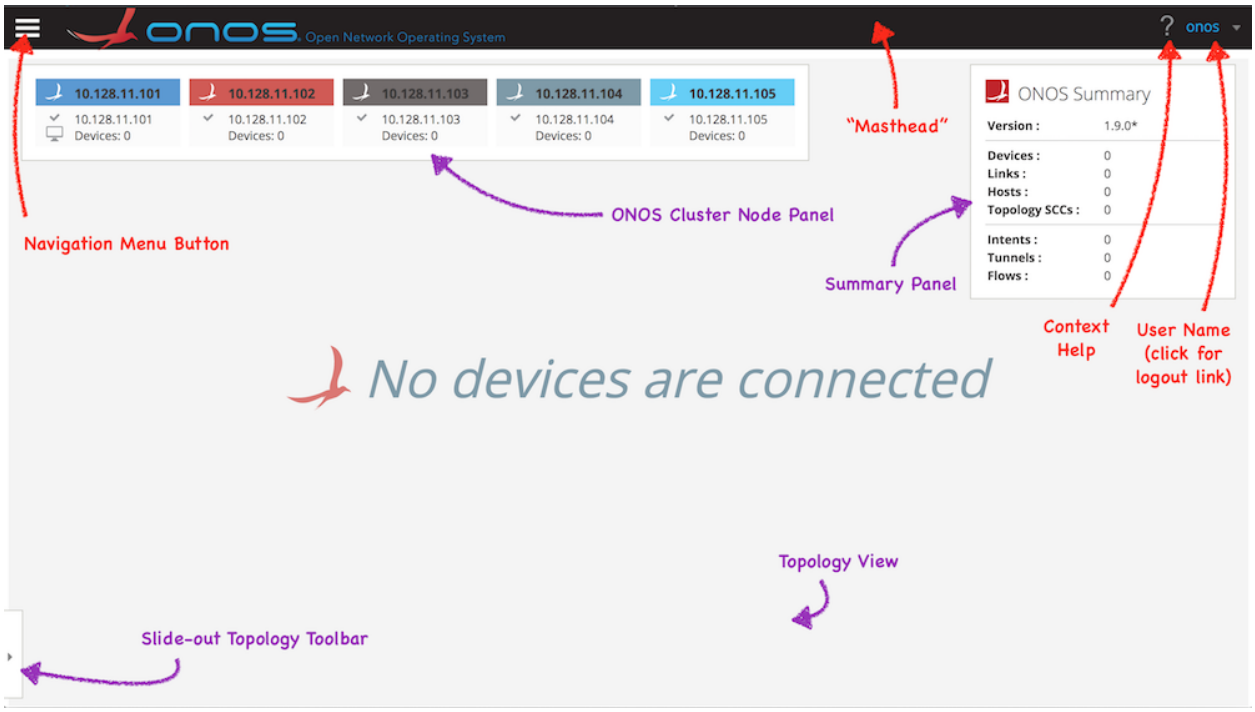

Figura 2.4: GUI de ONOS

- Menú de opciones para modificar los parámetros referentes al sistema.
- Opción para visualizar el *cluster* de nodos.
- Opción para acceder a las diferentes vistas de la topología.
- Opciones para ver la lista de *devices*, *links*, *hosts*, *intents* y *tunnels*.

- Una barra horizontal situada en la parte superior donde se encuentra el menú para cambiar de usuario.
- Cuadro con un resumen de los distintos componentes de la red visualizada.
- Una barra lateral oculta que permite modificar una serie de parámetros que afectan unicamente a la visualización.
- En la zona central es dónde se representa el grafo de la topología

2.2.2. Consola de comandos (CLI)

La consola de comandos de ONOS[12], basada en *Karaf*, permite al usuario ejecutar comandos directamente en la máquina en la que está corriendo el servidor ONOS. Para acceder a ella basta con ejecutar el siguiente comando:

```
onos [DIRECCIÓN-IP-SERVIDOR-ONOS]
```

Una vez se ha accedido a la *CLI*, se podrán ejecutar una serie de comandos propios de ONOS que permiten tanto obtener información sobre la red como añadir o eliminar ciertos elementos, como *intents* o *flows*. Para acceder a la lista de comandos completa que ofrece ONOS basta con introducir el comando *help onos*.

```
onos> help onos
COMMANDS
onos:add-host-intent                Installs host-to-host connectivity intent
onos:add-multi-to-single-intent     Installs connectivity intent between multiple ingress d
onos:add-optical-intent             Installs optical connectivity intent
onos:add-point-intent               Installs point-to-point connectivity intent
onos:add-protected-transport        Adds ProtectedTransportIntent
onos:add-single-to-multi-intent     Installs connectivity intent between a single ingress d
onos:add-test-flows                 Installs a number of test flow rules - for testing only
onos:add-vnet-intent                Installs virtual network connectivity intent
onos:allocations                    Lists allocated resources
onos:annotate-device                Annotates network model entities
onos:annotate-link                  Annotates network model entities
onos:annotate-port                  Annotates port entities
onos:app                            Manages application inventory
onos:app-ids                        Lists application ID information
onos:apps                           Lists application information
onos:balance-masters                Forces device mastership rebalancing
onos:cfg                            Manages component configuration
onos:cluster-devices                Lists devices of the specified topology cluster in the
onos:cluster-links                  Lists links of the specified topology cluster in the cu
onos:clusters                       Lists all clusters in the current topology
onos:config-link                    Configure link.
onos:config-link-discovery          Adds configuration to disable LLDP link discovery
onos:counter                        Displays the current value of a atomic counter
onos:counters                       Lists information about atomic counters in the system
onos:cycle-intents                  Installs random intents to test throughput
onos:device-add-interface           Configures a device interface
onos:device-configuration           [Deprecated]Gets the configuration of the specified typ
onos:device-controllers             gets the list of controllers for the given infrastructu
onos:device-interfaces              Lists all interfaces or interfaces of a device.
onos:device-key-add                 Adds a device key. Adding a new device key with the sam
onos:device-key-remove              Removes a device key
onos:device-keys                    Lists all device keys
onos:device-ports                   [Deprecated]Gets the ports of the specified device.
onos:device-remove                  Removes an infrastructure device
onos:device-remove-interface        Removes an interface configuration from a device
onos:device-role                    Sets role of the controller node for the given infrastr
onos:device-setconfiguration        [Deprecated]Sets the configuration of the specified fil
onos:device-setcontrollers          sets the list of controllers for the given infrastructu
onos:devices                        Lists all infrastructure devices
```

Figura 2.5: Lista de comandos de ONOS (incompleta)

2.2.3. REST API

El servidor *ONOS* cuenta con una extensa *REST API* generada en *Swagger* que ofrece las mismas funcionalidades que se pueden realizar en la CLI mediante el uso de peticiones *CRUD* (las siglas vienen de crear, leer, actualizar y eliminar). Esta *REST API* se puede consultar en el siguiente enlace:

`http://DIRECCION-IP-SERVIDOR-ONOS:PUERTO-ONOS/onos/v1/docs`

CAPÍTULO 3

Arquitectura del plugin Net2Plan-ONOS

En este capítulo se detallará el desarrollo del plugin *Net2Plan-ONOS*, que consistió en los siguientes pasos:

- Generación del cliente *REST/API* para la conexión con un servidor *ONOS* desde *Java*.
- Implementación de un plugin para *Net2Plan* con la integración del cliente anteriormente citado, así como las mejoras en la interfaz gráfica necesarias para la correcta visualización de los datos obtenidos por el cliente.

3.1. Arquitectura de plugins de Net2Plan

Dado su carácter modular, *Net2Plan* permite la implementación de *plugins* que permiten extender la herramienta de manera simple y poco intrusiva. Para ello, únicamente habrá que crear un proyecto *Maven* con las dependencias necesarias para su correcto funcionamiento incluidas en su propio *POM*, y una clase que implemente la clase abstracta *IGUIModule*.

La clase *IGUIModule*, situada en el paquete *com.net2plan.internal.plugins*, está compuesta por los una serie de métodos abstractos, que habrá que sobreescribir con las funcio-

nalidades que se deseen proporcionar. Los principales son los siguientes:

- ***start()*:** Especifica el comportamiento a realizar para arrancar el plugin.
- ***stop()*:** Especifica el comportamiento a realizar para parar correctamente el plugin.
- ***configure()*:** Este método recoge la configuración de la interfaz gráfica del plugin una vez iniciado. Los plugins comúnmente están formados por dos ventanas, una para la representación gráfica de la topología de red, y otra en la que se pueden consultar toda la información relativa a los elementos que componen la red, organizados en tablas.
- ***addKeyCombinationAction()*:** En este método aparecen las combinaciones de teclado ofrecidas al usuario para facilitar el uso de la herramienta, junto a las acciones asignadas a cada combinación.

Una vez completado el proceso de implementación del plugin, se deberá compilar el proyecto *Maven* para obtener el *.jar* que contiene las clases que componen el plugin y sus dependencias. Este *.jar* habrá que depositarlo en la carpeta *Net2Plan-X.X.X/plugins*. Al ejecutar *Net2Plan* mediante el fichero *Net2Plan.jar*, podremos acceder al plugin mediante el menú *Tools*.

