Universidad Politécnica de Cartagena

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Hacia un entorno de código abierto para emular conjuntamente el plano de control y datos de redes ópticas desagregadas

TRABAJO FIN DE GRADO

Grado en ingeniería telemática

Autor: Ignacio Iglesias Castreño

Directores: Pablo Pavón Mariño,

Miquel Garrich Alabarce

Cartagena, 29 de septiembre de 2020

Resumen

Las redes definidas por software (SDN) en infraestructuras ópticas, persiguen arquitecturas desagregadas en las que el equipo óptico se gobierna mediante un controlador SDN con interfaces estándar y abstracciones de modelado comunes.

Los agentes de software son una pieza fundamental en las redes ópticas desagregadas ya que traducen los comandos del controlador SDN en acciones específicas que deben ser realizadas por el equipo de hardware.

Este trabajo consiste en la creación de un marco de desarrollo básico para la emulación de redes ópticas mediante el despliegue automatizado de topologías emuladas por agentes de software.

Para esto, se ha utilizado el framework Netopeer2 – Sysrepo para la construcción de agentes de software, el controlador SDN ONOS, los modelos YANG basados en el transpondedor Cassini y su extensión para ONOS, todos ellos desarrollos recientes dentro de las iniciativas TIP (Telecom Infra Project) y ODTN (Open Disaggregated Transport Network), respectivamente.

Palabras clave

SDN, SDON, redes ópticas, Cassini, NETCONF, OpenConfig, Netopeer2, Sysrepo, agentes de software, ONOS.

Contenido

1. Introducción

1.1 Ventajas y características de SDN

Actualmente los operadores de redes de telecomunicaciones tienen su infraestructura de red compuesta por múltiples dominios, tecnologías y equipos de diferentes fabricantes, lo cual pone a prueba su capacidad operativa. Además, en su infraestructura suele estar acoplado el plano de control (que decide cómo manejar el tráfico) y el plano de datos (que reenvía el tráfico según las decisiones del plano de control). Esto agrava los problemas operacionales, reduce la flexibilidad, genera las indeseables *vendor islands* y limita la innovación de la infraestructura de la red.

El llamado *Software Defined Networking* (SDN) entró en la escena en la última década permitiendo la programabilidad avanzada de la red al desacoplar las acciones del plano de datos de reenvío de las decisiones del plano de control [1]. El SDN mejora la programabilidad de la red con interfaces de programación de aplicaciones (APIs) gracias a un controlador centralizado.

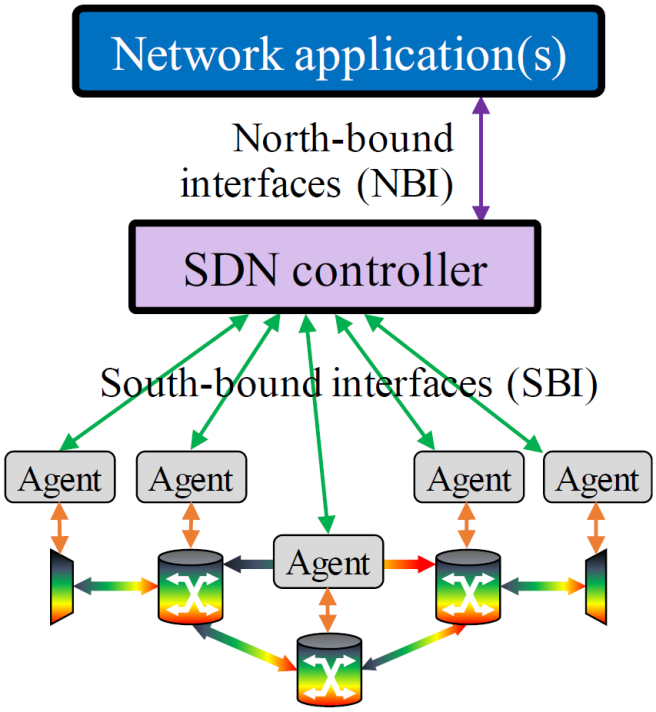

Figura 1: vision general de la estructura de SDN

Como se ilustra en la Fig. 1, el controlador SDN ejerce el control sobre los elementos del plano de datos (flechas verdes) mediante APIs específicas comúnmente denominadas *South Bound Interface* (SBI). Cabe señalar que todos los elementos del

plano de datos incorporan un agente de software estrechamente acoplado encargado de traducir los comandos SBI a acciones específicas del hardware (flechas naranjas). SDN ofrece una visión global de la red con el fin de que las aplicaciones de red que consumen la interfaz *North Bound Interface* (NBI) (flecha morada) definan rutas y políticas en función de las necesidades de tráfico, mejorando así potencialmente el rendimiento y permitiendo oportunidades de optimización [2].

1.2 Estructura de SDON y limitaciones

SDN es una tecnología útil para controlar y gestionar la gran variedad de elementos que componen las redes ópticas con diferentes particularidades específicas relacionados con la transmisión y conmutación fotónica [3]. En este contexto, las redes ópticas definidas por software (SDON) tienen por objeto aprovechar la flexibilidad del control de SDN para apoyar aplicaciones de red con una infraestructura óptica subyacente [4]. En SDON, los modelos de software utilizados para representar (es decir, abstraer) elementos, dispositivos y sistemas ópticos son de gran importancia porque permiten la creación de agentes de software. Existen diferentes modelos [5] para la creación de agentes de software.

En general, algunos de los objetivos de SDON son los de alcanzar las soluciones de “white box” y “disaggregation”.

El primero de los objetivos consiste en que múltiples componentes pueden ser combinados e incorporados en soluciones completas. De esta forma, los vendedores pueden centrarse en la construcción de un componente específico de un dispositivo sin tener que construir un dispositivo completo, acelerando la innovación y reduciendo los costes.

El objetivo de “disaggregation” consiste en la capacidad de combinar por separado el hardware y los sistemas operativos de la red o del nodo que controla los elementos desagregados. De esta manera, se puede adquirir un dispositivo de un fabricante y cargar el sistema operativo de red que se desee.

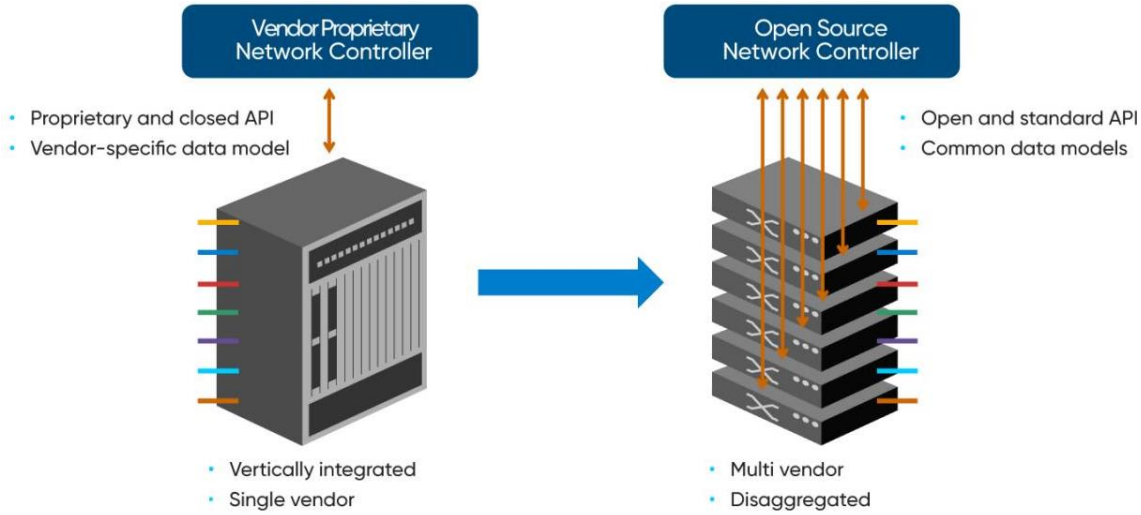

Figura 2: Open Disaggregated Transport Network (ODTN) [6]

La Fig. 2 describe las diferencias entre los sistemas propietarios de los fabricantes y los sistemas abiertos que funcionan con controladores de red de código abierto. En la situación de la izquierda, el operador se encuentra atado a un solo proveedor y su hardware y software propietario. En el caso de la derecha, se puede disponer de dispositivos de diferentes fabricantes de forma desagregada usando un controlador de red y/o del nodo de código abierto.

Las tendencias actuales persiguen el uso de modelos basados en las plantillas descritas utilizando el lenguaje de modelado *Yet Another Next Generation* (YANG) [7] combinados con el uso del protocolo NETCONF [8]. Por ejemplo, OpenConfig [9] y OpenROADM [10] son dos iniciativas de estandarización bien establecidas para SBI en redes ópticas basadas en modelos YANG. Este enfoque ha demostrado ser lo suficientemente flexible y versátil para captar la complejidad de las tecnologías ópticas [11]. Notablemente, la combinación YANG-NETCONF ha despertado el interés de los operadores de telecomunicaciones por perseguir y fomentar los enfoques de *white box* y *disaggregation* [12]. No obstante, como se ha expuesto brevemente arriba, los agentes de software (por ejemplo, basados en los modelos YANG) requieren desarrollos específicos para su interacción con el *hardware*. Este tema, que suele ser infravalorado en la bibliografía a pesar de su importancia para permitir el funcionamiento real con la infraestructura subyacente, es en el que se centra este trabajo.

1.3 Objetivo del trabajo

En este trabajo se presenta una versión simplificada de un entorno de emulación óptica con la creación automática de agentes en un entorno SDON. La sección 2 proporciona una visión general de las tecnologías empleadas. En la sección 3, se describe la arquitectura para la creación automática de agentes para redes ópticas desagregadas. En particular, se presta especial atención a las principales funcionalidades y APIs que se utilizarían para la interacción con equipos ópticos de hardware en condiciones realistas. La sección 4 ilustra el uso de la arquitectura desarrollada centrándose a modo de ejemplo en una topología europea de 18 nodos basados en Cassini. Finalmente, la sección 5 resume el trabajo realizado proponiendo posibles trabajos a desarrollar en el futuro.

2. Tecnologías habilitantes

En esta sección, se revisan los principales elementos que componen el trabajo, comenzando con el controlador SDN ONOS. Posteriormente se trata el protocolo NETCONF y su relación con los modelos YANG. A continuación, se detalla el modelo YANG específico del transpondedor Cassini que se ha empleado en el trabajo y los principales componentes dentro de los agentes de software para redes ópticas y su creación con el framework Netopeer2/Sysrepo. Finalmente se realiza una breve introducción a la tecnología de virtualización basada en Dockers, elemento muy relevante en este trabajo.

2.1 Controlador SDN ONOS

El proyecto ONOS (Open Network Operating System) es un proyecto de código abierto gestionado por la *Linux Foundation* con objetivo de crear un sistema operativo para redes definidas por software para los proveedores de servicios de comunicaciones. Es uno de los controladores SDN de código abierto más usados para el desarrollo de soluciones SDN/NFV de siguiente generación.

