



UNIVERSIDAD NACIONAL DE CÓRDOBA
FACULTAD DE CIENCIAS EXACTAS FÍSICAS Y
NATURALES

PROYECTO INTEGRADOR
INGENIERÍA EN COMPUTACIÓN

**Administración de un muxponder a través
de Redes Definidas Por Software**

Autor:
Matias KLEINER
37590431
kleiner.matias@gmail.com

Director:
Ing. Hugo CARRER
Co-director:
Matthew AGUERREBERRY

Julio 2019

Para mi familia...

Administración de un muxponder a través de Redes Definidas Por Software

Matias KLEINER

"

Resumen

Agradecimientos

Muchas gracias a mi familia, por su apoyo incondicional a lo largo de todos estos años de estudio.

Este proyecto no hubiera sido posible sin el soporte, la confianza, la supervisión y el duro empeño de nuestros directores, Hugo Carrer y Matthew Aguerreberry.

Un especial agradecimiento a mis amigos, y todas las personas que tuve el placer de conocer durante estos años de carrera.

Agradezco a la Fundación Fulgor y a la Fundación Tarpuy, y a todo su personal, por las oportunidades y enseñanzas compartidas.

Finalmente, agradecemos a la Facultad de Ciencias Exactas Físicas y Naturales de la Universidad Nacional de Córdoba por la oportunidad de realizar esta carrera de grado.

Índice general

Resumen	v
Agradecimientos	vii
1. Introducción	1
2. Marco teórico	3
2.1. Redes tradicionales	3
2.1.1. Plano de Control	4
2.1.2. Plano de Datos	4
2.2. Redes Definidas por Software	5
2.2.1. Definición de <i>SDN</i>	5
2.2.2. Arquitectura de <i>SDN</i>	6
Plano de Datos	6
Plano de Aplicación	6
Plano de Control	6
2.3. Gestión de la Red	7
2.3.1. Protocolos de Gestión	7
<i>Command Line Interface</i>	7
<i>Simple Network Management Protocol</i>	8
Otras alternativas	9
2.3.2. <i>NETCONF</i>	9
Definición	10
Conceptos del Protocolo	11
Capacidades	11
Sesión orientada a la conexión	12
Sesión orientada a la autenticación	12
Bases de datos	12
Operaciones del protocolo	13
Notificaciones	15
2.3.3. Lenguaje de Modelado YANG	16
Conceptos del Lenguaje	16
Módulos y submódulos	16
Declaraciones y Definiciones de Datos	18
Identificador de instancia	18
Funcionalidades	19
2.3.4. Redes Ópticas de Transporte	19
<i>Transponders</i> y <i>Muxponders</i>	20
Aplicaciones	21

3. Análisis de las tecnologías	23
3.1. Herramientas de Hardware	23
3.1.1. <i>Muxponder</i> 40Gb	23
3.2. Herramientas de Software	25
3.2.1. Controlador <i>ONOS</i>	25
Arquitectura del controlador	25
Interfaz Southbound en <i>ONOS</i>	27
Justificación de la elección del controlador	28
3.2.2. Análisis de agentes <i>NETCONF</i>	28
Sysrepo	29
Yuma123	29
Evaluación de las implementaciones	29
Justificación de elección del agente	32
4. Diseño e Implementación	33
4.1. Entorno de trabajo	33
4.1.1. Topología	33
4.1.2. Requerimientos del sistema	34
4.2. Integración del protocolo <i>NETCONF</i> al <i>muxponder</i>	36
4.2.1. Requerimientos	36
4.2.2. Compilación instalación del agente	37
4.2.3. Diseño del módulo YANG	38
4.2.4. Diseño de la librería C para el agente <i>NETCONF</i>	41
4.3. Diseño del Driver	44
4.3.1. Requerimientos	45
4.3.2. Descubrimiento del dispositivo	45
4.3.3. Descubrimiento de Enlaces	47
4.3.4. Operaciones definidas en el Driver	47
4.4. Diseño de la interfaz Northbound e Interfaz de usuario	50
4.4.1. Requerimientos	51
4.4.2. Implementación de la REST	51
4.4.3. Implementación de la GUI	52
Bibliografía	57

Índice de figuras

2.1. Comportamiento de dispositivos en redes tradicionales.	3
2.2. Arquitectura de un controlador <i>SDN</i> tradicional.	7
2.3. Operaciones típicas en <i>SNMP</i>	9
2.4. Separación conceptual del protocolo <i>NETCONF</i>	10
2.5. Arquitectura cliente-servidor en el protocolo <i>NETCONF</i>	11
2.6. Ejemplo de comunicación entre cliente y servidor <i>NETCONF</i>	17
2.7. Estructura de un módulo <i>YANG</i>	17
2.8. Ejemplo de identificador de instancia en <i>YANG</i>	18
2.9. Funcionamiento básico de un <i>transponder</i>	21
2.10. Funcionamiento básico de un <i>muxponder</i>	21
2.11. Separación de la red en capa de paquetes y capa de transporte.	22
3.1. Vista del panel frontal del <i>muxponder</i> de 40Gb utilizado.	24
3.2. Arquitectura distribuida de <i>ONOS</i>	26
3.3. Arquitectura completa del controlador <i>ONOS</i>	27
3.4. Interfaz <i>Southbound</i> en <i>ONOS</i>	28
3.5. Demanda de recursos de las implementaciones analizadas.	32
4.1. Topología implementada en el proyecto.	34
4.2. Conexión física de la topología.	34
4.3. Caso de uso desde la perspectiva del administrador.	35
4.4. Requerimientos del sistema.	35
4.5. Topología implementada en el proyecto.	36
4.6. Topología implementada en el proyecto.	37
4.7. Topología implementada en el proyecto.	41
4.8. Topología implementada en el proyecto.	42
4.9. Diagrama de actividad de las operaciones sincronicas con el cliente.	43
4.10. Diagrama de actividad de las notificaciones.	44
4.11. Requerimientos para el driver desarrollado.	45
4.12. Topología implementada en el proyecto.	46
4.13. Topología implementada en el proyecto.	48
4.14. Topología implementada en el proyecto.	49
4.15. Topología implementada en el proyecto.	50
4.16. Topología implementada en el proyecto.	50
4.17. Topología implementada en el proyecto.	51
4.18. Topología implementada en el proyecto.	52
4.19. Topología implementada en el proyecto.	53
4.20. Topología implementada en el proyecto.	54
4.21. Topología implementada en el proyecto.	55
4.22. Topología implementada en el proyecto.	56

Índice de cuadros

2.1. Ejemplo de operaciones disponibles en <i>NETCONF</i>	15
---	----

Lista de acrónimos

API	A pplication P rogramming I nterface
CLI	C ommand L ine I nterface
SNMP	S imple N etwork M anagement P rotocol
GUI	G raphical U ser I nterface
ONF	O pen N etworking F oundation
IETF	I nternet E ngineering T ask F orce
SDN	S oftware D efined N etwork
UDP	U ser D atagram P rotocol
TCP	T ransmission C ontrol P rotocol
MIB	M anagement I nformation B ase
TLS	T ransport L ayer S ecurity
SSH	S ecure S hell
OTN	O ptical T ransport N etwork
OTU	O ptical T ransport U nit
ITU	I nternational T elecommunication U nion
RFC	R equest F or C omments
RPC	R emote P rocedure C all
XML	E Xtensible M arkup L anguage
YANG	Y et A nother N ext G eneration
NETCONF	N ETwork C ONfiguration P rotocol
IANA	I nternet A ssigned N umbers A uthority
FEC	F orward E rror C orrection
FPGA	F ield P rogrammable G ate A rray
ONOS	O pen N etwork O perating S ystem
REST	R Epresentational S tate T ransfer
BSD	B erkeley S oftware D istribution
MIT	M assachusetts I nstitute T echnology

Capítulo 1

Introducción

Capítulo 2

Marco teórico

En este capítulo se comprenderán conceptos teóricos sobre las tecnologías claves en las cuales se basa el proyecto. Como introducción, se analizará el funcionamiento de las redes tradicionales, donde se dejará en evidencia la necesidad de un nuevo paradigma.

Luego, se analizarán los fundamentos en los que se basan las Redes Definidas por Software y por qué este paradigma resuelve los problemas presentados por el enfoque de las redes tradicionales.

También, se introducirán conceptos de lenguajes de modelado y se abordará la importancia de la gestión de la red. Se estudiará *NETCONF* como protocolo de gestión de red. Finalmente, se abordan conceptos de dispositivos ópticos de transporte.

2.1. Redes tradicionales

La infraestructura actual de las redes tradicionales basan su funcionamiento íntegramente en los dispositivos de red [6]. Cada dispositivo lleva su propia gestión sobre el plano de datos y el plano de control de manera local y comunica a los demás dicha información de ser necesario.