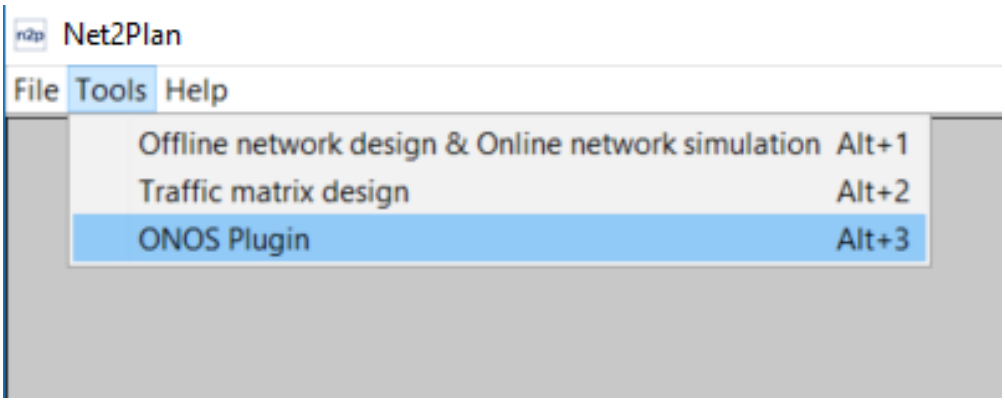

Figura 3.1: Menú Tools de Net2Plan

3.2. Generación automática del cliente REST/API

La comunicación entre un usuario y el servidor ONOS se puede realizar de diferentes maneras. Por ejemplo, se puede acceder a los datos de la topología que controla ONOS mediante su interfaz gráfica, crear o eliminar *intents* a través de comandos en la consola de *Karaf*, o bien realizar todas estas acciones mediante su *REST/API*.

Para interaccionar con el servidor ONOS desde el plugin programado en *Java*, lo más interesante es hacerlo mediante la *REST/API*. Lo único que habría que hacer es implementar un cliente capaz de comunicarse mediante operaciones *CRUD* (*create, retrieve, update, delete*), y procesar la respuesta del servidor. *Swagger* te ofrece un cliente *REST/API* a medida de manera simple, realizando un modelado automático de los modelos de datos definidos en el servidor para facilitar su lectura en *Java*.

A continuación, se detallarán los pasos realizados para la obtención e integración de un cliente *REST/API* para ONOS generado en *Swagger*:

Lo primero que se debe realizar es obtener la definición en *Swagger* del servidor con el que se desee comunicar, en este caso ONOS. Debido a que la *REST/API* de ONOS fue generada mediante *Swagger*, podemos tener acceso a su definición, lo que facilita la generación del cliente. En caso contrario no se podría obtener el cliente mediante *Swagger* (a no ser que la definición del servidor se confeccione a mano). Para obtener la definición del servidor ONOS se debe acceder a la siguiente *URL*:

```
IP-SERVIDOR:PUERTO/onos/v1/docs/apis/onos/v1/swagger.json
```

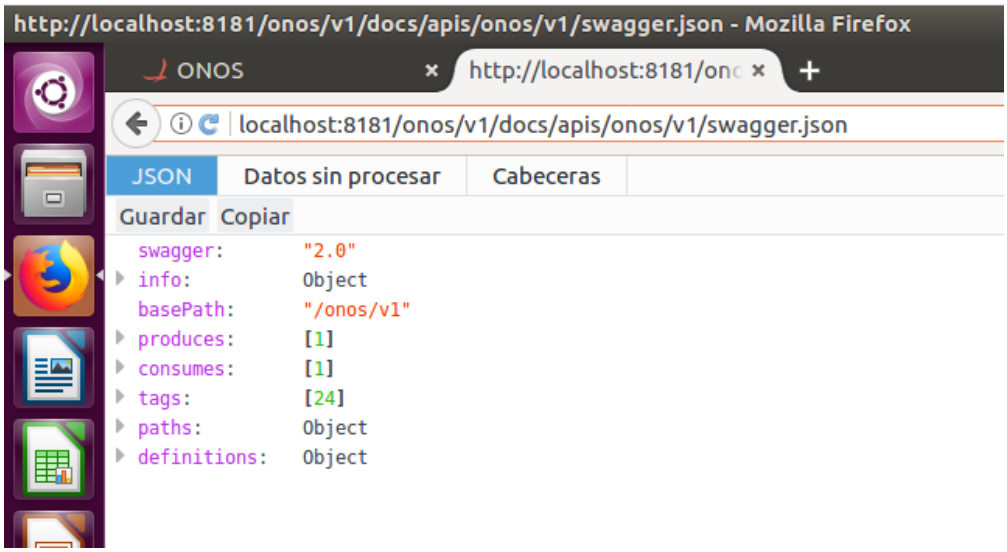

Figura 3.2: Definición Swagger del servidor ONOS

El siguiente paso es descargar la definición en formato *JSON* para importarlo en el editor de *Swagger*. Esto se puede realizar vía web sin necesidad de descargar ningún *software*, mediante la *URL* `https://editor.swagger.io/`. Una vez dentro, para importar el *JSON* descargado con la definición del servidor se hará mediante *File* - *Import File*. Se deberá visualizar lo siguiente:

Como se puede observar en la figura anterior, aparecen una serie de errores en la parte derecha que imposibilitan la creación automática del cliente. Esto es debido a que el

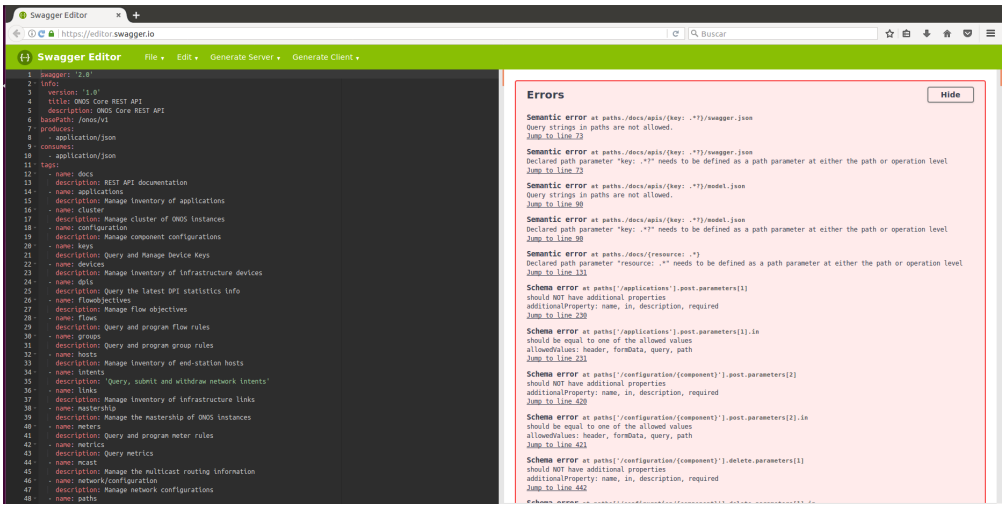

Figura 3.3: Importación de la definición del servidor ONOS en el editor de Swagger

servidor *REST/API* de ONOS fue generado con una versión muy antigua de *Swagger*. Para la generación del cliente se debieron corregir todos y cada uno de los errores, y realizar unas pequeñas modificaciones. A continuación se detallará paso a paso este proces:

- Se identificó un error común que aparecía en la definición de las *URL* relativas de cada objeto. Cuando se quería especificar un valor variable en la *URL* (por ejemplo, el *ID* de un *device*), se utilizaba la cadena de caracteres "*{key: .*?}*". Para solucionar los errores de este tipo hubo que sustituirla por lo siguiente: "*{key}*".
- Sustitución de los tipos de datos *int64*, *int32*, *uint16* o *byte* por *integer*.
- Sustitución de los tipos de datos *Hex16* por *string*.
- Sustitución de los tipos de datos *String* por *string*.
- Simplificación de la definición del servidor, eliminando parte de los bloques con errores. Debido a que la mayoría de métodos *POST* contenían errores, se decidió eliminar todos estos métodos, ya que el plugin sólo iba a realizar tareas de lectura.
- Añadir la etiqueta *host* con la *IP* y el puerto del servidor ONOS (en este caso "*host: 'localhost:8181'*".
- Añadir la etiqueta *securityDefinitions* y *security*, que definen la autenticación que tiene que realizar el cliente para acceder al servidor, con el siguiente contenido:

```
securityDefinitions:
   onos_auth:
      type: basic
security:
   - onos_auth: []
```

Una vez corregidos todos los errores, el resultado debería ser el siguiente:

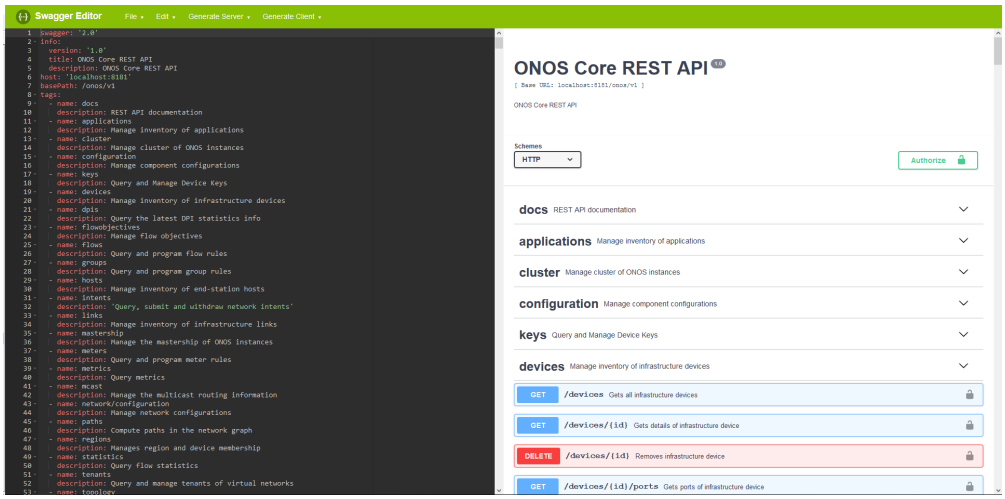

Figura 3.4: Corrección de los errores del servidor ONOS en Swagger

En este punto ya se podría generar el cliente. Para ello, en el menú *Generate Client*, se debe elegir el lenguaje de programación en el que se quiera que esté programado, en este caso *Java*. Se generará una carpeta comprimida con todas las clases que componen el cliente. En la siguiente sección se explicará la composición de estas clases y su integración en el plugin *Net2Plan-ONOS*.