Figura 3: logo oficial de ONOS

Se trata de un sistema distribuido, de alta disponibilidad y rendimiento, modular, extensible y dividido en varios subsistemas [13]. Sus características se detallan a continuación:

Alta disponibilidad y robustez: está diseñado para operar como un clúster de nodos redundante que soporta caídas de nodos sin causar problemas al control de la red.

Rendimiento de escala: ONOS está pensado para el control de redes de gran tamaño. Este gran rendimiento se obtiene gracias a la posibilidad de realizar escalado horizontal añadiendo nuevas instancias si es necesario.

Software modular: el software de ONOS se ha diseñado primando la modularidad, manteniendo bien diferenciadas las funciones de software y creando las correctas abstracciones e interfaces. Esto permite una mayor facilidad de mantenimiento del software y de personalización del software por parte de los operadores de red. ONOS incluye actualmente más de 135 extensiones por defecto, entre las cuales se incluyen aplicaciones de control de tráfico, de superposición de redes, modelos YANG precompilados y drivers de dispositivos.

Abstracciones de NBI: ONOS proporciona abstracciones en la NBI que simplifican la creación, despliegue, operación, configuración y mantenimiento de aplicaciones. Ejemplos de esto son la visión global y el framework de *intents*. Las aplicaciones pueden ser fácilmente desarrolladas usando interfaces nativas o interfaces REST/gRPC.

Abstracciones de SBI: ONOS abstrae características de los dispositivos de manera que el sistema central no necesita conocer el protocolo particular que se está usando. Soporta un gran número de protocolos como OpenFlow, NETCONF, TL1, SNMP y RESTCONF.

Interfaz y framework gráfico: la interfaz gráfica de ONOS permite ver la red multicapa y explorar los elementos de red, conectividad, estado de la red, estadísticas, errores y más.

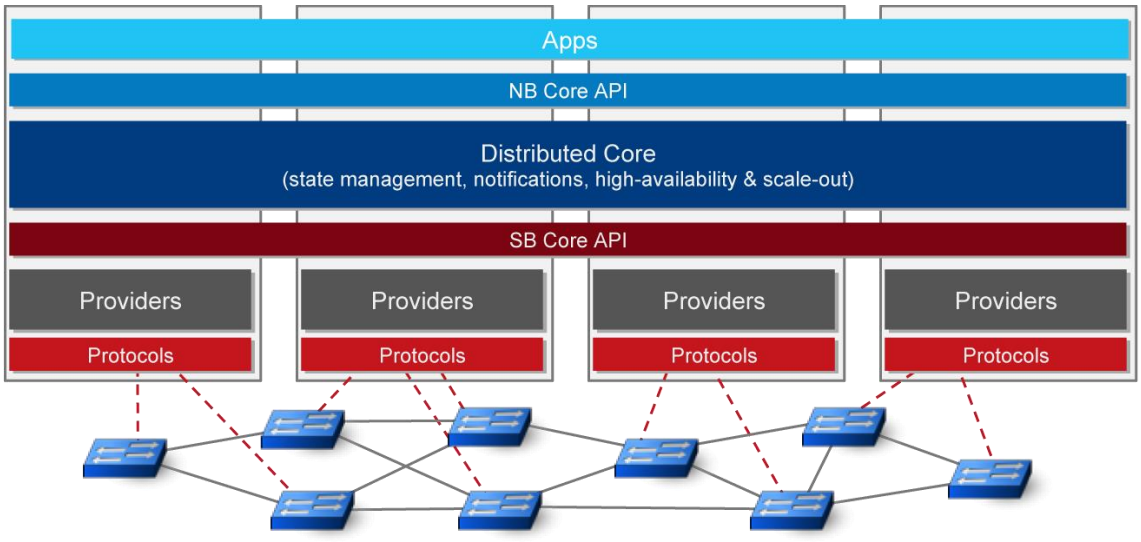

Figura 4: arquitectura modular de ONOS [14]

Como se puede apreciar en la Fig. 4, ONOS consta de una arquitectura formada por diferentes componentes bien diferenciados que interactúan entre sí. En la parte superior se encuentra la interfaz NBI y las aplicaciones de optimización y control de red que la consumen. En la parte inferior se sitúa la interfaz SBI y los protocolos de configuración de red que interactúan con el hardware. Todas estas interfaces son controladas por el componente central. Gracias a esto, ONOS se puede ampliar con capacidades ópticas, lo cual hace que sea una tecnología ideal para este proyecto.

Existen numerosas entidades que forman parte del ecosistema ONOS [14]. Operadoras como AT&T, China Unicom, Comcast, T-Mobile, Google, NTTGroup y Turk Telekom; fabricantes como Edge-Core Networks e Intel y cuenta con miembros como Telefónica, Fujitsu, Ericsson, Broadcom, Nokia, Sprint, Samsung y ZTE.

2.2 Protocolo NETCONF

El protocolo NETCONF, creado por el IETF [8], es un protocolo cliente-servidor de llamada de procedimiento remoto (RPC) que utiliza lenguaje de marcado (XML) para la serialización de datos [15]. Este protocolo permite a los operadores de red instalar, modificar y eliminar configuraciones de elementos de red. NETCONF usa conexiones seguras mediante SSH o TLS con certificados X.509.

Los mensajes XML intercambiados durante las sesiones NETCONF y los datos de configuración se definen por dispositivo siguiendo el formalismo utilizado en el lenguaje de modelado YANG [7]. Como se verá más adelante, este lenguaje permite mapear los parámetros de configuración e información de estado para cada dispositivo en una estructura de datos de árbol.

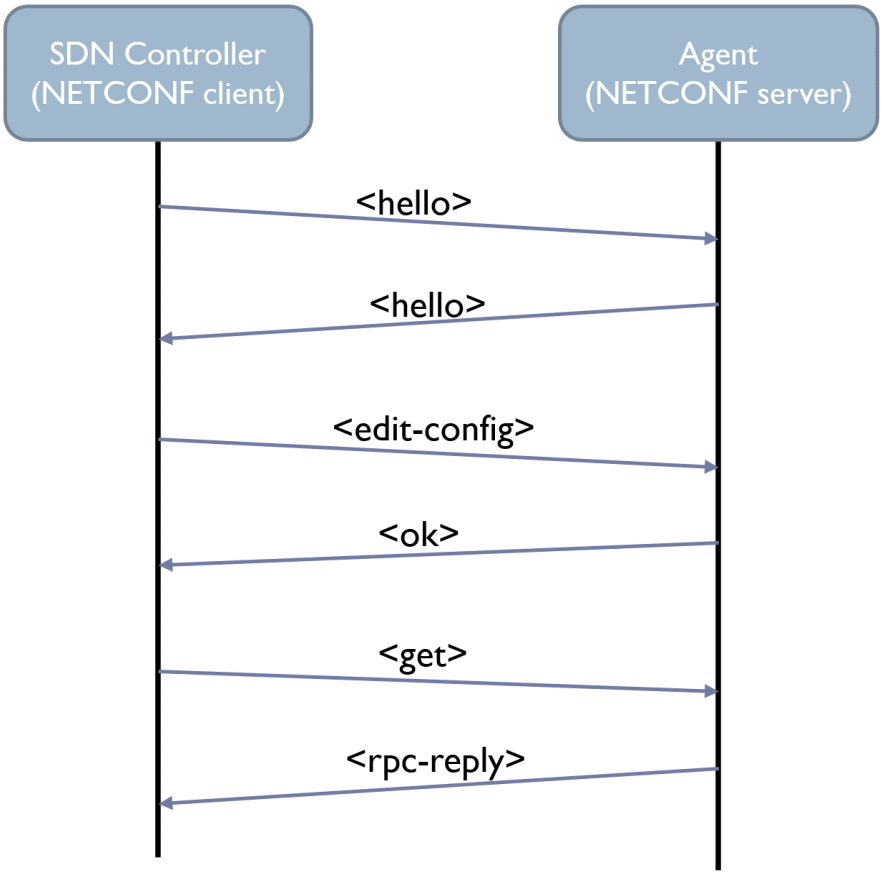

Figura 5: ejemplo de intercambio de mensajes en NETCONF

La Fig. 5 muestra un ejemplo de intercambio de mensajes entre un cliente NETCONF, en este caso el controlador SDN, con un servidor NETCONF que se trata de un agente de software. En primer lugar, se realiza un establecimiento de conexión entre cliente y servidor. A continuación, el cliente NETCONF envía una solicitud de cambio de configuración que resulta exitosa. Finalmente, el cliente NETCONF pregunta por un determinado dato de la configuración del servidor y este se lo envía.

```
NETCONF client request:
<rpc>
  <get-chassis-inventory>
    <detail/>
  </get-chassis-inventory>
</rpc>
]]>]]>

NETCONF server reply:
<rpc-reply xmlns="URN" xmlns:junos="URL">
  <chassis-inventory xmlns="URL">
    <chassis>
```

```
      <name>Chassis</name>
      <serial-number>1122</serial-number>
      <description>M320</description>
      <chassis-module>
        <name>Midplane</name>
        <!--other child tags for the midplane-->
      </chassis-module>
      <!--tags for other chassis modules-->
    </chassis>
  </chassis-inventory>
</rpc-reply>
]]>]]>
```

En las líneas de código anteriores se puede apreciar el contenido de un mensaje NETCONF de petición de un dato por parte del cliente y la correspondiente respuesta del servidor. El cliente pide un dato dentro de su árbol de datos y el servidor le devuelve el valor solicitado, siguiendo las estructuras formales de su correspondiente modelo YANG.

Además de NETCONF, existe otro protocolo muy similar llamado RESTCONF [16] que usa métodos HTTP para implementar las operaciones equivalentes de NETCONF. RESTCONF no intenta reemplazar a NETCONF, sino complementarlo ofreciendo la posibilidad de usar *Representational State Transfer* (REST) [17]. La arquitectura REST permite que los sistemas accedan y manipulen representaciones textuales de recursos mediante el uso de un conjunto uniforme y predefinido de operaciones sin estado.

RESTCONF no cuenta hoy en día con el amplio soporte que tiene NETCONF, por lo que se ha optado por el segundo en este trabajo.

2.3 Lenguaje de modelado YANG

YANG (Yet Another Next Generation) [18] es un lenguaje de modelado de datos que se utiliza para describir la configuración y datos de estado empleados en elementos de red, llamadas a procedimientos remotos y notificaciones de los protocolos de control de red NETCONF y RESTCONF. YANG está mantenido por el IETF y fue publicado en el RFC 6020 [18] en 2010.

Los archivos YANG tienen una estructura de árbol y están formados por los siguientes componentes principales [18]:

- Módulo (module): nivel jerárquico más alto que define el nombre del módulo, la versión de lenguaje YANG, el espacio de nombres, un prefijo para acceder a los módulos internos, declaración de *imports e includes* de otros YANG, nombre de organización, contacto y un historial de revisión.
- Submódulo (submodule): un módulo YANG puede estar formado por varios submódulos de igual estructura.
- Definición de tipo (typedef): define nuevos tipos de datos que van a ser usados dentro del módulo. Solo es necesario si el tipo de datos no está incluido en el lenguaje YANG.
- Contenedor (container): describe el contenido de un nodo de datos. No tiene un valor por sí mismo, pero proporciona una lista de nodos hijos en el árbol de datos.
- Lista (list): conforma una lista con todas las propiedades de un contenedor.
- Hoja (leaf): define una variable escalar de un tipo de datos.
- Lista de hoja (leaf-list): define un array de valores escalares de un particular tipo de datos.
- Tipo (type): tipo de dato asociado a esa hoja. Puede ser un tipo de datos nativo de YANG o uno definido previamente (typedef).
- Notificación (notification): define la estructura de una notificación NETCONF. Consta de un identificador y de información sobre la notificación.