Un ejemplo de esto se puede observar en la figura 2.1, donde dos dispositivos intercambian información referente al plano de control.

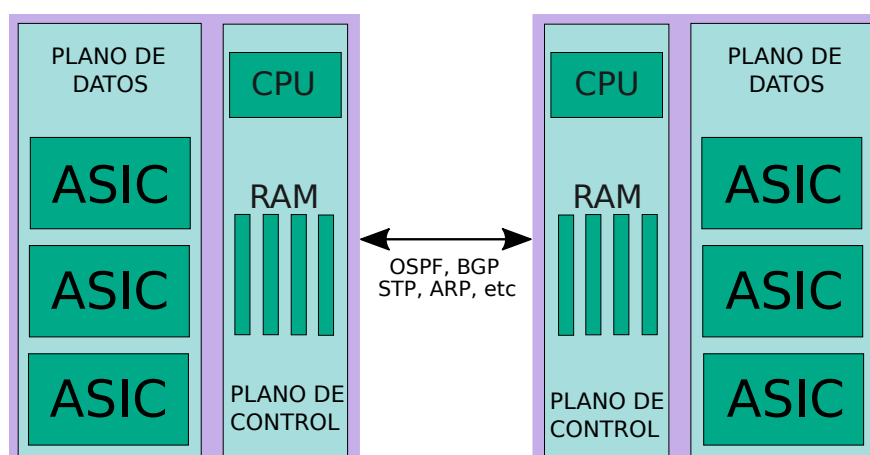


FIGURA 2.1: Comportamiento de dispositivos en redes tradicionales.

2.1.1. Plano de Control

Comprende la configuración del sistema, la administración y el intercambio de información de ruteo entre los dispositivos [30]. Es el responsable de administrar la configuración del equipo y de programar el camino que será usado para el flujo de los paquetes. En otras palabras, es en este plano donde se calculan y se toman las decisiones de enrutamiento y reenvío. En las redes tradicionales, cualquier aplicación que utilice el dispositivo para administrar su configuración reside en esta capa.

El proceso de establecimiento de la topología de red utilizando un plano de control que se ejecuta localmente, es compleja debido a que no existe ningún dispositivo que sea conocido por toda la red. Para gestionar cambios o actualizaciones en cada dispositivo se debe estar conectado a su plano de control de forma individual, lo que no resulta en un enfoque inteligente.

2.1.2. Plano de Datos

También conocido como plano de usuario o plano de reenvío [29], es el encargado de transportar el tráfico de usuario hacia el destinatario final. Tiene como objetivo el reenvío de los paquetes hacia el próximo salto basándose en las decisiones tomadas por la capa de control.

El enfoque dado por las redes tradicionales cumplió con las necesidades de una época donde las arquitecturas cliente-servidor eran dominantes. Tiene como ventaja ser simple a nivel lógico, mientras que el plano de control implica el uso de microprocesadores para tratar los paquetes y conformar las tablas, el plano de datos se desarrolla en silicio. A pesar de ello, presenta una serie de problemas [8]:

- **Funcionalidad de la red integrada en los dispositivos:** El plano de control se encuentra íntegramente en los dispositivos de red, lo que resulta en una configuración de red estática, inflexible y descentralizada.
- **Escalabilidad:** La escalabilidad resulta afectada y no apropiada para la explosión de las nuevas tecnologías como *Big Data*, *Cloud Computing* y el *Streaming*, donde la complejidad de la red incrementa rápidamente debido a que cada dispositivo agregado debe ser configurado y administrado.
- **Políticas inconsistentes:** Si las políticas de configuración cambian a nivel de red, implica un cambio en todos los dispositivos que la componen por parte de los administradores de red.
- **Dependencia del fabricante y personalización:** El plano de control integrado a los dispositivos de red resulta en una dependencia a los ciclos de producción de equipamientos por parte de los fabricantes para incorporar nuevas funcionalidades. Además, con la finalidad de asegurar la calidad de servicio y brindar alta performance, la industria define los protocolos de red de forma específica y aislada, sin el beneficio de una acción conjunta e incapacitando a los operadores a personalizar la red para sus entornos individuales y específicos.

2.2. Redes Definidas por Software

A diferencia de las aplicaciones y los nuevos requerimientos de los usuarios, las redes no han cambiado mucho respecto a los últimos 30 años [7]. El desarrollo de las *SDN* se inició en 1990 donde se introdujo el concepto de funciones programables en la red, teniendo gran innovación en 2001-2007 donde se propone separar el plano de datos del plano de control. El próximo gran paso de las *SDN* llegó en 2007-2010, con la implementación de la *API OpenFlow*.

Las redes definidas por software nacen en respuesta a la dinámica y flexibilidad que requieren las nuevas tendencias, donde el enfoque presentado por las redes tradicionales no cumple dado su naturaleza estática.

2.2.1. Definición de *SDN*

Según la *ONF* [8], la red definida por software, también conocida como red programable o automatizada, consiste en una arquitectura donde el plano de datos se encuentra separado del plano de control y donde este último a su vez puede controlar varios dispositivos.

Tal como destaca *SDx Central* en su reporte [13], este nuevo paradigma presenta las siguientes ventajas:

- **Plano de control centralizado:** A diferencia del enfoque presentado por las redes tradicionales donde se tenía un plano de control distribuido entre los diferentes equipos que conforman la red, ahora se tiene un plano de control centralizado y presente a nivel lógico en un mismo punto. De esta forma, se tiene una visión general y global de toda la red, relajando las comunicaciones entre los dispositivos y las complejidades introducidas por las configuraciones individuales de cada uno. Además, el plano de control ahora es directamente programable, sin tener que usar como intermediario el plano de datos. Todo el tráfico ahora está bajo la supervisión de este nuevo plano de control centralizado, transformando a la red en una red programable.
- **Costos:** Los costos relacionados al control de la gestión del tráfico y de configuración de los diferentes equipos se ven reducidos en tiempo y esfuerzo dado el plano de control centralizado.
- **Automatización:** Un beneficio indirecto de tener un plano de control centralizado, es poder tomar diferentes decisiones y políticas en base a la visibilidad global de la red en tiempo real, aplicando configuraciones en los diferentes equipos de forma automática.
- **Escalabilidad:** *SDN* admite topologías dinámicas con capacidades para adaptarse a cambios, debido a la automatización de la configuración de los dispositivos. Con la capacidad de ajustar los picos y las bajas en la carga del tráfico, las empresas pueden crear e implementar nuevos servicios y aplicaciones sin demora debido a la infraestructura más flexible.
- **Mantenimiento y monitoreo:** Por medio del controlador *SDN* se puede conocer, en cualquier momento, el estado actual de la red incluyendo los dispositivos que la componen.

- **Seguridad:** Dado que la administración de toda la red se realiza en un solo punto, se asegura que no existan debilidades o inconsistencias en las configuraciones de las aplicaciones y los equipos.

2.2.2. Arquitectura de SDN

En las redes tradicionales, cada dispositivo tiene integrado tanto el plano de datos como el plano de control. En *SDN*, el plano de datos se encuentra desacoplado del plano de control y, además, se puede diferenciar un nuevo plano llamado *plano de aplicación* [31]. A continuación, se analizará cuál es la función que cumple cada plano en esta nueva arquitectura propuesta por las *SDN*.

Plano de Datos

Comprende la misma funcionalidad que en las redes tradicionales. Consiste en un conjunto de dispositivos de red con funcionalidades de reenvío de paquetes.

Plano de Aplicación

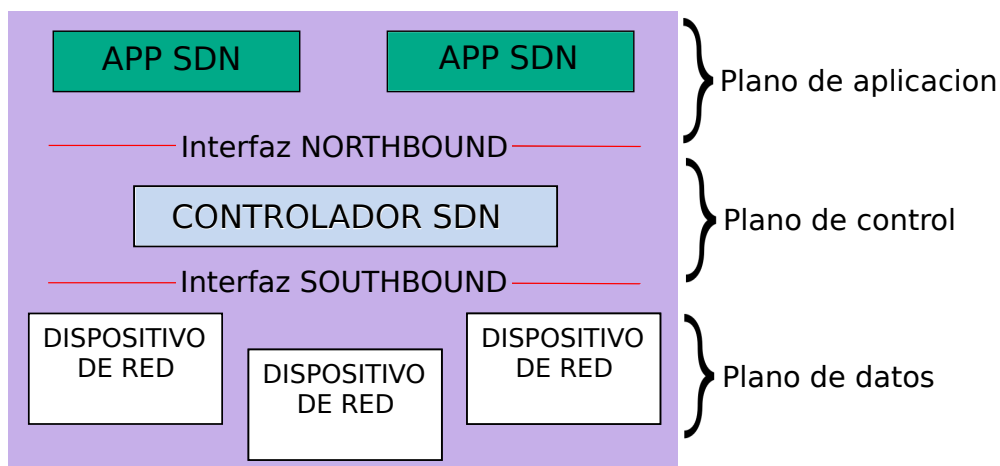
Con el enfoque de las redes tradicionales, el plano de aplicación se encontraba integrado en el plano de control. En *SDN*, el plano de aplicación se desacopla al igual que el plano de control. En este plano se encuentran las aplicaciones de red que implementan las funcionalidades de más alto nivel y que participan en las decisiones de administración y control de ruteo.