3.3. Módulos del plugin

En esta sección se detallará el comportamiento de los principales módulos que componen el software desarrollado para la elaboración del plugin *Net2Plan-ONOS*. Cabe destacar que el plugin desarrollado parte del plugin *networkDesign* que ofrece *Net2Plan* en su versión *0.5.2*. De este modo se han aprovechado las funcionalidades que ya ofrece de por sí dicho plugin.

El desarrollo llevado a cabo en este proyecto ha consistido en:

- Simplificar el plugin *networkDesign* eliminando funcionalidades innecesarias, como por ejemplo el soporte multicapa o el sistema de algoritmos y de informes.
- Añadir las funcionalidades referentes a *ONOS*.
- Agilizar el sistema de tablas para facilitar la creación de nuevas tablas.
- Añadir un *wrapper* que actúa de intermediario entre *ONOS* y *Net2Plan*.

3.3.1. Módulo *io.swagger.client*

Este módulo contiene el cliente generado por *Swagger* para facilitar la comunicación con el servidor ONOS. El cliente está dividido en cuatro paquetes, que contienen las clases necesarias para realizar lecturas y escrituras en el servidor a través de objetos modelados automáticamente por *Swagger*. Estos paquetes son:

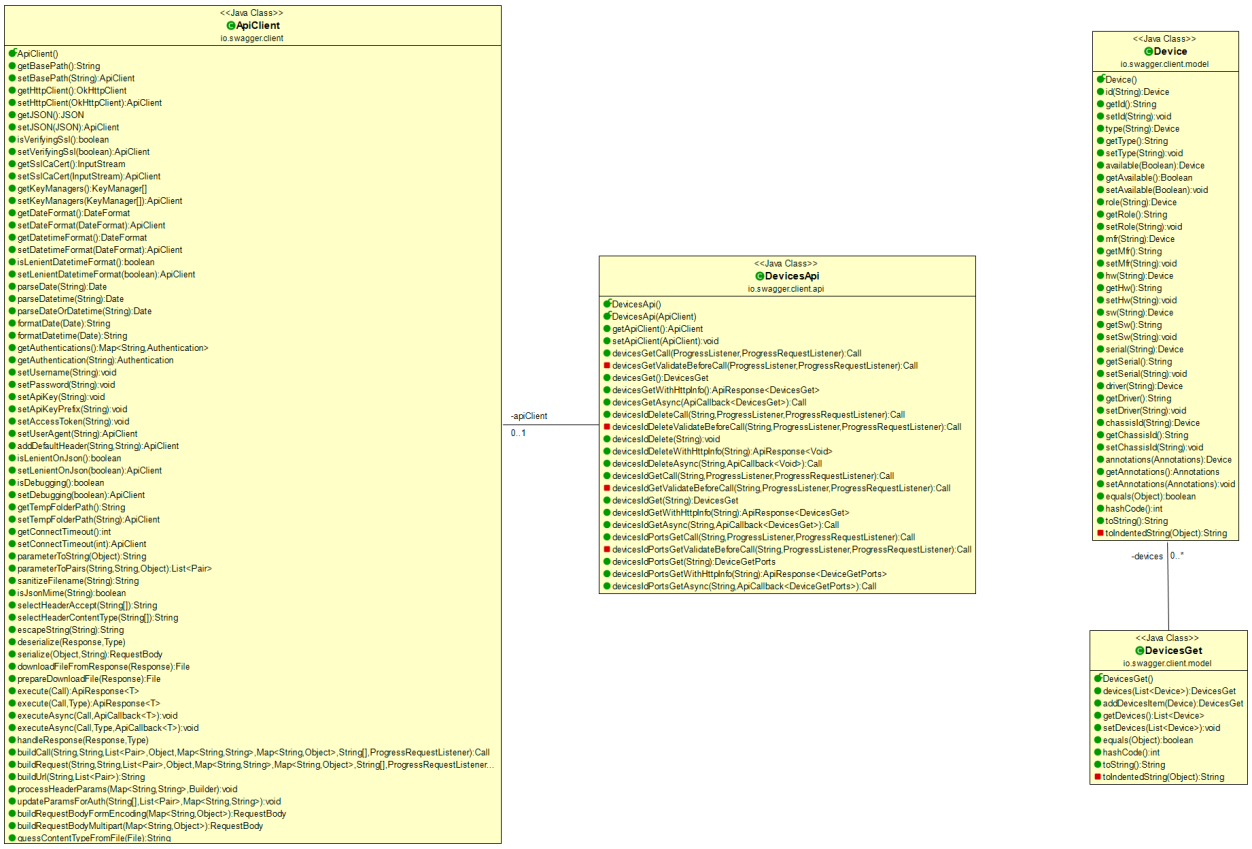

Figura 3.5: Diagrama de clases simplificado del módulo io.swagger.client (sólo devices)

- ***io.swagger.client*:** Paquete principal que engloba las clases que manejan la comunicación con el servidor. Destaca la clase *ApiClient.java*, que establece una comunicación *HTTP* con la dirección *IP* y puerto apropiado. Además, incluye otras clases para el manejo de excepciones, utilidades que facilitan la interacción con el cliente y una clase para trabajar con *JSON*, que es el lenguaje en el que están escritas las respuestas del servidor.

- ***io.swagger.client.api*:** Este paquete incluye las clases que habrá que utilizar para obtener cada tipo de modelo de datos específico. Todos los objetos de interés incluidos en ONOS (*devices*, *links*, *intents*, etc.) tienen una clase específica que modela su API. A través de estas clases se puede realizar cualquier petición *CRUD*. Por ejemplo, a través de la clase *DevicesApi.java* se pueden obtener toda la lista de *devices* de la topología que controla ONOS, obtener el *device* dado un *ID* (o incluso eliminarlo).

- ***io.swagger.client.model*:** Definen los objetos *Java* con los que trabajan las APIs del paquete anterior. Para cada objeto se definen todos los atributos tal y como aparecen en *ONOS*, en el tipo de datos adecuado. Además, incluyen los métodos *get* para obtener dichos atributos. También aparecen los métodos *set* de cada atributo, pero al generar un cliente simplificado las peticiones *POST* realizadas al servidor no están implementadas.

- ***io.swagger.client.auth*:** Contienen las clases que garantizan la seguridad de la comunicación entre cliente y servidor. Cabe destacar que para que esta comunicación se realice correctamente, el cliente debe proporcionar un usuario y contraseña para tener acceso al servidor. Esta autentificación la controla la maneja *HttpBasicAuth.java* utilizando la técnica de control de acceso Basic Authentication (BA), que es la seguridad más simple que ofrece *HTTP*.

3.3.2. Módulo *com.net2plan.onos.informationModel*

El módulo *informationModel* compone el nexo de unión entre el cliente *Swagger* y *Net2Plan*. En otras palabras, contiene las clases que realizan la conversión del modelo de datos de ONOS (definidos en el paquete *io.swagger.client.model*) al modelo de datos de *Net2Plan*.

Para ello se ha definido un *wrapper* que asigna a cada entidad de ONOS una entidad de *Net2Plan*, de manera transparente para el usuario. De esta manera se aprovechan las ventajas que ofrece de por sí *Net2Plan*, como puede ser el guardado y cargado de topologías, la ejecución de algoritmos e informes o la visualización de la información de la topología mediante tablas. Además, el *wrapper* está hecho a la medida de ONOS, por lo que ofrecerá única y exclusivamente las funcionalidades permitidas por el servidor. El *wrapper* está compuesto por las siguientes clases:

- ***OnosNet.java*:** Clase principal que contiene información de toda la topología. Es usado por todo el plugin *Net2Plan-ONOS* como medio para obtener información sobre el diseño. Además, a través de esta clase se tiene acceso a las siguientes funcionalidades:

  - Crear una nueva topología. La clase *OnosNet.java* ofrece tres opciones para crear un nuevo diseño: crear un diseño vacío, crear un diseño a partir de un fichero *XML* y crear un diseño a partir de la información obtenida por un servidor ONOS. Esta última opción se realiza por medio de la clase *TopologyCreator.java*, que será explicada más adelante en esta misma sección.
  - Guardar la topología actual en un fichero *XML*.
  - Obtener o modificar el nombre y la descripción de la topología.
  - Obtener la lista de elementos de red de un tipo determinado (*device*, *host*, *intent*, etc.).