En el Anexo 1 se puede observar el contenido simplificado de un módulo YANG de un transpondedor óptico, del cual se hablará más adelante.

Desde la creación del lenguaje YANG, han aparecido varios estándares creados por organizaciones y fabricantes de hardware con el fin de que todos sigan las mismas reglas para modelar equipos. Los estándares YANG más conocidos relacionados con redes ópticas son OpenConfig y OpenROADM.

Figura 6: logos de OpenConfig y OpenROADM

- OpenConfig es un conocido estándar en la industria muy adoptado por fabricantes de hardware.
- OpenROADM es un estándar creado para dar soporte únicamente a ROADMs.

Existen múltiples implementaciones de YANG, siendo las más relevantes Pyang [19] y Libyang [20]. En este trabajo se ha optado por la segunda implementación.

2.4 Modelo YANG de Cassini basado en OpenConfig

En este trabajo se ha usado el modelo YANG del transpondedor Cassini [21]. Cassini es un transpondedor de paquetes de tipo *white box* desarrollado dentro del *Telecom Infra Project* (TIP) [22] por Edgecore Networks.

Figura 7: aspecto físico del transpondedor Cassini AS7716-24SC

Cassini (ver Fig. 7) dispone de 16 puertos conmutados 100-Gigabit Ethernet y 8 líneas ópticas 200 Gbps para la interconexión entre centros de datos y *backhaul*, un factor de forma rack 1,5" y un rendimiento agregado de hasta 3.2Tbps al incorporar el chipset *Broadcom TomahawkTM* [21].

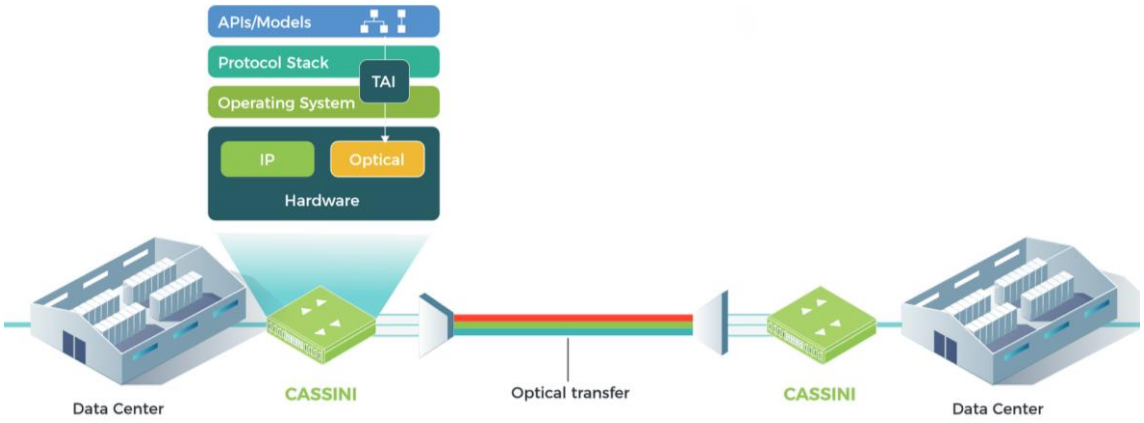

Figura 8: emplazamiento del Cassini en redes ópticas

Este dispositivo es el primer transpondedor de paquetes *white box* creado para permitir a los operadores de red migrar o extender redes metro y DWDM existentes hacia prestaciones 200Gbps y servicios *Layer 3* dentro de *datacenters*, todo en una plataforma abierta.

Al estar modelado en estándares YANG de OpenConfig, es compatible con controladores genéricos de interfaces SBI basados en NETCONF. En el caso del controlador SDN ONOS, se distribuye con drivers para transpondedores Cassini incorporados [23].

2.5 Netopeer2 y Sysrepo

Estas dos tecnologías conforman un framework para la creación de agentes de software. Ambas son *open source* e implementan estándares abiertos como NETCONF y YANG.

Netopeer2 es una implementación de servidor del protocolo NETCONF. Se encarga de la comunicación con el cliente NETCONF, -normalmente el controlador SDN- y de enviar estas órdenes a Sysrepo.

Sysrepo tiene varias funcionalidades: es la base de datos de la configuración de un agente, asegura la consistencia de los datos y las restricciones según los modelos YANG instalados y puede enviar llamadas de retorno de acuerdo con eventos específicos, ya sea a un hardware emulado o una aplicación.

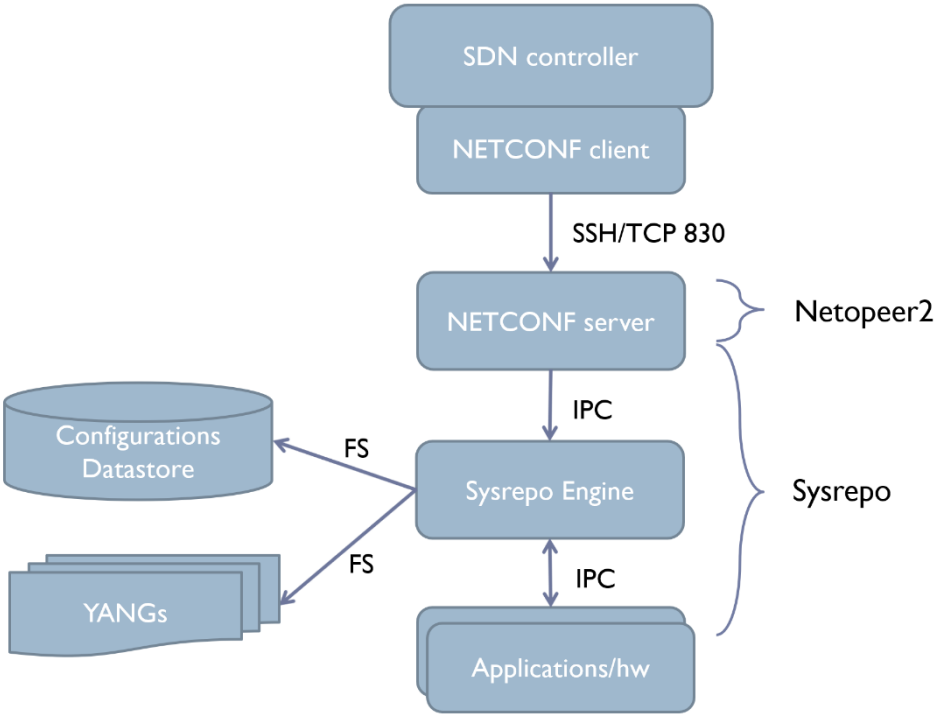

Figura 9: estructura del framework Netopeer2/Sysrepo

La Fig. 9 esquematiza la separación entre los diferentes módulos y el flujo de datos. En primer lugar, el controlador SDN (cliente NETCONF) establece la conexión con un agente de software. Tras esto, puede realizar peticiones de solicitud de datos o de cambios de configuración. Estas peticiones son recibidas por el servidor NETCONF del agente, que se trata del programa Netopeer2. En caso de una solicitud de datos, este componente transmite la petición a Sysrepo, que es el encargado del mantenimiento de las configuraciones y responderá con la configuración requerida. En caso de una petición de modificación de configuración, modificará una configuración de la base de datos. En este último tipo de peticiones, Sysrepo comprueba las restricciones de los archivos YANG instalados de manera que cumpla estrictamente con estos modelos.

Si la operación de modificación se puede realizar, Sysrepo actualiza la configuración de la base de datos y notificando estos cambios a otros componentes que estuvieran subscritos, ya sean aplicaciones o de equipos físicos.

2.6 Docker

Esta tecnología juega un papel importante en este trabajo por sus funciones de virtualización y automatización. Docker permite virtualizar los agentes de software en contenedores completamente aislados del exterior y de otros contenedores, como se encontrarían en un escenario real. Además, permiten automatizar su creación y despliegue siguiendo una serie de configuraciones definidas.

Estos contenedores, ligeros y portables, posibilitan que las aplicaciones software puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo de la máquina, facilitando así los despliegues.

En el ecosistema Docker, existen dos componentes principales:

- Imágenes Docker: paquete de software ligero, independiente y ejecutable que contiene todo lo necesario para ejecutar una aplicación: código, runtime, herramientas, librerías y configuraciones.
- Contenedores Docker: una imagen Docker pasa a ser un contenedor Docker cuando es ejecutada por Docker Engine. Las aplicaciones que forman el contenedor Docker se ejecutan siempre de la misma forma y son invariantes, independientemente de la infraestructura y del sistema operativo. Además, estos contenedores se encuentran aislados de su entorno.

El concepto de imágenes y contenedores Docker puede parecer similar al de las máquinas virtuales, no obstante, existen diferencias de funcionamiento significativas.

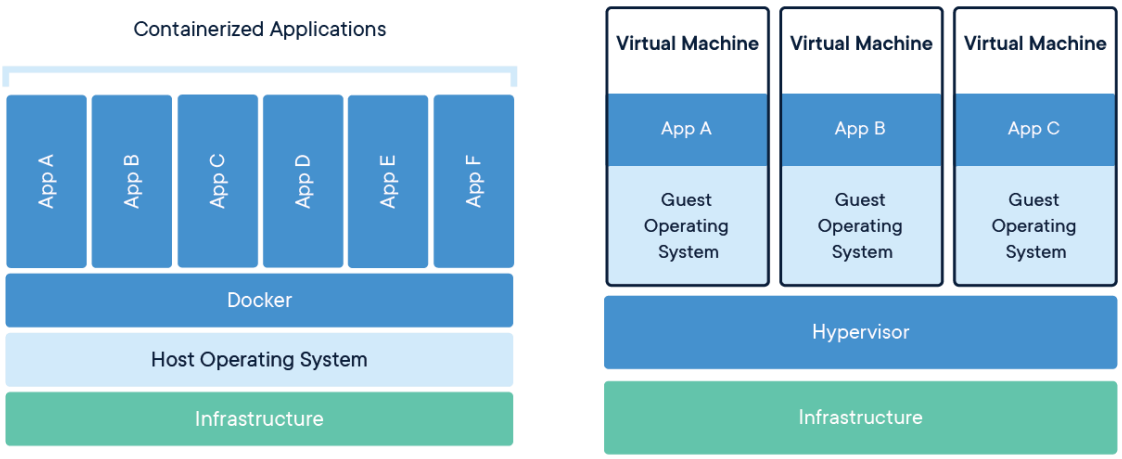

Figura 10: diferencias entre contenedores Docker y máquinas virtuales [24]

Como se puede apreciar en la Fig. 10 en el esquema de la derecha, cada máquina virtual debe incluir un sistema operativo completo, con un kernel independiente y todas los binarios y librerías necesarias. Por el contrario, en el esquema de la izquierda, Docker comparte el mismo kernel del sistema operativo original. Gracias a esto último, los contenedores Docker son mucho más ligeros, rápidos, eficientes y ocupan menos espacio que las máquinas virtuales.