Plano de Control

Toda la función de control se encuentra centralizada fuera de los dispositivos, permitiendo a los desarrolladores de aplicaciones utilizar las capacidades de la red pero haciendo una abstracción de su topología o sus funciones. Tiene como objetivo mediar, organizar y facilitar la comunicación entre los diferentes equipos y las aplicaciones. Además, este plano ahora está disponible para poder ser programado desde un software externo al controlador.

En la figura 2.2, se expone la anatomía de un controlador *SDN*. En ella, se puede observar dos interfaces comprendidas por el plano de control [36]: *Southbound* y, *Northbound*.

- ***Southbound API*:** necesaria por la separación del plano de control del plano de datos. Define la *API* de comunicación entre el controlador y los diferentes dispositivos de red, en otras palabras, entre el plano de control y el plano de datos.
- ***Northbound API*:** funciona como interfaz tanto de alto como de bajo nivel, es necesaria para permitir que las aplicaciones que se ejecutan en la parte superior de *SDN* puedan comunicarse con el mismo. En el primer caso, la interfaz provee una abstracción de la red en sí misma, permitiendo a los desarrolladores no tener que preocuparse por los dispositivos individuales, sino manejar la red como un todo. En el segundo caso, la interfaz advierte a las aplicaciones sobre la existencia de los dispositivos individuales y sus enlaces, pero oculta las diferencias entre los dispositivos.

FIGURA 2.2: Arquitectura de un controlador *SDN* tradicional.

2.3. Gestión de la Red

En la actualidad se puede encontrar una gran variedad de redes, desde pequeñas redes domésticas de intranet hasta redes empresariales o de proveedores de servicios. Cada una de estas redes tienen diversos requerimientos de gestión. Las pequeñas redes domésticas, que consisten en unos pocos dispositivos conectados, requieren una sobrecarga de administración baja, y con frecuencia, pueden gestionarse manualmente de forma eficiente. No así las redes más grandes, que podrían contener cientos de dispositivos conectados requiriendo un enfoque más sistemático para hacer frente a las complejidades que surgen debido al tamaño de la red. A medida que la red crece en estructura y complejidad, se hace evidente la necesidad de una solución eficiente para la gestión de la misma [35].

2.3.1. Protocolos de Gestión

Existen múltiples formas de llevar a cabo la administración de la configuración en los diversos dispositivos que conforman la red. En esta sección, se analizarán dos alternativas: *CLI* y *SNMP*.

Command Line Interface

CLI es el enfoque más común en el ámbito de gestión de la configuración, adoptado por múltiples empresas. Consiste en un método para comunicarse con las aplicaciones que la subyacen, a través de una interfaz de usuario simple basada en texto. De esta forma, permite que el administrador pueda ingresar instrucciones en una línea de comandos a través de una terminal y recibir las respuestas en la misma. La aplicación subyacente es la encargada de procesar la instrucción y devolver alguna respuesta al usuario. Generalmente, las respuestas están orientadas a que resulten fáciles de entender para las personas, sin embargo, no se encuentran orientadas a las APIs, ya que no existe un formato o un estándar de cómo representar dichas respuestas. Además, las implementaciones internas podrían ser diferentes entre los distintos dispositivos, incluso entre dispositivos del mismo fabricante, de modo que, tanto los comandos como las respuestas podrían variar significativamente entre los

equipos.

Un ejemplo de una operación en *CLI* puede verse en la figura 2.1. La primer línea ingresada, hace referencia a un acceso al modo de configuración de un dispositivo cualquiera. En la segunda, se agrega una entrada estática a la tabla de ruteo y en la tercera se abandona el modo de configuración.

LISTING 2.1: Interacción típica con un dispositivo mediante *CLI*.

```
> configure terminal
#> ip route 192.0.2.0/8 ethernet 1/2 192.0.2.4
#> !
```

Este enfoque presenta una serie desventajas [28]. En primer lugar, la implementación de las aplicaciones subyacentes a la *CLI* no están estandarizadas, por lo que las operaciones varían drásticamente entre dispositivos de diferentes fabricantes e incluso en implementaciones *CLI* del mismo fabricante. A su vez, los fabricantes podrían brindar una actualización de software del dispositivo, donde los comandos *CLI* de la versión anterior se vean modificados o eliminados, lo que no solo se traduce a problemas para el administrador de red, sino también para las *API's* que hagan uso de la *CLI*.

En segundo lugar, realizar un cambio en el estado de un dispositivo podría requerir múltiples transacciones, y en el caso de que alguna de estas falle, el dispositivo podría quedar en un estado inconsistente. Por ejemplo, en la figura 2.1 se observó que para realizar una operación sencilla como agregar una entrada a la tabla de ruteo, implicó el uso de al menos tres comandos. De forma similar, podrían existir operaciones que requieran de transacciones con una mayor cantidad de instrucciones. *CLI* no define de forma estándar una solución para deshacer los cambios aplicados en el dispositivo.

Simple Network Management Protocol

SNMP es un protocolo de monitoreo y administración de red, estandarizado por primera vez en 1988 por la *IETF* [1]. Su funcionamiento se basa en una arquitectura cliente-servidor, donde los mensajes se intercambian a través del protocolo de transporte no orientado a la conexión *UDP*. Consiste en una colección de agentes y administradores formando entre ellos una red, donde se denomina administrador a aquel dispositivo que tiene el rol de ejecutar aplicaciones de administración de red, mientras que los dispositivos que requieran ser administrados se denominan agentes [2].

Las capacidades para administrar la red en *SNMP*, quedan representadas en lo que se conoce como *MIB*. Una *MIB*, es una base de datos que contiene información jerárquica y estructurada en forma de árbol de todos los parámetros gestionables de la red. Dicha base de datos se debe cargar en el administrador *SNMP*, para ello cada agente *SNMP* expone al administrador *SNMP* una serie de módulos *MIB*. Con esta información el administrador podría alterar dinámicamente la configuración del agente.

El uso de *SNMP* como monitoreo es una práctica común desde su publicación, sin embargo, se desalentó su uso en áreas de gestión de configuración por las siguientes razones [25]:

- Problemas inherentes al protocolo de transporte *UDP*, donde los mensajes pueden perderse o llegar desordenados, así como también la falta de mecanismos de seguridad para los mismos jugaron un papel importante para reemplazar *SNMP* por otros protocolos de administración de red.
- No existe una estandarización de los módulos *MIB* para configurar las funciones de red. El trabajo de descubrir correctamente los módulos *MIB* para cada dispositivo es tarea del usuario, lo que resulta compleja y no eficiente.

La figura 2.3 muestra las operaciones más comunes de *SNMP*.

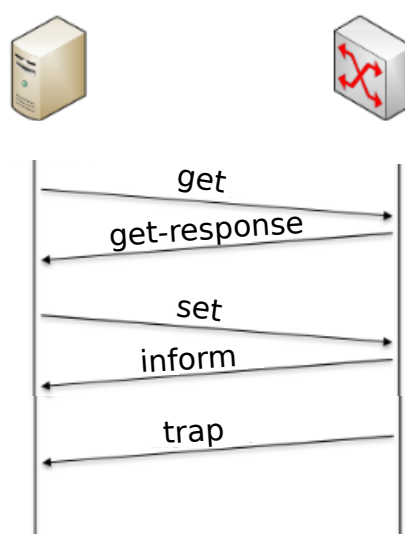


FIGURA 2.3: Operaciones típicas en *SNMP*.

Otras alternativas

Algunos enfoques para la gestión de la red pueden incluir soluciones basadas en páginas web, que permiten al administrador modificar las configuraciones en el dispositivo de forma gráfica y más amigable, pero generalmente resultan más limitadas que las *CLI*. Además, algunos dispositivos pueden brindar soluciones propietarias para la gestión de la configuración, sin embargo, estas soluciones suelen ser muy específicas a un dispositivo o una familia de dispositivos, y rara vez suelen ser compatibles entre sí. Estos últimos también representan una carga para los administradores, donde cada solución requiere que el administrador aprenda otra manera de configurar las funcionalidades de la red.