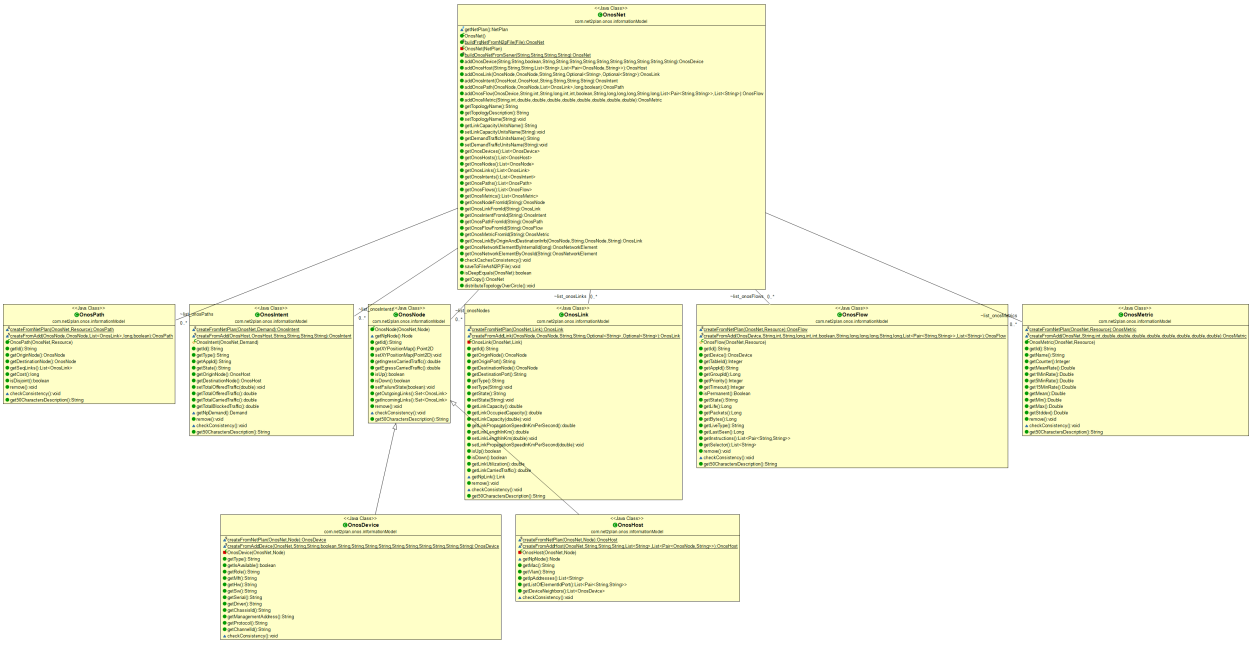

Figura 3.6: Diagrama de clases del módulo com.net2plan.onos.informationModel

- Obtener un elemento de red de un determinado tipo a partir de su *ID*.
- Crear un nuevo elemento a partir de los atributos que lo caracterizan.

Para ofrecer estas funcionalidades, la clase *OnosNet.java* trabaja internamente con un objeto *NetPlan*, que permanecerá oculto para el resto de clases.

- ***OnosNetworkElement.java*:** Esta clase abstracta será extendida por todas las clases que representen un elemento de red de ONOS. De esta manera se deberán implementar una serie de métodos que deben de ser comunes para cada elemento. Estos métodos son:

  - Métodos para leer, escribir o borrar atributos de usuario. Estos atributos están definidos en *Net2Plan* como una pareja de clave más valor, y serán guardados junto al resto de información de la topología.
  - Métodos para leer, escribir o borrar etiquetas de usuario. Al igual que los atributos, son información adicional asociadas a los elementos de red. En este caso sólo están definidos por una cadena de caracteres.
  - Método para borrar un elemento de red.
  - Método para obtener el índice y el *ID* interno de cada elemento usado por *Net2Plan*. De nuevo, esta clase trabaja con una clase análoga de *Net2Plan*, en este caso *NetworkElement*. Por ello, a cada elemento de red de ONOS definido por el *wrapper* se le deberá asignar un elemento de red de *Net2Plan* que extienda la clase *NetworkElement*.

- ***OnosNode.java*:** Esta clase abstacta representa un nodo de ONOS, que puede ser un *device* o un *host*. Está asociada a un objeto *Node* de *Net2Plan*. Extiende la clase *OnosNetworkElement.java*, y tiene las siguientes características que son comunes a los *devices* y *hosts* de ONOS:
  - *ID* único del nodo en ONOS, representado por una cadena de caracteres.
  - Mapa de coordenadas x e y utilizado para su representación en la interfaz gráfica. Es modificable por el usuario.
  - Tráfico cursado entrante y saliente.
  - Enlaces entrantes y salientes.
  - Métodos para conocer el status del nodo (si está activo o desactivo).

- ***OnosDevice.java*:** Representa a un *device* presente en la red de ONOS. Extiende la clase *OnosNode.java*, por lo que está asociado a un objeto *Node* de *Net2Plan* y está definido por un *ID*. Contiene los siguientes atributos leídos por el cliente *Swagger*, todos ellos con sus métodos *get* correspondientes:
  - ***deviceType*:** En ONOS, un *device* puede representar varios tipos de equipos de red, como por ejemplo *switches*, *routers*, *ROADMs*, etc. Este atributo indica de qué tipo se trata cada *device*.
  - ***isAvailable*:** Un valor booleano que indica si el *device* está disponible o no.
  - ***deviceRole*:** El rol del *device*
  - ***deviceMfr*:** Contiene el fabricante del *device* (ej: *Nicira*).
  - ***deviceHw*:** Indica la versión de *hardware* del *device* (ej: *Open vSwitch*).
  - ***deviceSw*:** Indica la versión de *software* del *device*.
  - ***deviceSerial*:** Contiene el número de serie del *device*.
  - ***deviceDriver*:** Indica el *driver* del *device*.
  - ***deviceChassisId*:** Contiene el *ID* del *chassis* contenido en el *device*.
  - ***deviceManagementAddress*:** Indica la dirección (*IP* + puerto) de gestión del *device*.
  - ***deviceProtocol*:** El protocolo usado por el *driver*, como por ejemplo *OpenFlow*.

- ***OnosHost.java*:** Representa a un *host* de ONOS. Es similar a la clase *OnosDevice.java* (su equivalencia en *Net2Plan* es un objeto de tipo *Node* y tiene un *ID* único). Sus atributos característicos son:
  - ***hostMac*:** Indica la dirección *MAC* del *host*.
  - ***hostVlan*:** Indica la *VLAN* del host.
  - ***hostIpAddresses*:** Lista de direcciones *IP* definidas en el *host*.

- ***hostDeviceNeighbours*:** Lista con la información de los *device* vecinos del *host*.

- ***OnosLink.java*:** Esta clase modela los enlaces unidireccionales presentes en ONOS. Sus atributos característicos son:
  - ***originNodeId*:** *ID* del *device* origen del enlace.
  - ***originPort*:** Puerto origen.
  - ***destinationNodeId*:** *ID* del *device* destino del enlace.
  - ***destinationPort*:** Puerto destino
  - ***linkType*:** Tipo de enlace. El enlace puede representar un enlace directo, un enlace indirecto, un enlace óptico, un túnel *MPLS*, etc.
  - ***linkState*:** Estado del enlace (activo o inactivo).

  En ONOS, sólo se consideran los enlaces entre dos *devices*. Por tanto, los enlaces entre *hots* y *device* se crearán a partir del atributo *hostDeviceNeighbours* de cada *host* (los atributos *linkType* y *linkState* de estos enlaces estarán vacíos) . Su objeto *Net2Plan* análogo es *Link*. A través de él se tiene acceso a los siguientes parámetros:
  - Velocidad de propagación medida en kilómetros por segundo.
  - Longitud del enlace medida en kilometros.
  - Capacidad.
  - Capacidad ocupada.
  - Tráfico cursado.
  - Utilización.
  - Valor booleano que indica si el enlace está caído o no.

- ***OnosIntent.java*:** Equivalente en el plugin a la entidad *intent* de ONOS. Dado que un *intent* está caracterizado principalmente por un nodo origen y un nodo destino (ambos de tipo *host*), está asociado al objeto *demand* de *Net2Plan*. Sus atributos son:
  - ***intentId*:** *ID* del *intent*.
  - ***originHostId*:** *ID* del *host* origen.
  - ***destinationHostId*:** *ID* del *host* destino.
  - ***intentAppId*:** *ID* de la aplicación asociada al *intent* (ej: *org.onosproject.ovsdb*).
  - ***intentState*:** Estado del *intent*. Un *intent* puede encontrarse en los siguientes estados:
    - *COMPILING*.
    - *FAILED*.
    - *INSTALL_REQ*.

    - *INSTALLED*.
    - *INSTALLING*.
    - *RECOMPILING*.
    - *WITHDRAW_REQ*.
    - *WITHDRAWING*.
    - *WITHDRAWN*.
  - ***intentPriority*:** Prioridad del *intent*. Valor entero entre 1 (prioridad más baja) y 65535 (prioridad más alta).
  - ***intentResources*:** Listado de los recursos requeridos por el *intent*.