Dentro de las herramientas disponibles de Docker, es importante destacar Docker Compose, que se emplea para automatizar el despliegue de contenedores. Las configuraciones de lanzamiento se definen en un archivo YAML con el nombre “docker-compose.yml” y se ejecutan mediante un único comando.

3. Arquitectura para la creación automática de agentes en SDON

En este trabajo se ha desarrollado una arquitectura para la creación automática de agentes de software con el objetivo de emular el despliegue y funcionamiento de redes ópticas controladas por SDN. Esta arquitectura está compuesta por tres elementos principales: el controlador SDN ONOS, un conjunto de agentes de software y la API gateway.

3.1 Controlador SDN ONOS

El controlador SDN ONOS tiene una visión completa de la red óptica, conformando una pieza fundamental de la arquitectura.

ONOS permite cargar una topología mediante su interfaz NBI y ver información relacionada con los dispositivos, estadísticas, sus conexiones e incluso realizar algunas funciones de control de la red directamente utilizando su interfaz gráfica. Contando además con una consola de comandos donde se pueden realizar funciones adicionales.

Al ser ONOS un sistema modular, sus características y funciones se pueden extender mediante módulos, desarrollados por la comunidad o por los propios fabricantes.

Como se ha mencionado anteriormente, ONOS dispone de módulos con capacidad óptica nativa, que deben activarse dentro de la interfaz gráfica o mediante la consola de comandos.

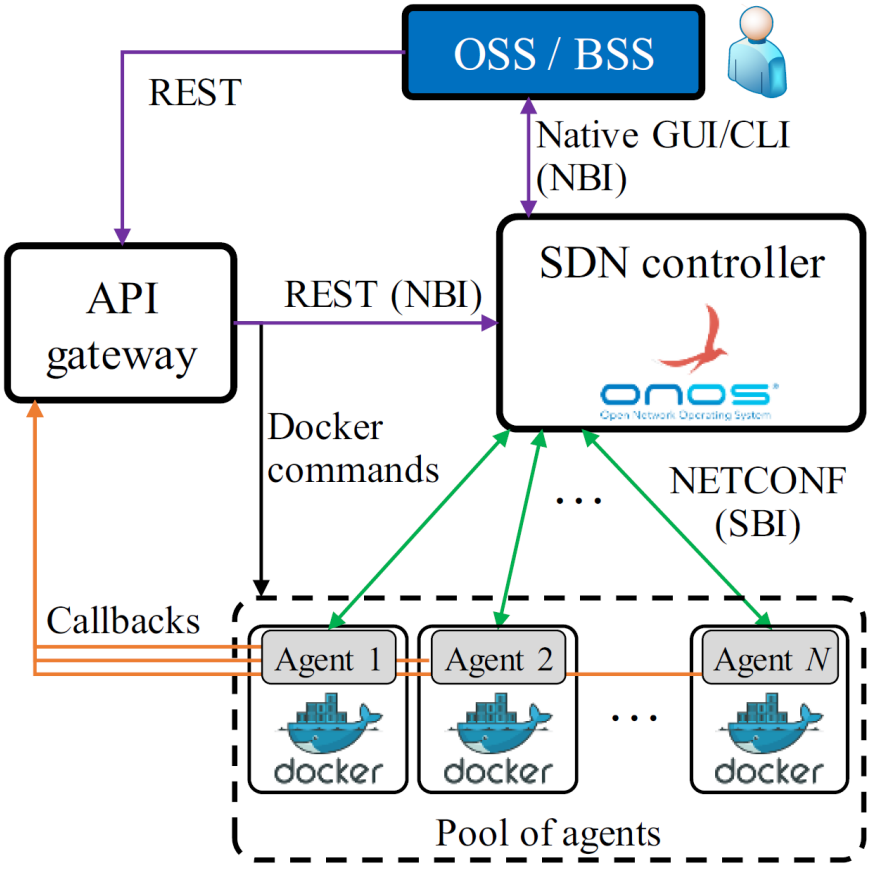

Figura 11: situación de ONOS en la arquitectura, resaltado en negro.

Como se puede ver en el esquema de la Fig. 11, ONOS recibe una topología por parte de la API gateway (flecha morada) mediante su interfaz NBI y además se comunica con los agentes de software (flechas verdes) mediante la interfaz SBI usando sesiones NETCONF. Además de esto, también es posible interactuar con ONOS por medio de su interfaz gráfica y consola de comandos incorporadas (flecha morada superior).

El flujo de datos se describe a continuación: en primer lugar, ONOS recibe una topología detallada por parte de la API gateway mediante la interfaz NBI. Tras esto, ONOS establece conexión con los agentes de software que están definidos en la topología mediante la interfaz SBI. Una vez que se ha conseguido establecer comunicación con los agentes, ONOS ya tiene el control sobre estos y abre la posibilidad de realizar simulaciones de redes ópticas.

3.2 Conjunto de agentes de software

Cada agente de software de la topología se emula con un contenedor Docker. En este caso, como estamos usando topologías compuestas por transpondedores Cassini, esto se aplica a cada uno de ellos.

Gracias al uso de contenedores Docker, se consigue un entorno realista en el que todos los agentes son independientes y están aislados unos de otros. Estos agentes de software son solo accesibles por ONOS a través de NETCONF. Además de esto, los agentes pueden enviar notificaciones al exterior cuando su configuración ha cambiado, permitiendo posibles optimizaciones mediante programas externos.

Estos contenedores tienen como sistema operativo Ubuntu 18.04 y los siguientes componentes incorporados:

- Netopeer2: servidor NETCONF del agente.
- Sysrepo: base de datos de configuraciones que comprueba restricciones y administra subscripciones a aplicaciones/hardware externo.
- Modelos YANG OpenConfig: el agente incorpora los modelos YANG del transpondedor Cassini, instalados en Sysrepo con el fin de administrar y restringir configuraciones.
- Scripts de carga e inicialización de los modelos YANG en Sysrepo.
- Archivos de configuración por defecto de Cassini, en los que se describen los parámetros iniciales del transpondedor
- Manipulador de notificaciones: código Python que actúa cuando cualquier configuración del agente ha cambiado y administra las subscripciones expedidas por Sysrepo comunicándolas a la API gateway.
- Libyang y libnetconf2: librerías adicionales requeridas por Netopeer2 y Sysrepo.
- Otras utilidades como compiladores C++ y un servidor SSH ya que NETCONF usa SSH como aplicación de transporte.

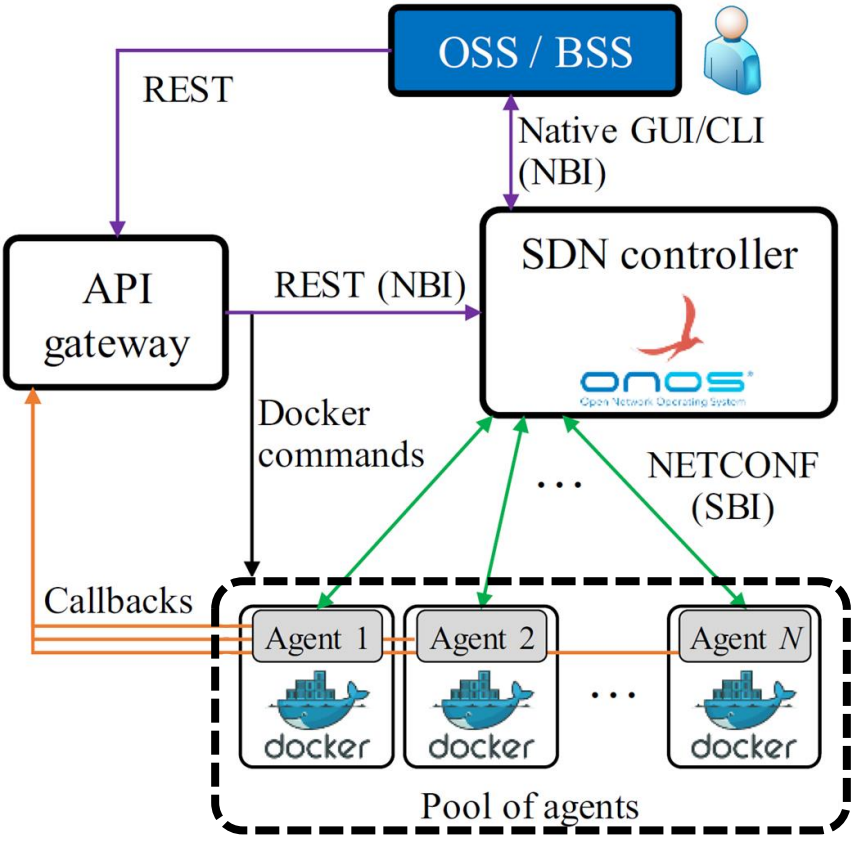

Figura 12: situación de los agentes de software en la arquitectura, parte inferior.

Como se puede ver en la Fig.12, los agentes de software son creados por la API gateway mediante comandos Docker (flecha negra) comunicándose con ONOS mediante sesiones NETCONF por la interfaz SBI (flechas verdes). Además, los agentes pueden notificar cambios de configuraciones (flechas naranjas) a la API gateway, centralizando las notificaciones y permitiendo realizar simulaciones a partir de ellas.

3.3 API Gateway

El último componente de la arquitectura que se ha desarrollado se ha denominado API gateway. Es un programa escrito en JavaScript basado en la arquitectura REST que centraliza y automatiza las operaciones de creación y notificaciones de agentes de software.

La API gateway recibe una topología estándar y a partir de esta crea el número requerido de contenedores Docker que forman los agentes de software. Tras esto, la topología se traduce en una red óptica emulada que puede ser controlada por ONOS enviándose al mismo mediante su REST API situada en la interfaz NBI.

Para este cometido, la API gateway extrae de la topología inicial el número y posición geográfica de los nodos y los enlaces entre ellos para una un doble propósito: generar los comandos Docker apropiados y comunicar la topología de la red a ONOS. La API gateway admite representaciones de redes basadas en XML de Net2Plan [25], así como el despliegue de un cierto número de nodos en ubicaciones aleatorias.

Además de lo descrito anteriormente, la API gateway tiene la función de recibir notificaciones de los agentes cuando sus configuraciones han sido modificadas, centralizando estas notificaciones en un único componente.