2.3.2. *NETCONF*

Esta sección repasa brevemente los conceptos y características principales que ofrece el protocolo *NETCONF*. Además, aspectos de seguridad, transporte y control de acceso del protocolo se discuten en detalle.

Definición

NETCONF fue estandarizado por la *IETF* por primera vez en el 2006, en el *RFC* 4741 [16]. Actualmente está siendo adoptado por los principales proveedores de dispositivos de red y ha ganado el apoyo de la industria. Según detalla Carl Moberg [14], podemos encontrar que fabricantes como Juniper, Huawei, Cisco, entre otros, brindan soporte desde hace tiempo del protocolo *NETCONF*.

La *IETF* define a *NETCONF* como un protocolo estándar para Instalar, manipular y borrar configuraciones en un dispositivo [17]. Permite implementar una *API* formal utilizando el lenguaje de modelado *YANG* para administrar y monitorear las funcionalidades de la red. *NETCONF* utiliza el paradigma de las *RPC*, donde construye los mensajes que intercambian información como un flujo con codificación *XML*. Funciona con una arquitectura cliente-servidor, donde los mensajes son transportados utilizando algún protocolo orientado a la conexión. El *RFC* 6241, en la sección 1.2, menciona una partición conceptual del protocolo en cuatro capas, dicha partición se refleja en la figura 2.4.

A continuación, se explica qué función cumple cada una de estas capas.

- **Capa de transporte seguro:** provee mecanismos de comunicación entre cliente y servidor.
- **Capa de mensajes:** encargada de la codificación y partición de los mensajes.
- **Capa de operación:** define las operaciones admitidas por el protocolo.
- **Capa de contenido:** relaciona la representación y el modelado de los datos en el protocolo.

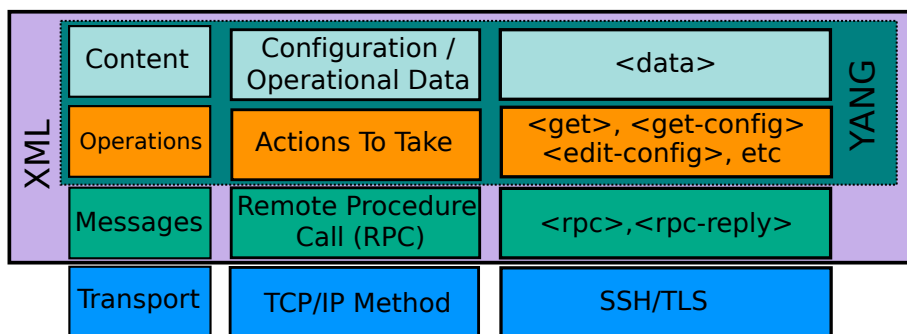


FIGURA 2.4: Separación conceptual del protocolo *NETCONF*.

Las características que destacan a *NETCONF* como protocolo de administración de red son [19]:

- Capacidad de restauración de los datos y *backup* de la configuración.
- De uso fácil, presentando la información de forma estructurada con una codificación entendible para las personas y las *API*'s.
- Implementa mecanismos de control de errores mediante validación de sintaxis y semántica.

- Separación clara de los datos de configuración y los datos de estado.
- Posibilidad de gestionar la configuración en un dispositivo de manera reactiva mediante notificaciones del mismo.

NETCONF separa los datos de configuración de los datos de estado de un dispositivo. Según lo detallado en la sección 1.1. y 1.4 del RFC 6242, se define a cada uno como:

- **Datos de configuración:** información que se puede leer o escribir y que se utiliza para llevar al dispositivo de un estado inicial a un estado deseado. Un ejemplo es la velocidad del ventilador del cpu del dispositivo.
- **Datos de estado:** representa información de sólo lectura y estadísticas brindadas por el dispositivo. Por ejemplo, la temperatura del cpu del equipo.

Conceptos del Protocolo

Como se mencionó anteriormente, *NETCONF* define un protocolo de administración de red con arquitectura cliente-servidor, donde el cliente en este caso es el sistema de administración de la red o el administrador del sistema, mientras que el dispositivo que contiene una o más funciones de red que deben ser administradas, actúa de servidor. El cliente y el servidor inician la sesión de protocolos mediante un primer mensaje que da lugar al intercambio de capacidades o *capabilities*, donde se definen qué operaciones estarán disponibles para su uso. Este primer mensaje recibe el nombre de *HELLO* [17]. La figura 2.5 ejemplifica la arquitectura presentada por el protocolo.

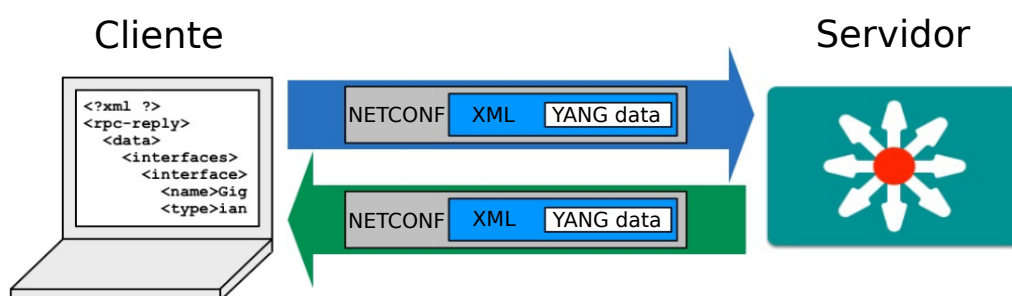


FIGURA 2.5: Arquitectura cliente-servidor en el protocolo *NETCONF*.

Capacidades

El protocolo *NETCONF* está diseñado para ser altamente extensible y, con este fin, es compatible con el intercambio inicial de capacidades entre cliente y servidor [17]. Este intercambio de información permite al cliente ajustar sus comportamientos basándose en las funcionalidades que admite el servidor. Cada capacidad establecida por el protocolo recibe un nombre asignado por la IANA. Además, también se incluye el intercambio de los modelos YANG que tiene implementado el servidor, lo cual es necesario no solo para que el cliente pueda aprender de los mismos, sino

también para reconocer las diferentes revisiones implementadas en el servidor.

La utilidad de esta característica reside en que, a través del intercambio de las capacidades entre el cliente y el servidor, el protocolo define cuáles serán las operaciones admitidas desde el inicio de la sesión, evitando así el ingreso de comandos de configuración incorrectos o no soportados.

Sesión orientada a la conexión

La sección dos del *RFC 6242*, referida a protocolos de transporte, detalla que *NETCONF* no está vinculado a ningún protocolo de transporte específico. El requisito necesario de *NETCONF* para el protocolo de transporte subyacente es que el mismo sea orientado a la conexión.

Esta es una de las principales ventajas frente a *SNMP*, donde los mensajes en este último eran transportados a través del protocolo no orientado a la conexión, *UDP*. Además, el hecho de que *NETCONF* no especifique el uso de un único protocolo de transporte orientado a la conexión, se traduce a una mayor flexibilidad y personalización para el administrador, pudiendo optar por aquella que mejor se ajuste a las necesidades de los equipos involucrados.

Sesión orientada a la autenticación

El protocolo *NETCONF* es orientado a la sesión con autenticación, utilizando una arquitectura cliente-servidor donde el servidor escucha un puerto asignado para recibir las conexiones con los clientes.

Según la sección dos del *RFC 6242* referida a seguridad, el protocolo mínimamente debe ofrecer autenticación, confidencialidad e integridad. Cualquier mensaje *NETCONF*, incluido el mensaje *HELLO*, se envían únicamente si el cliente y servidor se han autenticado de forma correcta. No se especifica un protocolo en particular, pudiendo utilizarse alguno de los múltiples protocolos de transporte seguros existentes en la actualidad como *TLS*, *SSH*, *BEEP*, etc. Cualquier implementación de *NETCONF* debe, al menos, soportar *SSH* como protocolo de transporte seguro.

Además, según el *RFC 6536* relacionado al control de acceso de usuarios, *NETCONF* admite una jerarquía de niveles de usuarios. Por ejemplo, tener dos grupos de usuarios donde uno tenga permisos de configuración más limitados que el otro.

Bases de datos

NETCONF define en la sección cinco del *RFC 6242*, la existencia de uno o más *datastores*, los cuales cumplen el papel de una base de datos conceptual que puede ser utilizada para almacenar y acceder tanto a los datos de configuración como a los datos de estado. El protocolo especifica y define tres tipos de base de datos: *running*, *startup* y *candidate*, de las cuales únicamente es obligatorio que se implemente la primera. Si la implementación admite otras bases de datos, como por ejemplo *startup* o *candidate*, el servidor informará al cliente esta capacidad en el mensaje *HELLO*. Cada operación en *NETCONF* debe especificar la base de datos a la cual se realizará la consulta o modificación.