- ***OnosFlowRule.java*:** Equivale a una *flow rule* (regla de flujo) de ONOS. En *Net2Plan* no hay una equivalencia exacta con este tipo de elemento de red, por lo que se utiliza el objeto genérico *resource*. Un *resource* en *Net2Plan* pertenece a un objeto de tipo nodo (aunque en la nueva versión de *Net2Plan* 0.6.0 se permite la posibilidad de que un *resource* no esté asignado a ningún nodo). Como una *flow rule* en ONOS está asociado a un *device*, en este caso el *resource* pertenecerá al objeto *node* asociado a dicho *device*. Los principales atributos de esta clase son:
  - ***flowRuleId*:** *ID* de la *flow rule*.
  - ***flowRuleDeviceId*:** *ID* del *device* donde se aplica la *flow rule*.
  - ***flowRuleAppId*:** Contiene el *ID* de la aplicación asociada a la *flow rule*.
  - ***flowRulePriority*:** La prioridad asignada a la *flow rule*. Al igual que los *intents*, a mayor valor mayor prioridad.
  - ***flowRuleTimeout*:** Valor entero que indica el tiempo de validez de la *flow rule*.
  - ***flowRuleIsPermanent*:** Booleano que indica la permanencia de la *flow rule*. Una *flow rule* será permanente si no tiene *timeout*.
  - ***flowRuleState*:** Indica el estado de la *flow rule*. Los estados posibles son los siguientes:
    - *ADDED*.
    - *FAILED*.
    - *PENDING_ADD*.
    - *PENDING_REMOVE*.
    - *REMOVED*.
  - ***flowRuleLife*:** Contiene el tiempo que ha transcurrido aplicada la *flow rule*, medido en segundos.
  - ***flowRulePackets*:** Indica el número de paquetes afectados por la *flow rule*.
  - ***flowRuleBytes*:** Similar al atributo anterior, pero con *bytes*.
  - ***flowRuleLastSeen*:** Indica cuando se consideró activa por última vez la *flow rule*.

  - ***flowRuleInstructions*:** Indica la lista de instrucciones para aplicar la *flow rule*.
  - ***flowRuleSelector*:** Define el tipo tráfico al que se deberá aplicar la *flow rule*.

- ***OnosPath.java*:** Esta clase modela las rutas o *paths* de ONOS. Vuelve a equivaler en *Net2Plan* a un *resource*, en este caso asociado al nodo origen de la ruta (que puede ser un *host* o un *device*). Sus atributos son:
  - ***pathId*:** *ID* de la ruta.
  - ***originId*:** *ID* del nodo origen de la ruta.
  - ***destinationId*:** *ID* del nodo destino de la ruta.
  - ***seqLinkIds*:** Lista ordenada con los *IDs* de los enlaces que definen la ruta. La lista de enlaces debe unir el nodo origen con el nodo destino de manera ininterrumpida.
  - ***pathCost*:** Valor decimal que contiene el coste asociado a la ruta.
  - ***pathIsDisjoint*:** Valor booleano que indica si la ruta es disjunta o no.

- ***OnosMetric.java*:** Esta clase modela las métricas de tráfico tomadas en la red de ONOS. Al igual que las rutas y las *flow rules*, está asociada a un *resource* de *Net2Plan*. Los atributos de este objeto son:
  - ***metricId*:** *ID* de la métrica.
  - ***metricName*:** Nombre de la métrica.
  - ***metricCounter*:** Número total de eventos registrados.
  - ***metricMeanRate*:** Tasa media de los eventos registrados.
  - ***metric1MinRate*:** Tasa total de los eventos registrados en el último minuto.
  - ***metric5MinRate*:** Tasa total de los eventos registrados en los últimos 5 minutos.
  - ***metric15MinRate*:** Tasa total de los eventos registrados en los últimos 15 minutos.
  - ***metricMean*:** El valor medio de todos los valores registrados por la métrica.
  - ***metricMin*:** El mínimo valor de la métrica.
  - ***metricMax*:** El máximo valor de la métrica.
  - ***metricStddev*:** La desviación estándar de todos los valores de la métrica.

- ***TopologyCreator.java*:** Clase encargada de construir la topología de *Net2Plan* a partir de la información obtenida por el servidor ONOS. Se accede a ella mediante el método estático *buildOnosNetFromServer* de la clase *OnosNet.java*. Este método instancia un objeto de la clase *TopologyCreator* a través de su único constructor, cuyos parámetros de entrada son la dirección *IP* y puerto del servidor ONOS y los datos necesarios para la autentificación (usuario y contraseña). Mediante estos parámetros

se establece una comunicación *HTTP* entre el *plugin* y el servidor, a través del cliente API generado por *Swagger*. Una vez establecida la comunicación, mediante el método *getOnosNet* se construye el objeto *OnosNet* que contiene la información de la topología, incluidos los objetos vistos anteriormente. Una vez concluido el proceso de creación, el objeto *OnosNet* resultante contiene todas las entidades de interés recogidas en el servidor ONOS con todos sus atributos, que se podrán visualizar de manera sencilla y amigable mediante tablas gracias al siguiente módulo.

3.3.3. Módulo *com.net2plan.gui.plugins.onosPlugin.viewEditTopolTables*

Módulo encargado de representar los datos de la topología en forma de tablas. Es un módulo que ya existe en el plugin *networkDesign* de *Net2Plan*, pero se le han realizado una serie de modificaciones para adaptarlo al resto de módulos del plugin *Net2Plan-ONOS*. Destacan las dos siguientes clases:

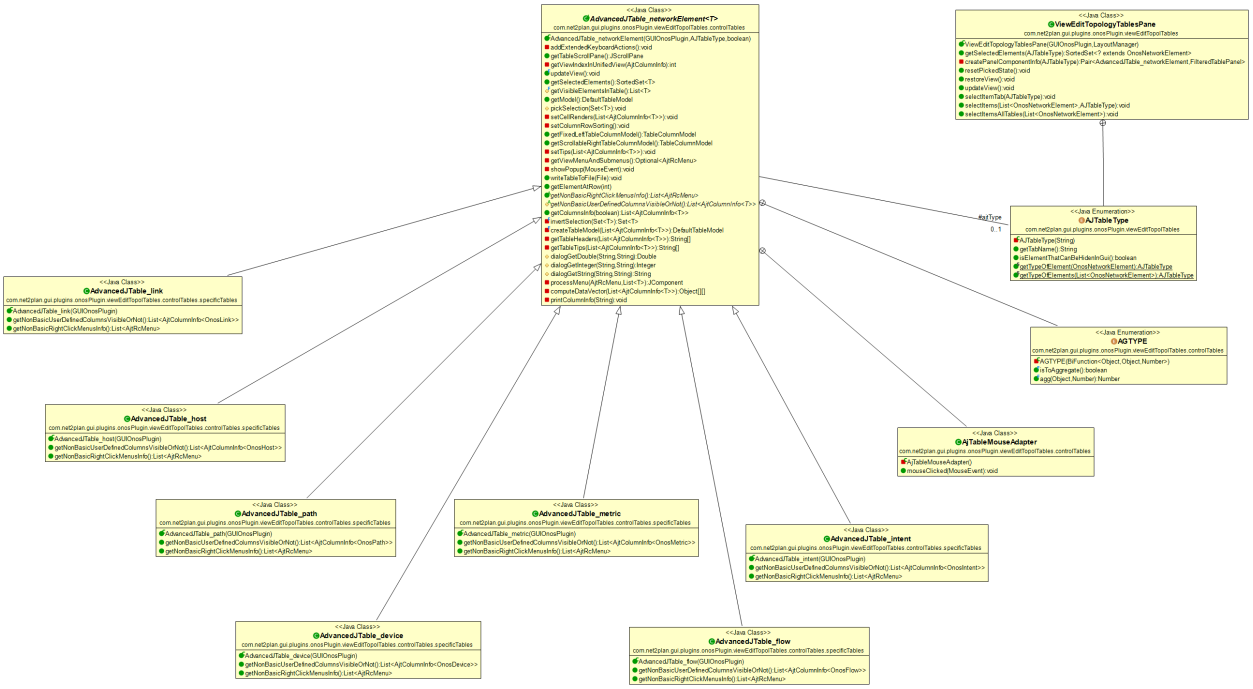

Figura 3.7: Diagrama de clases del módulo com.net2plan.onosPlugin.viewEditTopolTables

- ***ViewEditTopologyTablesPane.java*:** Clase encargada de construir la ventana en la que se pueden consultar las diferentes tablas. Esta ventana está compuesta por un sistema de pestañas, en el que a cada tipo de entidad de ONOS se le asigna una pestaña que contiene una tabla con los elementos de dicha entidad. Además, esta clase contiene los métodos encargados de actualizar las tablas para mostrar el contenido más reciente y gestionar la selección de filas (por ejemplo, seleccionar la fila que representa un elemento que ha sido seleccionado por el usuario en la interfaz gráfica).