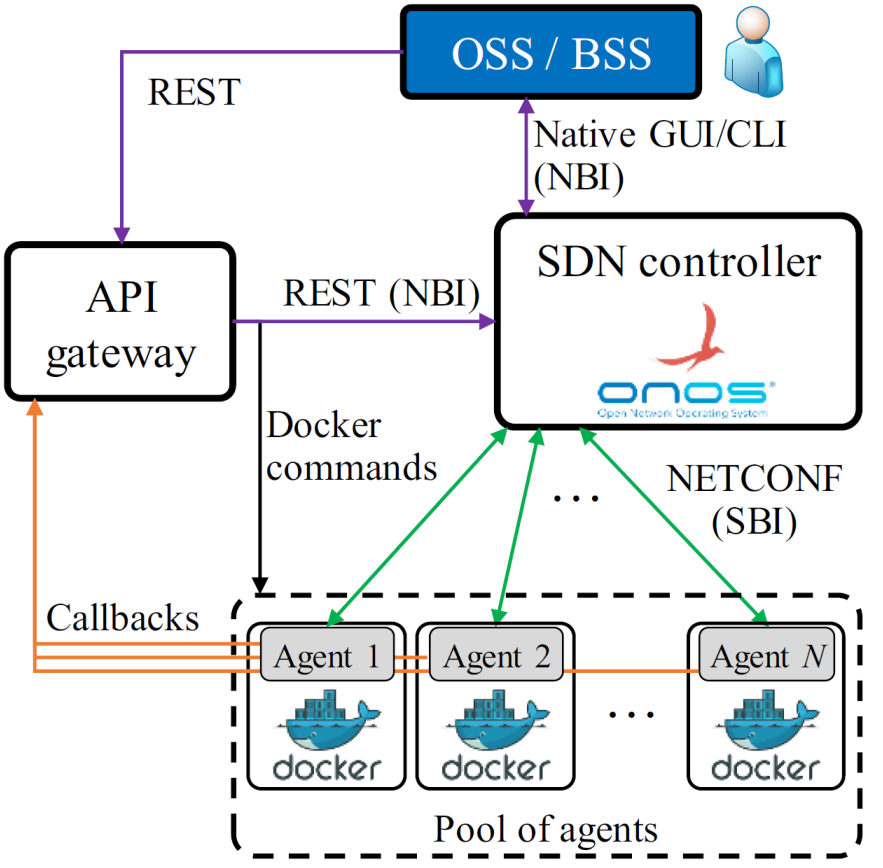

Figura 13: situación de la API gateway en la arquitectura, resaltado en negro a la izquierda.

La Fig. 13 muestra como la API gateway recibe una topología mediante su interfaz REST (flecha morada) y se encarga de traducirla y transformarla en una serie de comandos Docker que crean los agentes de software requeridos (flecha negra) y además envía a ONOS mediante su interfaz NBI una topología basada en la anterior (flecha morada). Esta nueva topología contiene información relacionada con los nodos, enlaces y sus datos de conexión, necesarios para que ONOS use los agentes de software como los nodos de la topología.

Se debe resaltar que una gran ventaja que ofrece la API gateway a la arquitectura es permitir la conexión con nuevas aplicaciones de manera que se amplíen y se creen nuevas funcionalidades de emulación.

4. Caso de uso ilustrativo con transpondedores Cassini

Esta sección contiene un caso particular que ilustra la arquitectura propuesta. En concreto se muestra el proceso de despliegue de una topología óptica en las principales capitales europeas donde cada nodo se asume que es un transpondedor y los enlaces son enlaces ópticos directos entre ellos, sin más elementos ópticos.

4.1 Instalación y configuración de ONOS

En primer lugar, se debe proceder a la instalación y configuración de ONOS. En este ejemplo se han realizado las instalaciones en una máquina local y por tanto las direcciones apuntan a *localhost*. En caso de que se realizaran en una máquina remota, simplemente habría que sustituir *localhost* por la IP remota. Además, los puertos de la máquina remota y posibles firewalls entre medias deben estar accesibles.

Como se ha mencionado anteriormente, ONOS puede instalarse manualmente o virtualizado en un contenedor Docker. La instalación manual [26] se desaconseja ya que es algo tediosa y propensa a errores.

Si por el contrario se elige instalar mediante Docker es necesario, en primer lugar, tener Docker Engine instalado [27] en la máquina. Es importante destacar que, para hacer funcionar Docker en Windows, la placa base debe tener las opciones de virtualización activas y la versión de Windows debe ser Profesional o superior, ya que las versiones estándar no permiten la virtualización.

La instalación de ONOS mediante Docker se realiza mediante el uso de un único comando:

```
docker run -t -d -p 8181:8181 -p 8101:8101 -p 5005:5005 -p
830:830 --name onos onosproject/onos:2.3.0
```

La explicación de los contenidos del comando es la siguiente:

- `docker run:` inicia un contenedor Docker a partir de una imagen.
- `-t o --tty:` opción que asigna una consola al proceso principal.
- `-d o --detach:` opción para ejecutar el contenedor en *background.*

- `-p o --publish:` mapea los puertos del host con los del contenedor. En este caso estamos mapeando los puertos del contenedor 8181, 8101, 5005 y 830 a sus homólogos del host. El puerto 8181 se trata de interfaz web de ONOS, el 8101 es la consola de comandos de ONOS, el 5005 permite realizar *debugging* y el puerto 830 es el de cliente NETCONF.
- `--name onos:` asigna un nombre al contenedor, para poder referirnos a el más adelante como "onos" y no tener que usar identificadores aleatorios.
- `onosproject/onos:2.3.0:` imagen oficial de ONOS de la cual arrancará el contenedor. El nombre de la imagen está formado por el nombre del repositorio y la etiqueta de la versión. Todas las versiones se encuentran disponibles en la página oficial de ONOS del repositorio DockerHub [28].

Tras ejecutar el comando, la imagen se descargará y creará el contenedor de ONOS en segundo plano. Esta imagen solo se descargará la primera vez, ya que, al ser invariantes, Docker guarda las imágenes para futuros usos.

Una vez la imagen esté descargada y el contenedor funcionando, debemos acceder a la interfaz web para activar las funcionalidades ópticas de ONOS. Para ello, se debe navegar a la dirección http://localhost:8181/onos/ui/login.html en el navegador. En el formulario de login se debe introducir "karaf" como usuario y como contraseña por defecto. Acto seguido, se deben activar las dependencias "odtn-service", "roadm" y "optical-rest" en la pestaña "apps".

Esto mismo también se puede realizar mediante la consola de comandos de ONOS. Para entrar a la consola, debemos realizar una conexión SSH al puerto 8101 por medio de un terminal:

```
ssh onos@localhost -p 8101
```

La contraseña para el usuario "onos" es "rocks" por defecto.

Dentro de la consola se deben introducir los siguientes comandos para activar las funcionalidades ópticas:

```
app activate odtn-service
app activate roadm
app activate optical-rest
```

Para salir se puede usar `control + d`.

Para terminar con la configuración de ONOS, debemos crear una red virtual llamada “sdn_optical_network” que será usada por los contenedores que forman los agentes de software. Para ello, se debe ejecutar el siguiente comando:

```
docker network create sdn_optical_network
```

4.2 Instalación y configuración de la API gateway

A continuación, se procede a explicar la instalación y configuración de la API gateway. Antes de comenzar con la instalación de este componente, es necesario tener Docker Engine instalado, igual que en la instalación de ONOS, Docker Compose, Node.js versión 8 o superior y Git. La documentación para la instalación de Docker Compose está disponible en [29], la de Node.js en [30] y la de Git en [31], aunque este último suele estar instalado por defecto en la mayoría de sistemas operativos.

Para desarrollar estos componentes se ha usado el editor de código Visual Studio Code [32], herramienta recomendada incluso para realizar la instalación y configuración de la API gateway, ya que es muy útil para administrar contenedores Docker y consolas de comandos. Como terminal se recomienda PowerShell para Windows, Terminal para Macintosh y Bash para Linux.

Primero, se debe descargar el código del proyecto desde el repositorio de GitHub [33]. Este código se puede descargar directamente desde el botón de descarga o clonando mediante Git, lo cual es más conveniente:

```
git clone https://github.com/iicc1/SDON-emulator.git
```

A continuación, dentro de la carpeta del proyecto en una carpeta llamada “api”, se deben instalar las dependencias de Node.js mediante el siguiente comando:

```
npm install
```

Tras esto, las dependencias de Node.js comenzarán a descargarse en una nueva carpeta llamada “node_modules”.

Una vez finalizada la instalación de las dependencias, se deben configurar las variables de entorno del proyecto. Dentro de la carpeta del proyecto existe un archivo llamado “env.example” que contiene todas las variables en un archivo de ejemplo. Este archivo se debe copiar o renombrar de manera que exista un archivo “.env”. La API gateway usará las variables de entorno incluidas en este archivo.

A continuación, se detallan las variables de entorno disponibles y sus principales funciones:

- API_SERVER_PORT: en este puerto escucha peticiones la API gateway. El puerto por defecto es el 8000.
- SERVER_IP: IP privada de la máquina donde está instalada la API gateway. No se puede usar localhost aunque sea la misma máquina ya que la comunicación agentes de software → API gateway debe pasar por un router. La función de esta IP es la de centralizar las notificaciones de los agentes en la API gateway.
- ONOS_API_ENDPOINT: URL donde está localizada la API REST de ONOS. Por defecto es http://localhost:8181/onos/v1/. En el caso de que ONOS esté ubicado en otra máquina, se debe sustituir localhost por la IP correspondiente.
- ONOS_API_USER: nombre de usuario de la API de ONOS. Por defecto es “karaf”.
- ONOS_API_PASSWORD: contraseña de la API de ONOS. Por defecto es “karaf”.

4.3 Despliegue de una topología óptica

Una vez instalada y configurada la API gateway, podemos inicializar el servidor. Para ello es necesario ejecutar el siguiente comando en la carpeta del proyecto:

```
node api/server.js
```

El servidor se iniciará e inmediatamente estará disponible su API para poder desplegar topologías ópticas en ONOS. Las rutas disponibles son las siguientes:

Ruta de prueba que únicamente devuelve un true si la API está funcionando.

```
GET /test
```

Esta ruta inicia el proceso de desplegar una topología seleccionada, donde “tipo topología” es el formato de topología soportado. En este caso, solo las topologías tipo Net2Plan están soportadas. Y “nombre topología” es el nombre de una de las topologías instaladas pertenecientes a un “tipo topología”.

```
GET /deploy/{tipo topología}/{nombre topología}
```

Elimina los contenedores Docker de agentes de software desplegados anteriormente.

```
GET /delete/containers
```

Elimina la topología actual de ONOS.

```
GET /delete/topology
```

Elimina los contenedores Docker de los agentes de software y la topología actual de ONOS.

```
GET /delete/all
```

Ruta usada por los agentes de software para enviar *callbacks* de cambios de configuración a la API gateway.

```
POST /callback
```

Como se ha indicado, se va a mostrar el despliegue de una topología de las principales capitales europeas, compuesta por 18 nodos y 66 enlaces. Para ello se debe realizar una petición GET a la siguiente URL:

```
GET /deploy/net2plan/eon_N18_E66_withTraffic
```

La petición se puede realizar haciendo uso de un navegador o de una herramienta como Postman [34].

Como se ha indicado, al encontrarse la API gateway en una máquina local y usar el puerto por defecto, se usa *localhost:8000* como dirección. Para desplegar una topología, añadimos a la URL */deploy;* al ser una topología de tipo net2plan, sigue con */net2plan* y finalmente */eon_N18_E66_withTraffic* es el nombre de la topología instalada de Net2plan que se va a usar.