A continuación, se detalla cada uno de los almacenes de datos mencionadas.

- **startup**: según lo especificado en la sección 8.7 del RFC 6242, dicha base de datos se utiliza para almacenar de forma persistente la información de configuración del dispositivo. El contenido de esta es copiado de manera automática a la base de datos conocida como *running* en el inicio del servidor *NETCONF*. De esta forma, el protocolo brinda una herramienta al dispositivo para poder aplicar una configuración dado al inicio del equipo.
- **running**: refleja la configuración actualmente en uso por el dispositivo. Es la única base de datos conceptual que admite la presencia tanto de datos de estado como datos de configuración. A alto nivel, esta base de datos diferencia del estado de *startup*, puesto que no es una configuración que será aplicada al inicio sino que refleja la configuración actual del dispositivo.
- **candidate**: se encuentra definido en la sección 8.3 del RFC 6242. Puede ser utilizado para realizar cambios que no se van a aplicar al dispositivo de forma directa, sino que lo harán una vez se realice un *commit* sobre dicha base de datos. De esta forma, el contenido de *candidate* es copiado a *running*. Si de lo contrario se desea descartar los cambios realizados en este *datastore*, la operación *discard-changes* copia el contenido de *running* a *candidate*. En esta base de datos conceptual únicamente se admiten datos de configuración. En otras palabras, la utilidad de esta base de datos reside en que permite brindar al administrador un entorno de pruebas, donde se podría aplicar una configuración temporal en el equipo, con capacidad de volver a la configuración anterior en caso de fallas.

Como se mencionó anteriormente, cualquier implementación de *NETCONF* debe admitir al menos el *datastore running*, esto es necesario ya que los datos de estado (necesarios para monitorear el dispositivo) únicamente se encuentran admitidos en dicho *datastore*.

Por último, se podría hacer una analogía entre la separación de los datos de estado y los datos de configuración con la separación conceptual de dichas bases de datos lógicas. En el primer caso, se busca distinguir entre un dato de solo lectura de otro que admite la escritura, mientras que el segundo trata de diferenciar entre un conjunto de estados bien definidos que puede alcanzar el dispositivo. Por ejemplo, distinguir la configuración que va a aplicarse únicamente en el inicio del dispositivo a través del *datastore startup*, de la configuración que podría llevar en un determinado momento a través del *datastore running*.

Operaciones del protocolo

Las operaciones en el protocolo *NETCONF* se definen como *RPC* en los modelos YANG relevantes. En dichos modelos también se definen los argumentos de entrada y los contenidos de salida para cada operación. Todas las operaciones están codificadas en *XML* dentro de los mensajes *RPC* que son, de hecho, los únicos mensajes que los clientes pueden enviar en las sesiones de *NETCONF* después del intercambio inicial del mensaje *HELLO*.

Como las operaciones son *RPC*, cada mensaje enviado por los clientes tendrá una respuesta por parte del servidor. Este resultado normalmente contiene *ok* para indicar que la operación resultó según lo esperado, o *error* indicando las razones por la cual falló dicha operación.

El protocolo define en la sección 7 del *RFC 6241* nueve operaciones básicas y necesarias para cualquier implementación del mismo, las cuales se describen a continuación:

- **get**: utilizado para consultar tanto datos de configuración como datos de estado al servidor *NETCONF*.
- **get-config**: operación que devuelve los datos de configuración del dispositivo. Puede incluir filtros para limitar la información enviada por parte del servidor.
- **edit-config**: definida para crear, actualizar o borrar datos de configuración en el servidor. Únicamente se admite esta operación en las bases de datos *running* o *candidate*.
- **copy-config**: crea o reemplaza completamente el contenido de una base de datos por otra. El caso de uso más común de esta operación es para copiar el contenido del *datastore running* al *datastore startup*.
- **delete-config**: Elimina completamente el contenido de un *datastore* determinado. No se admite esta operación para la base de datos *running*.
- **lock**: permite al cliente bloquear la configuración completa de un *datastore* específico en un dispositivo. Tales bloqueos son destinados a ser de corta duración, de esta forma un cliente puede realizar un cambio sin temor a la interacción con otros clientes de *NETCONF*. Además, como el protocolo es orientado a la sesión, todos los recursos tomados por la misma tales como los *datastores*, deben ser liberados en el momento de la finalización o cierre de la sesión.
- **unlock**: permite a la sesión liberar el recurso tomado por la operación *lock*.
- **close-session**: utilizada para finalizar la sesión entre cliente y servidor *NETCONF*. Cualquiera de las operaciones mencionadas en esta sección, quedan inhabilitadas una vez finalizada la sesión.
- **kill-session**: permite al administrador de red finalizar alguna sesión inactiva que tiene recursos tomados.

Además de estas nueve operaciones descritas por el protocolo, pueden proporcionarse operaciones adicionales basado en las capacidades anunciadas por el dispositivo, como por ejemplo operaciones *RPC* definidas en los módulos *YANG*.

También, *NETCONF* admite operaciones con capacidades más avanzadas. No es obligatorio que las diferentes implementaciones del mismo soporten las siguientes características, más bien, de hacerlo deben ser expuestas como capacidades admitidas en el mensaje *HELLO*. Dichas operaciones se describen a continuación:

- **commit:** operación utilizada para copiar atómicamente el contenido del *datastore candidate* al *datastore running*. Además, puede incluirse la operación *confirmed-commit*, esta última funciona como un *backup* de la configuración previa al *commit*, la cual se restablece al cabo de un *timeout* si no se recibe la operación *confirmed-commit*. *NETCONF* describe a esta última como una “confirmación de la confirmación”.
- **discard-changes:** revierte una operación que está en espera de confirmación. En otras palabras, se copia el contenido del *datastore running* al *datastore candidate*.
- **validate:** consiste en una operación que verifica la correctitud semántica y sintáctica de una configuración antes de aplicar el cambio en el dispositivo.

La tabla 2.1 resume las diferentes operaciones disponibles en *NETCONF* y a qué *datastore* podría aplicarse cada una de ellas.

Capacidad	Operación	Base de datos afectada
writable-running	lock edit-config unlock copy-config	running running running running ->startup
candidate	lock edit-config commit validate unlock copy-config	candidate candidate candidate ->running candidate candidate running ->startup
confirmed-commit	lock edit-config commit confirmed-commit validate unlock copy-config	candidate candidate candidate candidate ->running candidate candidate running ->startup

CUADRO 2.1: Ejemplo de operaciones disponibles en *NETCONF*

Notificaciones

Si bien *NETCONF* está diseñado principalmente para la administración de la configuración de la red mediante las operaciones expuestas anteriormente, existe una poderosa herramienta de monitoreo implementada en el protocolo llamada notificaciones. La *RFC 5277* define a las mismas como un servicio de entrega de mensajes asíncronas a los clientes mediante suscripción. Esta característica no es obligatoria para las diferentes implementaciones del protocolo. De soportarlo, el servidor deberá comunicar a los clientes dicha característica como una capacidad del servidor en el mensaje *HELLO*.

Esta herramienta es similar a las notificaciones en el protocolo *SNMP*, pero tiene la ventaja de que, en *NETCONF*, el cliente puede especificar a qué notificación particular se desea suscribir, lo que permite un monitoreo más flexible. Además, como se mencionó anteriormente, el servidor puede declarar permisos para los diferentes usuarios y sesiones por lo que las notificaciones serán enviadas únicamente a aquellos clientes suscritos y que cumplan con el nivel de acceso requerido por el servidor.

La importancia de las notificaciones reside en que los dispositivos de red tienen variables críticas que deben ser monitoreadas, por ejemplo la temperatura del equipo, el estado de los enlaces, la conectividad entre los mismos, etc. Dichas variables críticas reciben el nombre de alarmas.

De no existir un mecanismo de mensajes asíncronos, el monitoreo de las alarmas de un dispositivo podría implicar una sobrecarga en la red, debido a la cantidad de consultas periódicas que existirían sobre los dispositivos.

De esta forma, las notificaciones que define el protocolo *NETCONF* no solo implica un monitoreo eficiente mediante mensajes asíncronos, sino que permite al protocolo poder tomar medidas de forma reactiva a las diferentes alarmas que se presenten. Por ejemplo, se podría configurar al cliente *NETCONF* para que, de recibir una alarma de exceso de temperatura en el equipo, configure de forma automática una velocidad mayor en el ventilador del mismo.

Para finalizar, la figura 2.6 refleja una interacción típica entre cliente y servidor donde se observa el intercambio de capacidades en los mensajes *HELLO*, el uso de diferentes operaciones con respuestas *RPC* y las notificaciones.