- ***AdvancedJTable_networkElement.java*:** Clase abstracta que contiene todos los métodos comunes a todas las tablas creadas. En esta clase reside toda la complejidad del sistema de tablas, de manera que una vez implementada la creación de nuevas tablas es directa. Las funcionalidades que ofrece al usuario son las siguientes:
  - Funcionalidad para ordenar las columnas de manera ascendente o descendente.
  - Adición de una última fila que recoge la agregación de todos los valores de la columna.
  - Menú con opciones genéricas a todas las tablas accesibles mediante click derecho.
  - Opción de exportar el contenido de las tablas en un fichero Excel.
  - Funcionalidades accesibles mediante combinaciones de teclado. Por ejemplo, se pueden seleccionar una o varias filas y al pulsar la tecla *enter* se resaltan en la topología.
  - Resaltar filas o celdas de un color determinado al cumplirse una condición. Por ejemplo, pintar en rojo la fila que contiene los elementos caídos de la red.

- ***AdvancedJTable_XXXX.java*:** Clases que extienden la clase abstracta *AdvancedJTable_networkElement.java* y que representan una entidad ONOS determinada. Contiene las columnas específicas de cada tabla, y los métodos *get/set* de cada columna. Además, se debe de definir las opciones de click derecho específicas de cada tabla, así como el objeto de tipo *OnosNetworkElement* al que representa la tabla.

CAPÍTULO 4

Descripción de las funcionalidades desarrolladas

En este capítulo se detallará un caso de uso del plugin *Net2Plan-ONOS*. Aunque la configuración del servidor ONOS no entra dentro de los objetivos de este proyecto, la primera sección de este capítulo recogerá el estado del servidor que será consultado por el plugin. En segundo lugar, se verá el proceso realizado en la interfaz gráfica del plugin para obtener y visualizar la información de dicho servidor.

4.1. Configuración del servidor ONOS

Para el desarrollo de este proyecto se ha utilizado una máquina virtual con el sistema operativo *Ubuntu 17.10* en su versión de 64 bits. Sobre dicha máquina virtual se ha instalado un servidor ONOS para controlar una red virtual instanciada en *Mininet*.

4.1.1. Configuración de Mininet

La red que ha servido como ejemplo para visualizar los resultados de este proyecto ha sido una red de tipo árbol compuesta por 8 *hosts* y 7 *switches*. Para la obtención de esta topología se ejecutó el siguiente comando:

```
mn --controller=remote --mac --topo tree,3
```

Cabe destacar que para poder visualizar un *host* en el servidor ONOS este debe de inyectar tráfico a la red, por lo que se recomienda introducir el comando *pingall* en *Mininet*.

```
sdn@sdn-VirtualBox:~$ sudo mn --controller=remote --mac --topo tree,3
[sudo] password for sdn:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6)
 (s5, s7) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 X h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 1% dropped (55/56 received)
mininet>
```

Figura 4.1: Comandos ejecutados en Mininet

4.1.2. Configuración de ONOS

Una vez configurado *Mininet* y puesto en marcha el servidor ONOS, se puede ver la información referente a la topología puesta en marcha. ONOS ofrece varias vías para la visualización de esta información.

A través de la *GUI* de ONOS accesible vía web, se puede observar de manera visual la información completa de la topología.

Otra manera de visualizar la información es mediante comandos a través de *Karaf*. Existen comandos para visualizar información, o para añadirla.

En la configuración realizada en el servidor ONOS se han creado un total de 6 *intents* para su visualización en el plugin mediante el siguiente comando:

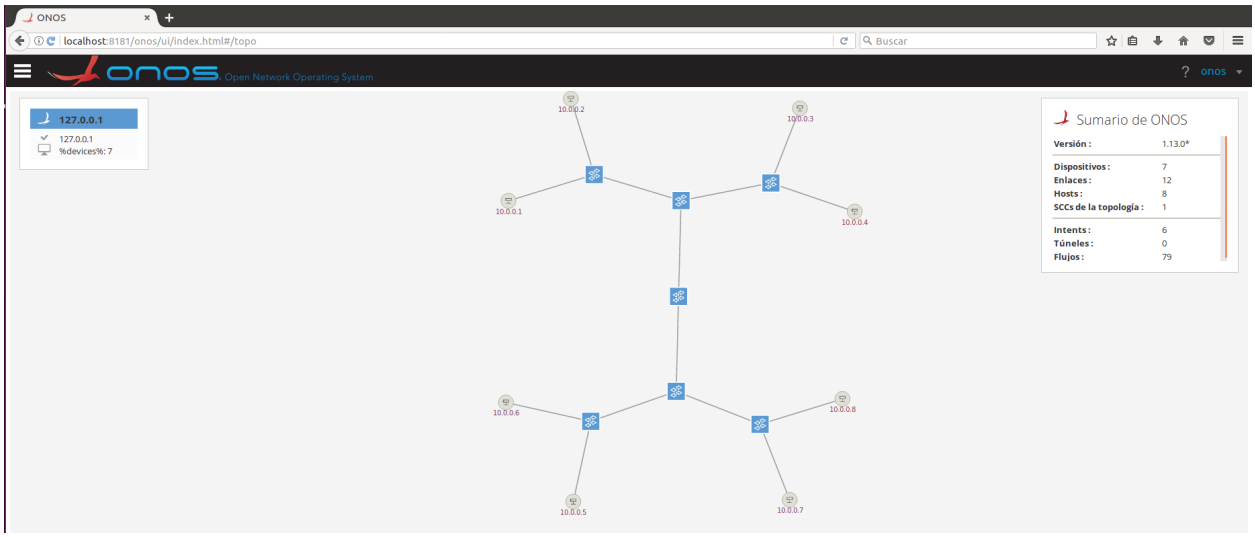

Figura 4.2: GUI de ONOS

```
add-host-intent [ID-HOST-ORIGEN] [ID-HOST-DESTINO]
```

4.2. Manual de usuario del plugin *Net2Plan-ONOS*

Partiendo de la configuración del servidor ONOS realizada en la sección anterior (o cualquier otra configuración), en esta sección se obtendrá la topología controlada por el servidor ONOS a través del plugin *Net2Plan-ONOS*.

En primer lugar, se ha de obtener el fichero *.jar* que contiene los *.class* que componen el plugin y sus dependencias. Para ello se debe compilar el proyecto *Maven*. Una vez obtenido, se coloca el *.jar* en la carpeta *plugins* de la versión compilada de *Net2Plan*. De esta forma el plugin será accesible al ejecutar *Net2Plan* mediante el ejecutable *Net2Plan.jar*.

Una vez ejecutado, se puede acceder al plugin *Net2Plan-ONOS* mediante el menú *Tools* tal y como se ve en la figura 3.1.

La interfaz gráfica del plugin consta de dos ventanas, una de ellas para la visualización gráfica de la topología y otra para las tablas. En la primera ventana se encuentran además una barra de botones que ofrecen diferentes funcionalidades, como cargar/guardar topología, hacer *zoom*, ampliar/reducir el tamaño de nodos y enlaces, etc. Destaca el botón *Connect*, que permite realizar la comunicación con un servidor ONOS.

Al hacer click a este botón aparece un panel que permite introducir la *IP* del servidor, el puerto en el que corre ONOS, el usuario y la contraseña de acceso.

Figura 4.3: CLI de ONOS

Al pulsar en el botón *Go!* se realiza la comunicación entre el cliente y el servidor. Si no se produce ningún error en la comunicación, se podrá visualizar la topología obtenida.

La ventana *ONOS Plugin - Information model tables* está compuesta por la pestaña *View/Edit network state*. Esta pestaña contiene a su vez subpestaña para cada tipo de modelo de red. En cada subpestaña aparecerá una tabla con los elementos de ese tipo específico presentes en la topología. Estas subpestañas son:

- ***Devices*:** Contiene los nodos de tipo *device* de la topología. Sus columnas editables son:
  - *Show/hide*: Permite ocultar o mostrar un *device* en la representación gráfica de la topología.
  - *X-pos/Y-pos*: Permite modificar las coordenadas de un *device*. Otra forma de modificar la posición de un nodo es arrastrando el nodo en la ventana de representación mientras se pulsa la tecla *Alt*.

  Además, otra funcionalidad que ofrece esta tabla es la de realizar la selección de uno o más *devices*, que consiste en resaltar el elemento o elementos seleccionados en la visualización gráfica de la topología. Esto se consigue haciendo doble click en la fila del nodo que se desea resaltar, realizando una selección múltiple y pulsar la tecla *enter*, o mediante la opción de click derecho *Pick selection*. Cabe destacar que esta selección se puede realizar a la inversa, haciendo click en un elemento de la visualización gráfica, se selecciona la subpestaña que contiene dicho elemento y se resalta la fila donde se encuentra.