Inmediatamente después de recibir la petición, la API gateway comenzará con el proceso de creación de agentes de software. La representación XML de esta topología se obtiene y analiza, determinando el número de agentes necesarios (uno por cada nodo Cassini) y los enlaces entre ellos. En este caso, como la topología tiene 18 nodos, creará 18 agentes de software mediante contenedores Docker.

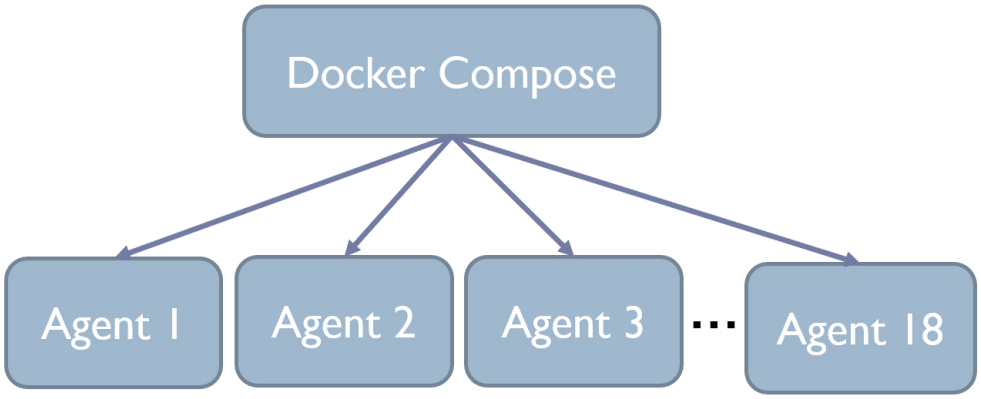

Figura 14: función de Docker Compose simplificada

El proceso de creación de agentes comienza cuando la API gateway recibe la petición en `controllers/routes.js`. Esta petición es parametrizada y enviada a procesar por `models/deploy.js`. Dentro de esta función, la petición de despliegue de la topología es procesada por medio de las funciones incluidas en `helpers/docker.js`, `helpers/Net2PlanParser.js` y `helpers/onos.js`. Concretamente, dentro de `helpers/docker.js` se realiza la siguiente llamada a Docker Compose:

```
docker-compose up -d --build --scale agent={número de agentes}
```

Este comando ordena a Docker Compose a lanzar un número determinado de instancias (en este caso 18, pues hay 18 nodos). Docker Compose inicia estas instancias a partir de su archivo de configuración, la cual se encuentra disponible en el Anexo 2.

En el interior de esta configuración se especifica el rango de puertos disponibles para los agentes y sus mapeos en el host; una variable de entorno que indica la URL de los callbacks; la red Docker a la cual deben pertenecer los nuevos contenedores y la ruta donde está contenido el archivo Dockerfile de los agentes.

El archivo Dockerfile, tal como se muestra en el Anexo 3, contiene una lista de instrucciones que sigue Docker para la creación del agente. Se puede destacar el uso de Ubuntu 18.04 como imagen base, instalación de numerosas dependencias relacionadas con la compilación de código C y C++, openssh y Python, clonado de repositorios como libyang, libnetconf2, sysrepo, netopeer2 y su posterior compilación y finalmente la instalación de los modelos YANG de Cassini en sysrepo y sus configuraciones iniciales.

La primera vez que se inicia el proceso de creación de agentes tarda un tiempo considerable, ya que debe descargar y compilar todos los componentes del agente. No obstante, sucesivos despliegues serán inmediatos ya que, como se ha indicado, Docker guarda las imágenes.

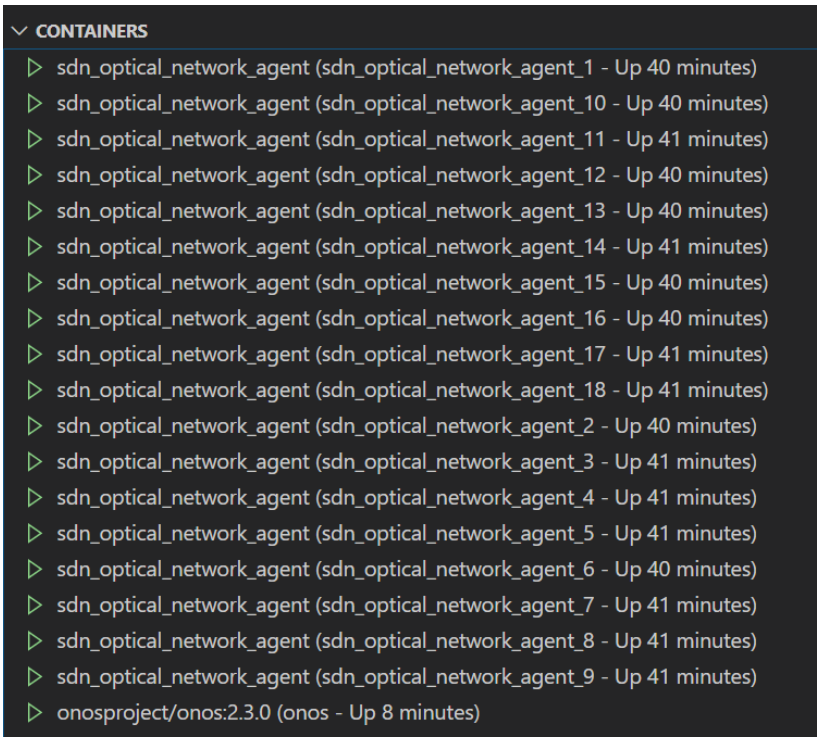

Figura 15: captura de pantalla de la visualización de los 19 contenedores activos en Visual Studio Code [32]

Tras haber sido generadas las imágenes Docker de los agentes e inicializados los contenedores, la API gateway esperará unos segundos antes de comunicar la topología a ONOS. Esta espera es necesaria ya que los agentes tienen que arrancar servicios como NETCONF, SSH y Sysrepo y realizar las configuraciones iniciales. Este desarrollo se podría ampliar de manera que los agentes se encarguen de comunicar cuando estén preparados.

Trascurrido ese tiempo de espera inicial, la API gateway adapta la topología proporcionada en forma de una topología interpretable por ONOS y se la envía a través de su REST API por la interfaz NBI.

Esta topología está definida en un archivo JSON por medio de siete objetos: “devices”, “links”, “hosts”, “apps”, “ports”, “regions” y “layouts”. En este trabajo solo se emplean los

objetos “devices” y “links”. En las siguientes líneas se puede observar un ejemplo de un valor dentro de cada componente usado y en el Anexo 4 se puede encontrar la topología de ejemplo completa.

```
"netconf:192.168.31.239:8007": {
      "basic": {
            "name": "Madrid",
            "driver": "cassini-openconfig",
            "locType": "geo",
            "latitude": "40.4188889",
            "longitude": "-3.6919444"
      },
      "netconf": {
            "ip": "192.168.31.239",
            "port": "8007",
            "username": "root",
            "password": "root",
            "idle-timeout": 0
      }
    }
```

Este JSON compone un nodo de la topología de ONOS. Define la dirección NETCONF asociada a su agente de software y los datos de acceso; el nombre del nodo; el driver que debe usar ONOS y la ubicación geográfica.

```
"netconf:192.168.31.239:8007/219-
      netconf:192.168.31.239:8003/216": {
      "basic": {
            "type": "OPTICAL",
            "metric": "1",
            "durable": true,
            "bidirectional": true
      }
    }
```

A diferencia del anterior JSON, este compone un enlace de la topología. Como se puede observar, contiene información relativa al puerto origen de un nodo y el puerto destino de otro nodo; tipo de enlace y si es bidireccional.

Tras recibir la topología óptica, ONOS inicia una sesión de NETCONF por cada agente y realiza un descubrimiento de la configuración. En este punto, ONOS tiene control total sobre los agentes a través del protocolo NETCONF.

Finalmente, en la interfaz web de ONOS se puede ver la topología europea correctamente desplegada, con 18 nodos y sus enlaces ópticos.

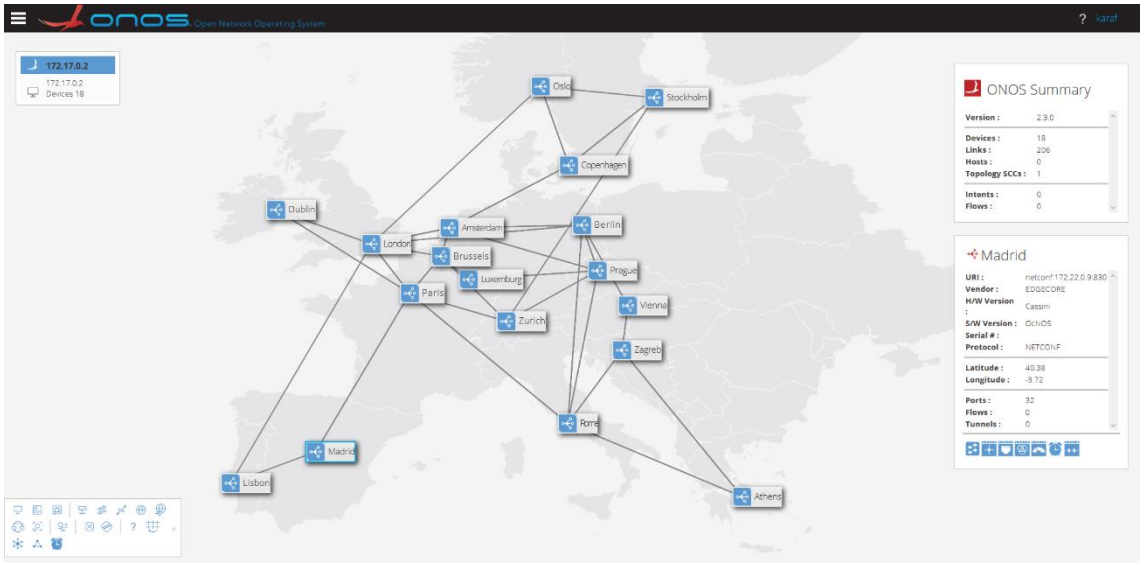

Figura 16: topología óptica desplegada en ONOS

Como se puede ver en la Fig. 16, cada nodo posee la información de su agente de software asociado. En el nodo seleccionado se puede apreciar la URI NETCONF, la posición geográfica, el nombre y datos del agente de software de Cassini asociado, como el fabricante, driver, puertos, etc.

4.4 Cambios de configuración en los agentes de software

Una vez la topología óptica está desplegada y administrada por ONOS, ya se puede usar para realizar emulaciones.