2.3.3. Lenguaje de Modelado YANG

Como se mencionó anteriormente, *NETCONF* utiliza *YANG* para modelar los datos de estado, los datos de configuración, las *RPC* y las notificaciones. *Yet Another Next Generation* es un lenguaje de modelado de datos desarrollado y estandarizado en la *RFC 6020* por la *IETF* en el año 2010 [39]. Si bien existen en la actualidad lenguajes de modelado como *XML Schema*, *SMI*, *UML*, entre otros, la ventaja que presenta *YANG* frente a los demás es que es un lenguaje de modelado específico para gestión de la configuración de red.

Conceptos del Lenguaje

YANG define, en la sección 4.1 de la *RFC 6020*, las funcionalidades de la red separando los datos de estado de los datos de configuración y presentando la información como una estructura de árbol jerárquica. Consiste en una serie de declaraciones y tipos que pueden ser usadas para definir los datos que se quieren modelar. Estas definiciones son contenidas en un módulo y describen qué tipo de datos admite una variable. A su vez, un módulo puede heredar definiciones de otro módulo.

Módulos y submódulos

Definen una estructura para el modelado de datos. Tienen un diseño predefinido que se debe seguir. Este diseño comienza con un encabezado, siguiendo de las declaraciones que contenga el módulo y por último las revisiones y comentarios respecto

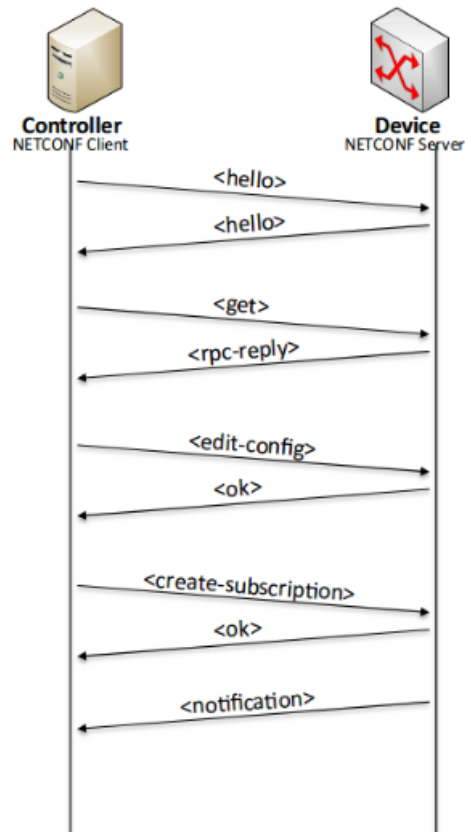


FIGURA 2.6: Ejemplo de comunicación entre cliente y servidor NETCONF.

al mismo.

Se define el nombre del módulo, un prefijo para identificarlo, las dependencias, información de contacto al autor, descripción y revisiones. La declaración “include” permite referenciar material que se describe en un submódulo, mientras que la declaración “import” permite referenciar material que se encuentra descrito en un módulo externo. La estructura básica de un módulo puede verse en la figura 2.7.

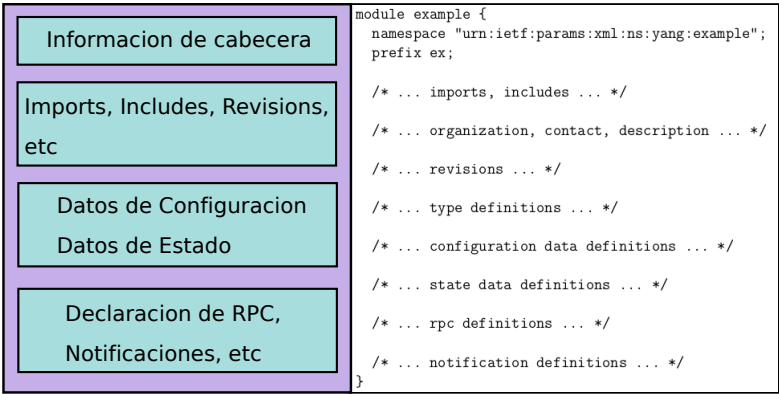


FIGURA 2.7: Estructura de un módulo YANG.

Declaraciones y Definiciones de Datos

A continuación, se describen algunas sentencias que podría contener un módulo YANG. Cada sentencia contiene la definición del tipo de dato y puede contener además algún valor para ese tipo de dato. Siempre representan a datos de configuración o datos de estado, realizando dicha distinción con la sentencia llamada “*config*”.

- **leaf**: contiene un dato simple como un entero o un *string*. Admite exactamente un valor para un tipo de dato particular y opcionalmente puede incluir una descripción.
- **leaf-list**: describe una secuencia de datos tipo *leaf*. Cada *leaf* admitirá un solo valor para el tipo de dato que especifique el *leaf-list*.
- **container**: es utilizado para agrupar datos lógicamente relacionados. Un *container* no admite un valor, pero sí admite cualquier número de tipos de datos como *leaf*, *leaf-list*, *container* o *list*.
- **list**: define una secuencia de tipo de datos donde cada tipo de dato es única, identificado por la sentencia *key*. Puede contener múltiples identificadores *key* y cualquier cantidad de tipo de datos *leaf*, *leaf-list*, *container*, etc.
- **choices - cases**: no describen algún tipo de dato, más bien ofrecen ramificaciones condicionales en la estructura del módulo. La sentencia *choice* es una condición que asegura que, como máximo, se cumpla una de las declaraciones dadas por case.

Las declaraciones y sentencias descritas anteriormente pueden ser utilizadas en conjunto para poder formar una estructura de datos tipo árbol más compleja. Además, YANG admite la reutilización de sentencias mediante las declaraciones *include* e *import* reduciendo así los posibles errores en el modelado de datos.

Identificador de instancia

Cada dato en YANG, así como el propio módulo, tiene un identificador único de instancia que se puede utilizar para referirse a él. Los identificadores se denominan *namespace*, y admiten un prefijo para poder acortar el nombre.

Por ejemplo, Bjorklund [3], definió un módulo YANG para la administración de interfaces. Dicho modelo tiene una estructura de datos de tres niveles para una interfaz básica. En el nivel superior del modelo se encuentra definido el *container* llamado “*interface*”, seguido de la *list* “*interface*” que contiene múltiples instancias de una interfaz, identificada por la *key* “*name*”. Además, cada interfaz tiene una *leaf* “*enabled*” que describe el estado de la misma. Un ejemplo de identificador para una instancia de “*interface*” llamada “*eth0*” puede verse en la figura 2.8.

```
/if:interface/if:interface['eth0']/if:enabled
```

FIGURA 2.8: Ejemplo de identificador de instancia en YANG.

Funcionalidades

YANG ofrece características especiales que lo distinguen de un documento *JSON*, permitiéndole describir de forma eficiente las funcionalidades de la red. Estas características incluyen la validación de modelos, una forma estandarizada de extender a los módulos y compatibilidad entre las diferentes revisiones de los mismos. En esta sección, se analizaron las principales funcionalidades ofrecidas por YANG.

- **Validación:** una de las características más importantes de YANG es la posibilidad de validar automáticamente todos los datos descritos en el modelo. Resulta importante ya que la validación de los datos es una tarea difícil. Dicha afirmación está respaldada por el hecho de que introducir datos erróneos y tomarlos como válidos, está catalogada como la principal amenaza de seguridad según OWASP [26], organización sin ánimo de lucro a nivel mundial dedicada a mejorar la seguridad de las aplicaciones y del software en general. Cada dato introducido en el modelo YANG puede ser validado semánticamente y sintácticamente. La validación de sintaxis es automática y garantiza que el dato contenga una secuencia de bytes válido, puesto que cada dato en el modelo tiene asociado un *type* (string, int, uint, etc). Por otra parte, la validación semántica resulta más compleja y puede ser usada para describir dependencias entre datos. YANG también admite sentencias como “*when*” o “*must*” que pueden ser usadas para evaluar condicionalmente un dato.
- **Compatibilidad:** cada módulo admite la indicación de una revisión, esto permite a YANG distinguir las versiones soportadas y adaptarse a la situación cuando la misma no es soportada. También, se describen reglas de actualización en los módulos que deben respetarse para mantener compatibilidad entre los modelos de datos anteriores. Por ejemplo, cualquier cambio en un módulo debe indicar una revisión en la cabecera, tanto el nombre del mismo como el namespace debe mantenerse, como así también las definiciones de datos obsoletas, lo que permite compatibilidad con modelos de datos anteriores. Esta característica permite a los módulos evolucionar con el paso del tiempo, sin romper aplicaciones existentes con versiones anteriores.
- **Extensión:** permite extender las funcionalidades de los módulos con nuevas definiciones de datos. Existen muchas razones por las cuales utilizar la extensión en YANG, como por ejemplo, desarrollar un nuevo módulo a partir de uno existente o con el fin de reducir errores reutilizando un módulo funcional. Una ventaja importante que tiene utilizar la extensión, es que al agregar nueva información en un módulo, se mantiene compatibilidad con el heredado.