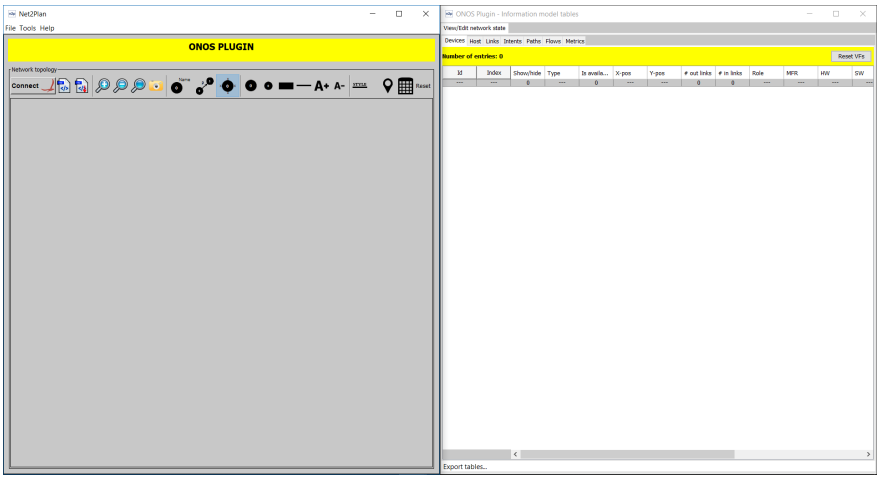

Figura 4.4: Ventana principal del plugin Net2Plan-ONOS

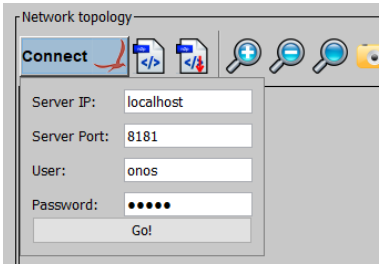

Figura 4.5: Botón Connect

- ***Hosts*:** Contiene los nodos de tipo *host* de la topología. Sus funcionalidades son exactamente las mismas que las de la subpestaña *device*.

- ***Intents*:** En esta subpestaña aparece el listado de *intents* definidos en ONOS. Al igual que la subpestaña *Links*, las columnas que definen los nodos origen y destino de los *intents* contienen *IDs* de objetos de tipo nodo, que la pulsar en el contenido resaltan dichos objetos. Por definición, los nodos involucrados en un *intent* tienen que ser de tipo *host*.

- ***Links*:** Ofrece la lista de los enlaces presentes en la topología. Las columnas a destacar son:

  - *Show/hide*: Permite ocultar o mostrar un enlace en la representación gráfica de la topología.
  - *Origin node / Destination node*: Contiene los *IDs* de los nodos origen y destino del enlace, que pueden ser nodos de tipo *host* o *device* (un enlace no puede tener dos nodos de tipo *host*). Al hacer click en un *ID* se resalta el objeto al que representa.
  - *Capacity (Gbps)*: Contiene la capacidad del enlace medida en *Gbps*. Este valor es editable.

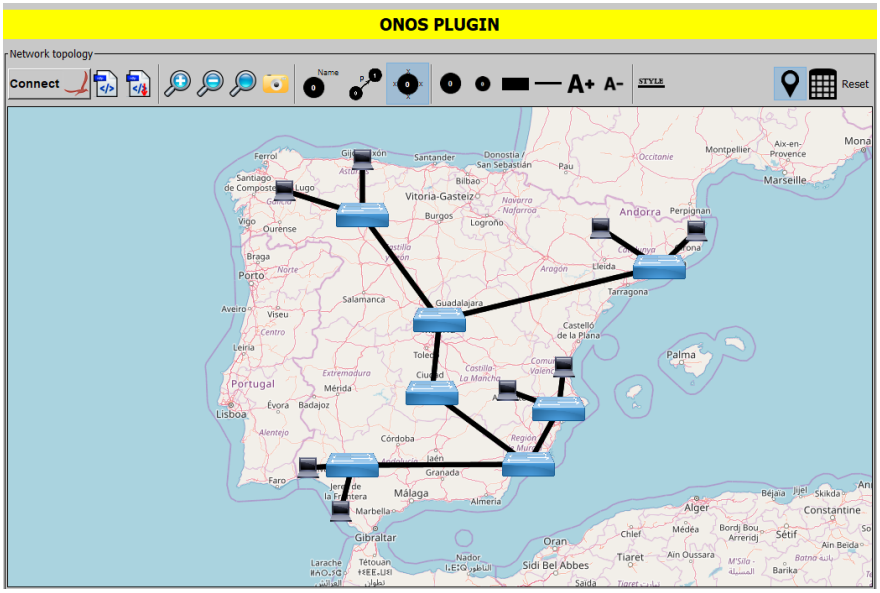

Figura 4.6: Representación gráfica de la topología obtenida

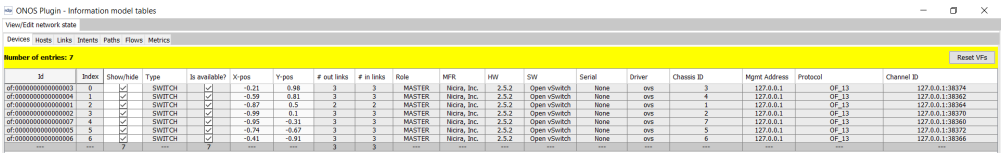

Figura 4.7: Subpestaña Devices

- *Length (km)*: Contiene la longitud del enlace medida en *km*. Editable por el usuario.
- *Propagation speed (km/s)*: Valor editable que contiene la velocidad de propagación del enlace medida en km/s.

Al igual que las subpestañas que contienen los objetos de tipo nodo, también se puede realizar la selección de enlaces mediante doble click, botón *enter* o opción de click derecho *Pick Selection*.

- ***Paths*:** Tabla con las rutas o *paths* recogidos por el servidor ONOS. Los *paths* no tienen columnas editables, pero sí columnas interactivas con los nodos origen y destino de cada entrada.

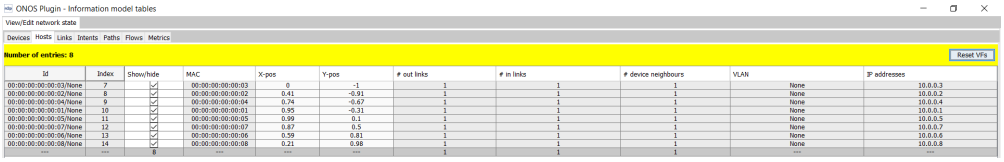

Figura 4.8: Subpestaña Hosts

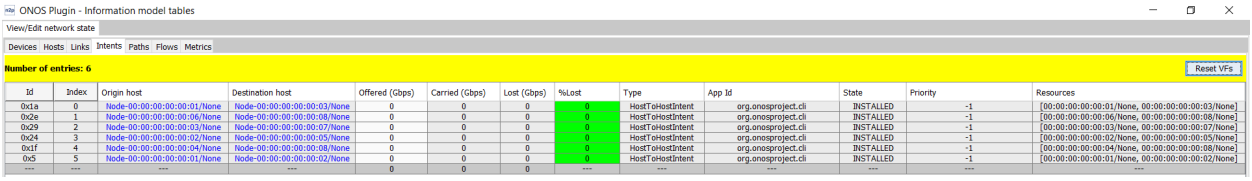

Figura 4.9: Subpestaña Intents

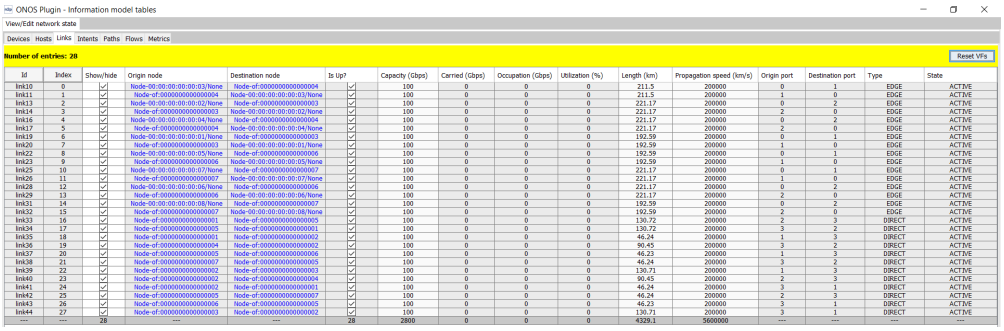

Figura 4.10: Subpestaña Links

- ***Flows*:** En esta subpestaña aparece toda la información relacionada con los flujos o *flows* de ONOS. En este caso sólo existe una columna interactiva, aquella que representa al *device* al que pertenece cada *flow*.

- ***Metrics*:** Por último, en la subpestaña *Metrics* aparece el listado de métricas definidas en ONOS.