Mediante las utilidades CLI incorporadas en ONOS, es posible ver y ajustar parámetros como la potencia de lanzamiento óptica, la longitud de onda y el formato de modulación de los puertos de los dispositivos ópticos basados en Cassini. A continuación, se muestran algunos ejemplos:

Con este comando se obtienen los valores de potencia en el puerto 202 del agente de software que posee la dirección 192.168.31.239:8022 como servidor NETCONF. En concreto, se obtienen los valores objetivos y actuales de entrada y salida del puerto óptico, en unidades de dBm (decibelio-millivatio).

```
onos@root > power-config get netconf:192.168.31.239:8022/22
The target-output-power value in port 202 on device
netconf:192.168.31.239:8022 is -5.100000.
The current-output-power value in port 202 on device
netconf:192.168.31.239:8022 is 0.120000.
The current-input-power value in port 202 on device
netconf:192.168.31.239:8022 is 0.120000.
```

Con este comando se ha modificado la potencia del puerto anterior a un valor de 2dBm.

```
onos@root > power-config edit-config
       netconf:192.168.31.239:8022/22 2
Set 2.00000 power on port
```

Tras haber modificado la potencia del puerto, la volvemos a comprobar para ver que efectivamente ha sido cambiada.

```
onos@root > power-config get netconf:192.168.31.239:8022/22
The target-output-power value in port 202 on device
netconf:192.168.31.239:8022 is 2.000000.
The current-output-power value in port 202 on device
netconf:192.168.31.239:8022 is 0.120000.
The current-input-power value in port 202 on device
netconf:192.168.31.239:8022 is 0.120000.
```

Este comando permite modificar la longitud de onda de los puertos ópticos del transpondedor.

```
onos@root > wavelength-config edit-config
      netconf:192.168.31.239:8022/22 1
Setting wavelength 386150.0
```

Este comando permite modificar la modulación de los puertos ópticos del transpondedor.

```
onos@root > modulation-config edit-config
      netconf:192.168.31.239:8022/22 16-QAM
Set modulation for 16-QAM for port 202 on device
netconf:192.168.31.239:8022.
```

Los anteriores cambios de configuración se han realizado directamente con las funcionalidades integradas en ONOS, que han sido internamente traducidos a peticiones de tipo NETCONF. Así pues, para realizar el resto de cambios y consultas de configuraciones posibles, se deben crear los XML correspondientes y usar un cliente NETCONF, como el que dispone ONOS integrado.

Al realizar un cambio de configuración en un agente de software, este inmediatamente dispara un aviso indicando los cambios realizados y la nueva configuración. Esta notificación se origina en Sysrepo, notificando a sus subscripciones. Como se ha

mencionado anteriormente, se ha desarrollado un programa que está suscrito a estas notificaciones de Sysrepo, las transforma y las retransmite a la API gateway mediante una petición POST por una variable de entorno definida en la creación de la imagen Docker de los agentes.

```
Callback received
Notification: verify
Changes: MODIFIED: old value/openconfig-
platform:components/component [name=oe1/1']/openconfig-
terminal-device:optical-channel/config/target-output-power = 5
new value/openconfig-
platform:components/component[name='oe1/1']/openconfig-
terminal-device:optical-channel/config/target-output-power = 2
```

```
Callback received
Notification: apply
Changes: MODIFIED: old value/openconfig-
platform:components/component [name='oe1/1']/openconfig-terminal-
device:optical-channel/config/target-output-power = 5
new value/openconfig-
platform:components/component[name='oe1/1']/openconfig-terminal-
device:optical-channel/config/target-output-power = 2
```

En el texto superior se puede observar una notificación de cambio de configuración emitida por un agente y mostrada por la API gateway. Esta notificación está formada por dos mensajes, el primero de verificación y el segundo de modificación. Esto es así ya que Sysrepo primero comprueba que los cambios son realizables y a continuación los modifica. El contenido de estos mensajes es generado automáticamente por el programa Python subscrito a los callbacks de Sysrepo. Como se puede apreciar en la notificación, se intenta modificar el valor de la potencia del puerto oe1/2 de 5 dBm iniciales a 2dBm, que resulta una modificación exitosa.

5. Resumen

En este trabajo se han presentado las bases de un entorno de emulación óptica con la creación automática de agentes de software en redes ópticas definidas por software (SDONs). Se ha comenzado describiendo todas las tecnologías habilitantes que se han utilizado, que incluyen: el protocolo NETCONF que a su vez utiliza el lenguaje de modelado YANG, el framework Netopeer2-Sysrepo para la creación de agentes, los modelos YANG OpenConfig del transpondedor Cassini, la tecnología Docker y finalmente el controlador ONOS. Se ha descrito la arquitectura del entorno de emulación y sus tres componentes, funcionalidades y APIs junto con el proceso de instalación y configuración. A modo de ejemplo, se ha desplegado una topología europea de 18 nodos basados en Cassini en los que se han modificado parámetros como la potencia de lanzamiento óptico, longitud de onda y modulación, recibiendo después notificaciones sobre los cambios realizados.

El desarrollo de este trabajo ha permitido realizar varias contribuciones a los repositorios oficiales de ONOS en GitHub, en concreto al repositorio *ODTN emulator* [35].

6. Conclusiones y líneas futuras

Los resultados de este trabajo han contribuido al desarrollo de un entorno de emulación automática para redes ópticas, emulando tanto el plano de datos como el plano de control.

Para conseguir ese objetivo final, los próximos desarrollos se podrán centrar en proporcionar inteligencia para estas redes ópticas emuladas, dado que ONOS sólo opera dispositivo a dispositivo.

Se plantea enviar las llamadas generadas por los agentes a la herramienta de planificación Net2Plan, que tiene capacidades ópticas. Después, se ejecutarían algunos algoritmos que reestimen las características ópticas de la capa física y, finalmente, este nuevo estado de red se aplicaría a la configuración del agente.

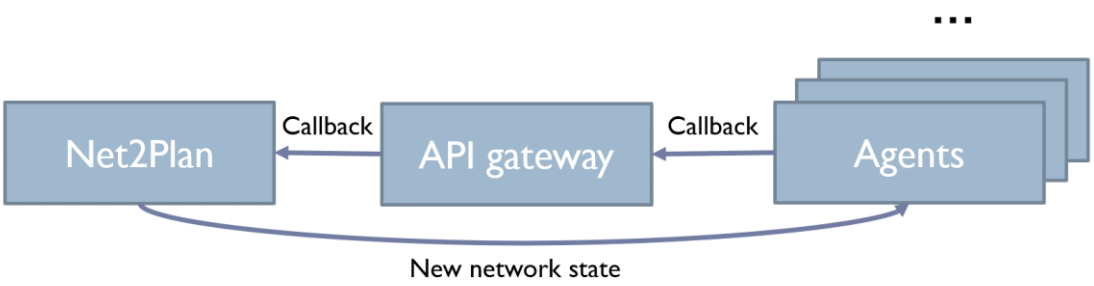

Figura 17: flujo de datos de los callbacks de los agentes en un entorno completo de emulación de redes ópticas

El proceso descrito conforma las directrices de un entorno de emulación de redes ópticas, que simula tanto el plano de control como los aspectos del plano de datos de la red.

7. Bibliografía

[1] D. Kreutz, P. E. V. Ramos, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE 103,* Jan. 2015.

[2] M. Garrich, F. J. Moreno-Muro, M. V. Bueno-Delgado y P. Pavón-Mariño, «Open-Source Network Optimization Software in the Open SDN/NFV Transport Ecosystem,» *JLT,* vol. 37, nº 1, pp. 75-88, 2019.

[3] M. Channegowda,, «Software-defined optical networks technology and infrastructure: Enabling software-defined optical network operations [invited],» *JOCN,* vol. 5, nº 10, pp. A274-A282, 2013.

[4] A. S. Thyagaturu, «Software Defined Optical Networks (SDONs): A Comprehensive Survey,» *IEEE Comm. Surveys and Tutorials,* vol. 18, nº 4, p. 2738–2786, 2016.

[5] T. Szyrkowiec, «Optical Network Models and Their Application to Software-Defined Network Management,» *International Journal of Optics,* 2017.

[6] Open and Disaggregated Transport Network (ODTN) project, [En línea]. Available: https://www.opennetworking.org/odtn/.

[7] M. Bjorklund, «YANG - A data modeling language for the network configuration protocol (NETCONF),» *IETF RFC 6020,* Oct. 2010.

[8] e. a. R. Enns, «Network configuration protocol (NETCONF),» *IETF RFC 6241,* June 2011.

[9] OpenConfig, [En línea]. Available: http://www.openconfig.net/.

[10] OpenROADM, [En línea]. Available: http://www.openroadm.org/.

[11] R. V. R. M. a. R. M. R. Casellas, «SDN Control of Disaggregated Optical Networks with OpenConfig and OpenROADM,» *ONDM,* May 2019.

[12] e. a. E. Riccardi, «An Operator's view on introduction of White Boxes in Optical Networks,» *JLT,* 2018.

[13] Opennetworking, ONOS, 12 2019. [En línea]. Available: https://www.opennetworking.org/wp-content/uploads/2019/12/ONOS-Features_v1.pdf.

[14] Opennetworking ONOS, [En línea]. Available: https://www.opennetworking.org/onos/.

[15] XML-RPC, [En línea]. Available: http://xmlrpc.scripting.com/.

[16] RESTCONF protocol, IETF, 2017. [En línea]. Available: https://tools.ietf.org/html/rfc8040.

[17] Representational State Transfer (REST), [En línea]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer.

[18] IETF, [En línea]. Available: https://tools.ietf.org/html/rfc6020.

[19] Pyang, [En línea]. Available: https://github.com/mbj4668/pyang.

[20] Libyang, [En línea]. Available: https://github.com/CESNET/libyang.

[21] Cassini by TIP, [En línea]. Available: https://cdn.brandfolder.io/D8DI15S7/as/q3wkdg-476u4o-8wg0g7/Cassini_at_a_Glance_-_Telecom_Infra_Project.pdf.

[22] The Telecom Infra Project, [En línea]. Available: https://telecominfraproject.com/.

[23] ODTN drivers for Cassini, [En línea]. Available: https://github.com/opennetworkinglab/onos/tree/master/drivers/odtn-driver/src/main/java/org/onosproject/drivers/odtn.

[24] Docker Containers, [En línea]. Available: https://www.docker.com/resources/what-container.

[25] Net2Plan Network Planning Tool, [En línea]. Available: http://www.net2plan.com/index.php.

[26] ONOS Developer Guide, [En línea]. Available: https://wiki.onosproject.org/display/ONOS/Developer+Guide.

[27] Docker Engine Installation, [En línea]. Available: https://docs.docker.com/get-docker/.

[28] ONOS Dockerhub Repository, [En línea]. Available: https://hub.docker.com/r/onosproject/onos/.

[29] Docker Compose Installation, [En línea]. Available: https://docs.docker.com/compose/install/.

[30] Node.js Installation, [En línea]. Available: https://nodejs.org/en/download/.

[31] Git Installation, [En línea]. Available: https://git-scm.com/download.

[32] Visual Studio Code, [En línea]. Available: https://code.visualstudio.com/.

[33] SDON emulator Github, [En línea]. Available: https://github.com/iicc1/SDON-emulator.

[34] Postman API tool, [En línea]. Available: https://www.postman.com/.

[35] ODTN Emulator Github, [En línea]. Available: https://github.com/opennetworkinglab/ODTN-emulator.