2.3.4. Redes Ópticas de Transporte

La explosión del tráfico digital provocado por los nuevos enfoques como *Big Data* o el *Streaming*, y los requerimientos de los usuarios donde existe un constante crecimiento de aplicaciones con alta demanda de ancho de banda, requieren de una nueva tecnología de transporte que pueda ocuparse de los patrones de tráfico y los contenidos de datos modernos. Para ello, se han realizados numerosos avances en los últimos años referente al plano de control y el plano de datos de las redes ópticas [27], surgiendo protocolos como SONET o OTN. En esta sección, se analiza las redes

ópticas, utilizadas para el transporte de los datos como así también los dispositivos que funcionan sobre dichas redes.

Una red de transporte óptica, es un tipo de red de comunicaciones de datos que utiliza la luz como medio de transporte para la información [23]. A diferencia de las redes basadas en cobre, los pulsos de luz de una red óptica pueden transportarse a una distancia considerable e incluso regenerarse a través de un dispositivo repetidor óptico. Después de que una señal óptica es recibida en su red de destino, la misma se convierte en una señal eléctrica a través de un receptor óptico, para luego ser enviado al nodo de la capa de paquetes.

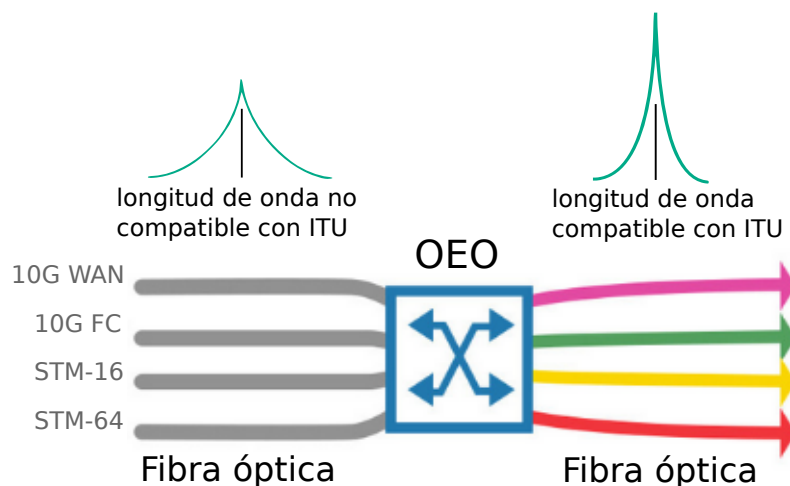
Un sistema de comunicaciones ópticas puede incluir diversos dispositivos, como ser:

- **Amplificadores ópticos.**
- **Switches ópticos**, encargados de conmutar de un canal a otro.
- **Divisores de luz**, cuya tarea comprende dividir la señal en diferentes caminos de fibra óptica.
- **Fibra óptica**, que cumple de medio de transporte de la información entre los diferentes equipos.
- **Transponders y Muxponders**, encargados de enviar y recibir las señales ópticas por las fibras. Generalmente son caracterizados por el ancho de banda que pueden transportar y la distancia que puede alcanzar la transmisión.

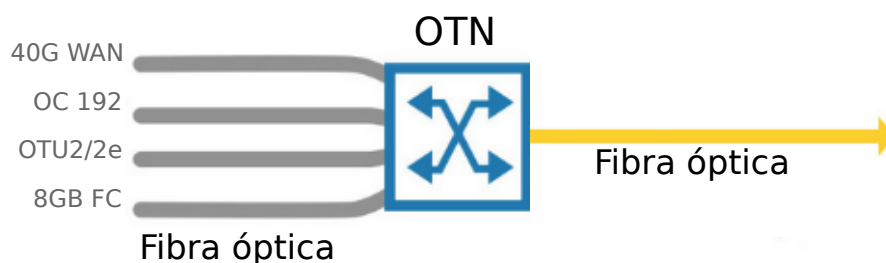
Transponders y Muxponders

El *transponder*, es un dispositivo que recibe múltiples señales ópticas a través de sus puertos clientes, dichas señales ópticas pueden tratarse de servicios diferentes como por ejemplo, *Ethernet*, *SONET*, *OTN*, entre otros. Luego, transforma estas señales en flujos de datos eléctricos, las procesa y regenera las mismas para nuevamente convertirlas en señales ópticas compatibles con el estándar *ITU*. De esta forma, realiza la función de recepción, amplificación y reemisión de una señal óptica en un proceso que comúnmente se denomina *optical electrical optical* (OEO) [15].

La figura 2.9 ejemplifica el proceso OEO típico de un *transponder*.

FIGURA 2.9: Funcionamiento básico de un *transponder*.

Por otra parte, los *muxponders* realizan una función similar a los *transponders*. También incluyen el proceso *OEO*, con la diferencia de que combinan múltiples servicios en una sola longitud de onda que luego se multiplexan en la misma fibra [15]. Por lo tanto, en lugar de asignar a cada servicio una longitud de onda dedicada, permite que varios servicios diferentes compartan la misma longitud de onda. Estos dispositivos maximizan la utilización de la fibra y ofrecen soluciones de bajo costo para empresas y transportistas. La figura 2.10 muestra el comportamiento de un *muxponder*.

FIGURA 2.10: Funcionamiento básico de un *muxponder*.

Aplicaciones

Resulta importante ahora separar la red en dos capas diferentes: la capa de paquetes o de *IP/MPLS*, y la capa óptica o de transporte [32]. La figura 2.11 muestra dicha separación. Los dispositivos mencionados anteriormente se utilizan en la capa de transporte mientras que los *routers* y *switches* convencionales se encuentran en la capa *IP/MPLS*.

La función que tienen los *muxponders* es la de proveer una conexión lógica entre los diferentes *routers*, quienes podrían estar separados por enormes distancias donde los protocolos como *Ethernet* no proveen un buen servicio de transporte.

De esta forma, los dispositivos de la capa *IP/MPLS* tienen conocimiento de sus vecinos pero no de la forma en la que se encuentran conectados ni de cómo se está realizando dicha comunicación, mientras que los equipos de la capa óptica esencialmente emparejan a los dispositivos de la capa *IP/MPLS*, pero sin tener conocimiento sobre los diferentes servicios que se prestan.

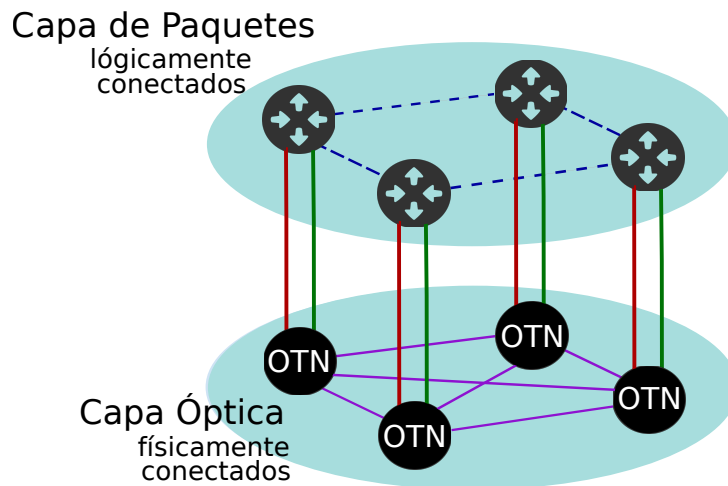


FIGURA 2.11: Separación de la red en capa de paquetes y capa de transporte.

Capítulo 3

Análisis de las tecnologías

Teniendo en cuenta los conceptos revisados en el capítulo anterior, en este se estudiarán las herramientas que permitirán la realización del proyecto.

En la primera sección, se realizará un análisis del dispositivo utilizado, un *muxponder* óptico coherente de 40Gb desarrollado por la institución donde se realizó el proyecto.

Luego, en la segunda sección se examinarán las herramientas de software involucradas. La misma se encuentra dividida en dos partes; la primera detalla el funcionamiento del controlador *SDN* utilizado, *ONOS*; la segunda refiere al estudio de dos agentes *NETCONF*: Sysrepo y Yuma123.

3.1. Herramientas de Hardware

Para cumplir con el objetivo del proyecto, será de suma importancia conocer las bondades y las limitaciones del equipo con el que se cuenta. Así, esta sección comprende el estudio de uno de los dispositivos mencionados en el capítulo anterior, un *muxponder*. Concretamente, se analizarán aspectos técnicos relacionados tanto al hardware como al software de un *muxponder* de 40Gb. El interés del análisis resulta en que es en este dispositivo en donde se integrará el protocolo de gestión *NETCONF*.