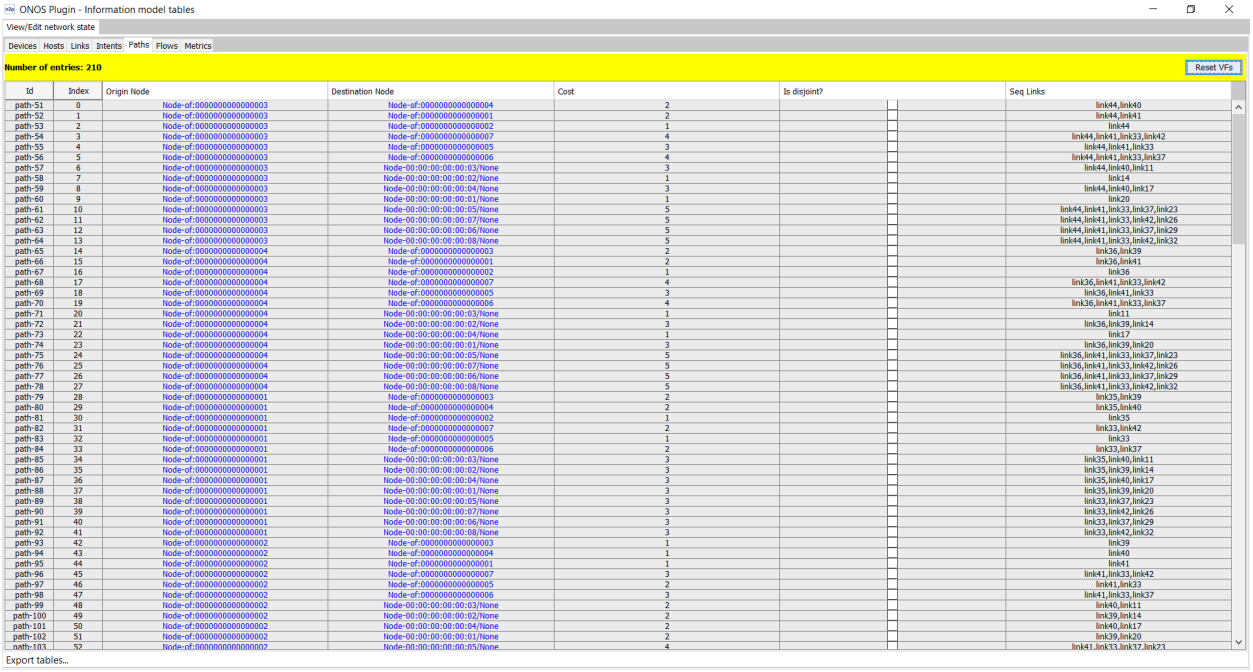

Figura 4.11: Subpestaña Paths

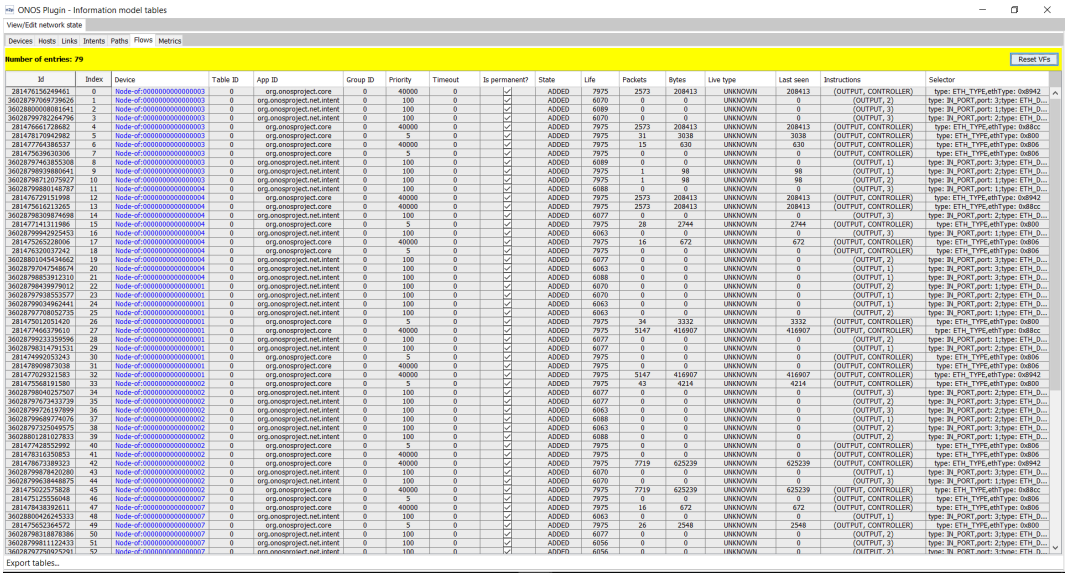

Figura 4.12: Subpestaña Flows

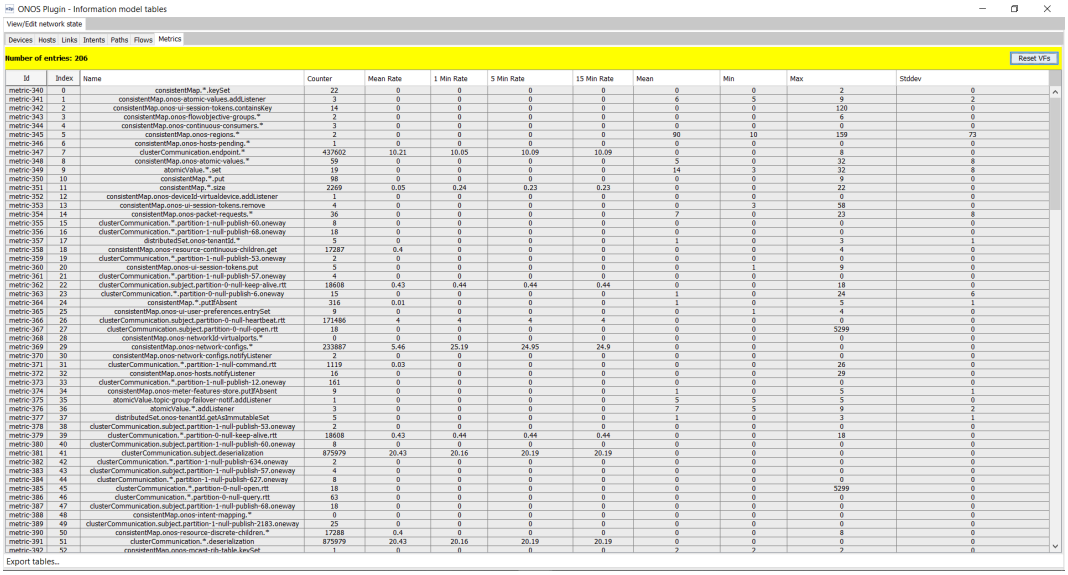

Figura 4.13: Subpestaña Metrics

CAPÍTULO 5

Conclusiones

El estudio realizado en este proyecto ha sido un buen punto de partida para el análisis e interacción con el controlador ONOS. Debido al auge de las soluciones SDN cada vez estará más presente en las redes actuales. El uso de *Net2Plan* como apoyo para la de optimización de las redes controladas por ONOS se presenta como valor de carácter añadido muy a tener en cuenta, dado su eficacia probada con anterioridad.

El objetivo primordial de este proyecto fue comunicar ambos *software*, por lo que el siguiente paso sería fortalecer esta comunicación. Hasta ahora el plugin es capaz de leer información del servidor ONOS, por lo que el siguiente paso será el de escribir en él. Por ejemplo, una evolución del plugin podría ser el de crear nuevos *intents*, crear *devices* para que sean controlados por ONOS (tanto reales como virtuales), o ejecutar algoritmos presentes ya en *Net2Plan* que optimicen el encaminamiento actual de la red y escriba las reglas de flujo calculadas en ONOS.

Otra línea futura sería la de añadir nuevas funcionalidades al plugin, tanto mejoras en la interfaz gráfica como añadir nuevas tablas para poder visualizarlas en el plugin que recojan información de, por ejemplo, las aplicaciones definidas en ONOS, los *drivers* o los *providers*.

CAPÍTULO 6

Acrónimos

API Application Programming Interface

BA Basic Authentication

CLI Command Line Interface

GUI Graphical User Interface

HTTP HyperText Transfer Protocol

IP Internet Protocol

JOM Java Optimization Modeler

JSON JavaScript Object Notation

MAC Medium Access Control

NFV Network Functions Virtualization

ODL OpenDaylight

ONOS Open Network Operating System

POM Project Object Model

REST Reepresentational State Transfer

SDN Software Defined Networking

URL Uniform Resource Locator

VLAN Virtual Local Area Network

XML Extensible Markup Language

Bibliografía

[1] "What's Software Defined Networking (SDN)? Definition," [Last accessed: October 2018]. [Online]. Available: https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/

[2] "The opendaylight platform | opendaylight," [Last accessed: October 2018]. [Online]. Available: https://www.opendaylight.org/

[3] "Onos - a new carrier-grade sdn network operating system," [Last accessed: October 2018]. [Online]. Available: https://onosproject.org/

[4] "Net2Plan - The open-source network planner," [Last accessed: October 2018]. [Online]. Available: http://www.net2plan.com

[5] "Telecommunication networks engineering group (girtel)," [Last accessed: October 2018]. [Online]. Available: http://girtel.upct.es/

[6] "Jom (java optimization modeler) - net2plan," [Last accessed: October 2018]. [Online]. Available: http://www.net2plan.com/jom/

[7] "Swagger | the best apis are built with swagger tools," [Last accessed: October 2018]. [Online]. Available: https://swagger.io/

[8] "Onos southbound: Protocol, providers, drivers," [Last accessed: October 2018]. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Southbound%3A+Protocol%2C+Providers%2C+Drivers

[9] "Onos overview | architecture, abstractions & application," [Last accessed: October 2018]. [Online]. Available: https://www.systems.ethz.ch/sites/default/files/slides_onos.pdf

[10] Y. Tseng, "Onos intents and northbound | onos build 2017," [Last accessed: October 2018]. [Online]. Available: https://onosproject.org/wp-content/uploads/2018/01/4-ONOS-Build-2017-Northbound.pdf

[11] "The onos web gui." [Online]. Available: https://wiki.onosproject.org/display/ONOS/The+ONOS+Web+GUI

[12] "The onos cli," [Last accessed: October 2018]. [Online]. Available: https://wiki.onosproject.org/display/ONOS/The+ONOS+CLI