8. Anexo

Anexo 1: *openconfig-terminal-device.yang*

```
module openconfig-terminal-device {

 yang-version "1";

 // namespace
 namespace "http://openconfig.net/yang/terminal-device";

 prefix "oc-opt-term";

 import openconfig-types { prefix oc-types; }
 import openconfig-transport-types { prefix oc-opt-types; }
 import openconfig-if-ethernet { prefix oc-eth; }
 import openconfig-platform { prefix oc-platform; }
 import openconfig-platform-transceiver { prefix oc-transceiver; }
 import openconfig-extensions { prefix oc-ext; }
 import ietf-yang-types { prefix yang; }
```

…

```
 grouping terminal-input-optical-power {
   description
     "Reusable leaves related to input optical power";

   leaf input-power {
     type decimal64 {
       fraction-digits 2;
     }
     units dBm;
     description
       "The input optical power of this port in units of 0.01dBm.
       If the port is an aggregate of multiple physical channels,
       this attribute is the total power or sum of all channels.";
   }
 }
```

…

```
 grouping terminal-ethernet-protocol-state {
```

```
    description
      "Ethernet-specific counters when logical channel
      is using Ethernet protocol framing, e.g., 10GE, 100GE";

      uses oc-eth:ethernet-interface-state-counters;
  }
```

...

```
grouping terminal-otn-protocol-multi-stats {
    description
      "Multi-value statistics containers for logical channels using
      OTN framing (e.g., max, min, avg, instant)";

    container pre-fec-ber {
      description
        "Bit error rate before forward error correction -- computed
        value with 18 decimal precision. Note that decimal64
        supports values as small as i x 10^-18 where i is an
        integer. Values smaller than this should be reported as 0
        to indicate error free or near error free performance.
        Values include the instantaneous, average, minimum, and
        maximum statistics. If avg/min/max statistics are not
        supported, the target is expected to just supply the
        instant value";

      uses oc-opt-types:avg-min-max-instant-stats-precision18-ber;
    }

    container post-fec-ber {
      description
        "Bit error rate after forward error correction -- computed
        value with 18 decimal precision. Note that decimal64
        supports values as small as i x 10^-18 where i is an
        integer. Values smaller than this should be reported as 0
        to indicate error free or near error free performance.
        Values include the instantaneous, average, minimum, and
        maximum statistics. If avg/min/max statistics are not
        supported, the target is expected to just supply the
        instant value";

      uses oc-opt-types:avg-min-max-instant-stats-precision18-ber;
    }

    container q-value {
      description
```

```
      "Quality value (factor) in dB of a channel with two
      decimal precision. Values include the instantaneous,
      average, minimum, and maximum statistics. If avg/min/max
      statistics are not supported, the target is expected
      to just supply the instant value";

    uses oc-types:avg-min-max-instant-stats-precision2-dB;
  }

  container esnr {
    description
      "Electrical signal to noise ratio. Baud rate
      normalized signal to noise ratio based on
      error vector magnitude in dB with two decimal
      precision. Values include the instantaneous, average,
      minimum, and maximum statistics. If avg/min/max
      statistics are not supported, the target is expected
      to just supply the instant value";

    uses oc-types:avg-min-max-instant-stats-precision2-dB;
  }
}
```

...

```
grouping terminal-optical-channel-state {
  description
    "Operational state data for optical channels";

  leaf group-id {
    type uint32;
    description
      "If the device places constraints on which optical
      channels must be managed together (e.g., transmitted on the
      same line port), it can indicate that by setting the group-id
      to the same value across related optical channels.";
  }

  uses oc-transceiver:optical-power-state;

  container chromatic-dispersion {
    description
      "Chromatic Dispersion of an optical channel in
      picoseconds / nanometer (ps/nm) as reported by receiver
      with two decimal precision. Values include the instantaneous,
      average, minimum, and maximum statistics. If avg/min/max
      statistics are not supported, the target is expected to just
      supply the instant value";
```

```
    uses oc-opt-types:avg-min-max-instant-stats-precision2-ps-nm;
  }

  container polarization-mode-dispersion {
    description
      "Polarization Mode Dispersion of an optical channel
      in picosends (ps) as reported by receiver with two decimal
      precision. Values include the instantaneous, average,
      minimum, and maximum statistics. If avg/min/max statistics
      are not supported, the target is expected to just supply the
      instant value";

    uses oc-opt-types:avg-min-max-instant-stats-precision2-ps;
  }

  container second-order-polarization-mode-dispersion {
    description
      "Second Order Polarization Mode Dispersion of an optical
      channel in picoseconds squared (ps^2) as reported by
      receiver with two decimal precision. Values include the
      instantaneous, average, minimum, and maximum statistics.
      If avg/min/max statistics are not supported, the target
      is expected to just supply the instant value";

    uses oc-opt-types:avg-min-max-instant-stats-precision2-ps2;
  }

  container polarization-dependent-loss {
    description
      "Polarization Dependent Loss of an optical channel
      in dB as reported by receiver with two decimal precision.
      Values include the instantaneous, average, minimum, and
      maximum statistics. If avg/min/max statistics are not
      supported, the target is expected to just supply the
      instant value";

    uses oc-types:avg-min-max-instant-stats-precision2-dB;
  }
}
```

Anexo 2: *docker-compose.yml*

```
version: '3'

services:
```

```
    agent:
        build: agent
        ports:
          - '8001-8099:830'
        environment:
          - CALLBACK_URL=http://${SERVER_IP}:${API_SERVER_PORT}/
                  callback

  networks:
    default:
        external:
            name: sdn_optical_network
```

Anexo 3: *Dockerfile*

```
FROM ubuntu:18.04

WORKDIR /root

COPY yang /root/yang
COPY config /root/config
COPY script /root/script

RUN apt-get update && apt-get upgrade -y && apt-get install \
    software-properties-common build-essential openssl libssl-dev \
    libpcre3 libpcre3-dev git make cmake bison flex pkg-config \
    graphviz doxygen valgrind zlib1g zlib1g-dev libev-dev libavl-dev \
    libprotobuf-c-dev protobuf-c-compiler swig python-dev libcurl4-
      openssl-dev \
    lib32ncurses5-dev iproute2 net-tools python-dev python-pip -y

RUN pip install netconf-console requests

RUN echo "export LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib" >> ~/.bash
      rc \
    && export LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib \
    && cd ~ && git clone http://git.libssh.org/projects/libssh.git \
    && cd libssh && mkdir build && cd build \
    && git checkout stable-0.8 \
    && cmake .. && make -j4 && make install \
    && cd ~ && git clone git://git.cryptomilk.org/projects/cmocka.git \
    && cd cmocka && mkdir build && cd build \
    && git checkout tags/cmocka-1.0.1 \
    && cmake .. && make && make install \
    && cd ~ && git clone https://github.com/CESNET/libyang \
    && cd libyang && git checkout v0.16-r3 \
```

```
    && mkdir build && cd build && cmake .. \
    && make -j4 && make install && make test -j4 \
    && cd ~ && git clone https://github.com/CESNET/libnetconf2.git \
    && cd libnetconf2 && git checkout v0.12-r1 \
    && mkdir build && cd build && cmake -DENABLE_TLS=ON -
      DENABLE_SSH=ON .. \
    && make -j4 && make install \
    && cd ~ && git clone https://github.com/sysrepo/sysrepo.git \
    && cd sysrepo && git checkout v0.7.7 \
    && mkdir build && cd build \
    && cmake -DCMAKE_BUILD_TYPE=Release -
      DCMAKE_INSTALL_PREFIX:PATH=/usr .. \
    && make -j4 \
    && ctest && make install \
    && cd ~ && git clone https://github.com/CESNET/Netopeer2.git \
    && cd Netopeer2 && git checkout v0.7-r1 && cd server \
    && mkdir build && cd build && cmake .. \
    && make -j4 && make install \
    && ssh-keygen -t rsa -b 2048 -f /etc/ssh/ssh_host_rsa_key \
    && sh -c 'echo root:root | chpasswd' \
    && cd /root/yang/openconfig-odtn && sh import-yangs.sh

ENTRYPOINT ["sh", "/root/script/push-data.sh"]
```

Anexo 4: *networkConfiguration.json*

```
{
 "devices": {
   "netconf:192.168.31.239:8018": {
     "basic": {
       "name": "Vienna",
       "driver": "cassini-openconfig",
       "locType": "geo",
       "latitude": "48.21",
       "longitude": "16.37"
     },
     "netconf": {
       "ip": "192.168.31.239",
       "port": "8018",
       "username": "root",
       "password": "root",
       "idle-timeout": 0
     }
   },
```

```
  "netconf:192.168.31.239:8017": {
    "basic": {
      "name": "Brussels",
      "driver": "cassini-openconfig",
      "locType": "geo",
      "latitude": "50.85",
      "longitude": "4.35"
    },
    "netconf": {
      "ip": "192.168.31.239",
      "port": "8017",
      "username": "root",
      "password": "root",
      "idle-timeout": 0
    }
  },
  "netconf:192.168.31.239:8016": {
    "basic": {
      "name": "Copenhagen",
      "driver": "cassini-openconfig",
      "locType": "geo",
      "latitude": "55.72",
      "longitude": "12.57"
    },
    "netconf": {
      "ip": "192.168.31.239",
      "port": "8016",
      "username": "root",
      "password": "root",
      "idle-timeout": 0
    }
  },
  "netconf:192.168.31.239:8015": {
    "basic": {
      "name": "Paris",
      "driver": "cassini-openconfig",
      "locType": "geo",
      "latitude": "48.87",
      "longitude": "2.34"
    },
    "netconf": {
      "ip": "192.168.31.239",
      "port": "8015",
      "username": "root",
      "password": "root",
      "idle-timeout": 0
    }
  },
  "netconf:192.168.31.239:8014": {
```

```
    "basic": {
      "name": "Berlin",
      "driver": "cassini-openconfig",
      "locType": "geo",
      "latitude": "52.52",
      "longitude": "13.42"
    },
    "netconf": {
      "ip": "192.168.31.239",
      "port": "8014",
      "username": "root",
      "password": "root",
      "idle-timeout": 0
    }
 }
 ...
},
"links": {
  "netconf:192.168.31.239:8018/231-
    netconf netconf:192.168.31.239:8002/223": {
    "basic": {
      "type": "OPTICAL",
      "metric": "1",
      "durable": true,
      "bidirectional": true
    }
  },
  "netconf:192.168.31.239:8018/219-
    netconf netconf:192.168.31.239:8014/219": {
    "basic": {
      "type": "OPTICAL",
      "metric": "1",
      "durable": true,
      "bidirectional": true
    }
  },
  "netconf:192.168.31.239:8013/216-
    netconf netconf:192.168.31.239:8002/204": {
    "basic": {
      "type": "OPTICAL",
      "metric": "1",
      "durable": true,
      "bidirectional": true
    }
  },
  "netconf:192.168.31.239:8011/223-
    netconf:192.168.31.239:8002/202": {
    "basic": {
      "type": "OPTICAL",
```

```
        "metric": "1",
        "durable": true,
        "bidirectional": true
      }
    },
    "netconf:192.168.31.239:8013/232-
      netconf:192.168.31.239:8011/200": {
      "basic": {
        "type": "OPTICAL",
        "metric": "1",
        "durable": true,
        "bidirectional": true
      }
    }
   ...
  },
  "hosts": {},
  "apps": {},
  "ports": {},
  "regions": {},
  "layouts": {}
}
```