3.1.1. Muxponder 40Gb

El *muxponder* con el que se cuenta es capaz de realizar una transmisión óptica de 40Gb/s sobre una señal de línea *OTU3*. La misma es lograda cumpliendo el estándar *ITU-T G.709* [9], utilizando una modulación coherente *DP-QPSK* o *DP-DQPSK*. Dispone de cuatro clientes ópticos asíncronos totalmente independientes de 10Gb/s cada uno, a través de módulos ópticos *XFP* removibles. Las longitudes de ondas soportadas para los clientes son 850/1310/1550 nm y admite los tipos de cliente *10Gb Ethernet LAN/WAN*, *OTU2* y *OTU2e*.

Además, incorpora el mecanismo de corrección de errores *FEC* para todas las señales, tanto para clientes como para línea. Mediante el mismo, el *muxponder* es capaz de realizar correcciones sin necesidad de retransmitir la información.

En términos de potencia, alcanza típicamente los 93 Watts. También, incorpora un amplificador óptico, el cual le permite alcanzar una distancia de hasta 2000Km. Si no se utiliza dicho amplificador, puede alcanzar una distancia de hasta 65Km.

Las interfaces de conexión soportadas para realizar configuración y monitoreo en el dispositivo son:

- 2 puertos *Ethernet*.
- 1 puerto serial *RS232*.
- 1 puerto *USB 2.0*.

En la figura 3.1 se puede observar en el panel frontal del equipo con las diferentes interfaces mencionadas anteriormente.

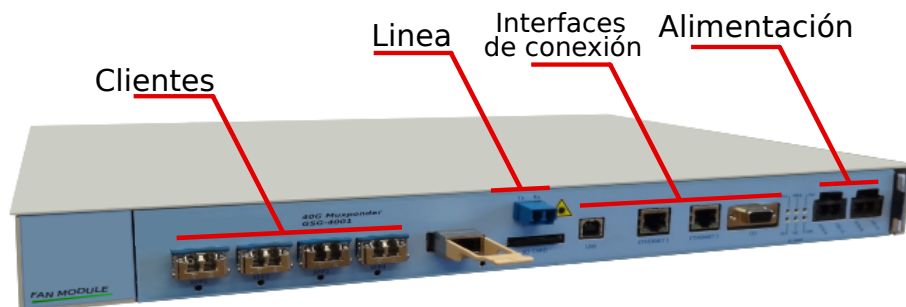


FIGURA 3.1: Vista del panel frontal del *muxponder* de 40Gb utilizado.

Por otra parte, el *muxponder* de 40Gb integra un total de 128Mb de memoria RAM y 512Mb de almacenamiento, con capacidad de extender esta última mediante una tarjeta SD. Además, cuenta con un sistema operativo Linux "*Buildroot*", el cual ocupa gran parte de estos recursos mencionados, dejando libre para las aplicaciones de usuario un total de 100Mb de RAM y 270Mb de almacenamiento.

El hecho de que presente dicho sistema operativo resulta en una ventaja por varios motivos, en primer lugar porque el mismo es un entorno conocido por el alumno, donde además podrán ejecutarse en él la mayoría de las aplicaciones UNIX típicas. En segundo lugar, el sistema operativo integra librerías y herramientas que facilitaran el desarrollo del proyecto, como por ejemplo la librería SSH, necesaria por el protocolo NETCONF.

Por último, el procesador que incorpora es un NIOS II de primera generación fabricado por Intel [18]. El mismo funciona a 125 Mhz y se encuentra integrado en una FPGA. Es importante destacar que la arquitectura de este procesador no es una arquitectura típica de una máquina de propósito general (por ejemplo x86_64), por lo tanto, las distintas aplicaciones que se ejecuten en esta plataforma deberán estar compiladas específicamente para la arquitectura NIOS.

Además, debido a las capacidades de la memoria primaria y secundaria del equipo, resulta imposible realizar la compilación de las aplicaciones sobre el mismo. Por lo tanto, se deberá realizar lo que se conoce como compilación cruzada, que consiste en preparar un sistema huésped (donde generalmente dicho sistema cuenta con mayores recursos y capacidades) para generar todos los binarios y librerías que requiere el dispositivo objetivo donde finalmente se ejecutarán las aplicaciones.

3.2. Herramientas de Software

Además del estudio del hardware utilizado, resulta de interés realizar un análisis de los componentes de software que conforman el proyecto. Para ello, la primer parte de esta sección estará dedicada a estudiar el controlador *SDN* empleado, mientras que en la segunda parte se analizarán dos agentes *NETCONF* disponibles de código abierto.

3.2.1. Controlador ONOS

El controlador *ONOS*, desarrollado y mantenido por la *ONF* [22], es uno de los controladores abiertos más comunes en la industria, donde destacan miembros como Google, Intel, AT&T, Samsung, entre una numerosa lista [24]. Está diseñado específicamente para los proveedores de servicios, donde sus principales objetivos son la escalabilidad y el alto rendimiento [11].

Las licencias compatibles con *ONOS* son *Apache 2.0*, *MIT* y *BSD* [12]. El hecho de que sea un proyecto *open-source*, supone ventajas como ser interoperabilidad, personalización, flexibilidad e independencia del fabricante.

Antes de detallar cómo funciona y realizar un análisis de su arquitectura, es importante explicar el problema que enfrentan los controladores *SDN* para poder entender las ventajas que supone el mismo.

Debido al crecimiento del consumo de tráfico en las redes y la demanda del ancho de banda en alza, es necesario para los proveedores de servicio que el rendimiento y la escalabilidad de sus redes no se vean afectadas por estos motivos. De este modo, los controladores *SDN* deben poseer tres atributos claves: escalabilidad, rendimiento y alta disponibilidad [21].

- **Escalabilidad:** como se explicó en el capítulo anterior, *SDN* introduce una autoridad de control centralizada. La misma, debe ser capaz de escalar de igual forma que las funcionalidades de la red, manteniendo su rendimiento.
- **Alta disponibilidad:** el plano de control que se encuentra centralizado en el controlador, juega ahora un papel crítico. Esto es así ya que si el mismo se encuentra sobrecargado o deja de estar disponible por alguna razón, la funcionalidad de la red se vería afectada. Por lo tanto, las diferentes soluciones *SDN* deberán brindar disponibilidad ininterrumpida del controlador.
- **Rendimiento:** el controlador también tiene que ser capaz de proveer mecanismos para adaptarse dinámicamente ante las fluctuaciones en la carga del tráfico y la congestión de la red, evitando que el rendimiento del mismo se vea afectado.

Arquitectura del controlador

Las características más importantes de la arquitectura presentada por *ONOS* [11] se detallan a continuación:

- **Núcleo distribuido:** la solución que propone *ONOS* para proveer escalabilidad, alto rendimiento y disponibilidad, se basa en un núcleo distribuido por

los diferentes nodos que conforman un *cluster*, lo que implica la posibilidad de soportar enormes cantidades de dispositivos de red. Esto último es así ya que ONOS permite la incorporación dinámica de nuevos nodos, con lo que la carga del controlador podría distribuirse entre ellos de forma adaptativa.

La figura 3.2 ejemplifica dicha distribución. El hecho de agregar esta redundancia implica una mayor disponibilidad del controlador. A su vez, permite realizar un balanceo de carga, lo que implica mayor rendimiento y escalabilidad.

- **Abstracción Northbound:** el plano aplicación, explicado en el capítulo anterior, se comunica con ONOS a través de una interfaz brindada por el controlador. El mismo, brinda a las aplicaciones gráficos y estadísticas de la red como así también aplicaciones basadas en intents para facilitar el control, administración y configuración de los equipos.
- **Abstracción Southbound:** de forma similar, el controlador ofrece una interfaz para comunicarse con el plano de datos. Cabe destacar que si bien ONOS basa su funcionamiento en el protocolo *OpenFlow*, también brinda soporte a otros como *NETCONF*, *REST*, *SNMP*, etc, con el fin de mantener compatibilidad con dispositivos más antiguos.

Una aproximación más detallada de la arquitectura que presenta ONOS puede verse en la figura 3.3. En la misma, se observan las interfaces mencionadas anteriormente junto a una serie de componentes que pertenecen a la interfaz *Southbound*. Estos componentes se analizarán más adelante.

- **Modularidad:** el controlador se encuentra desarrollado en *Java*, y mediante el *framework* OSGi, obtiene las características de una arquitectura modular. De esta forma, se provee a los desarrolladores facilidad para brindar actualizaciones a sus aplicaciones, poder monitorearlas, realizar depuración y mantenimiento.

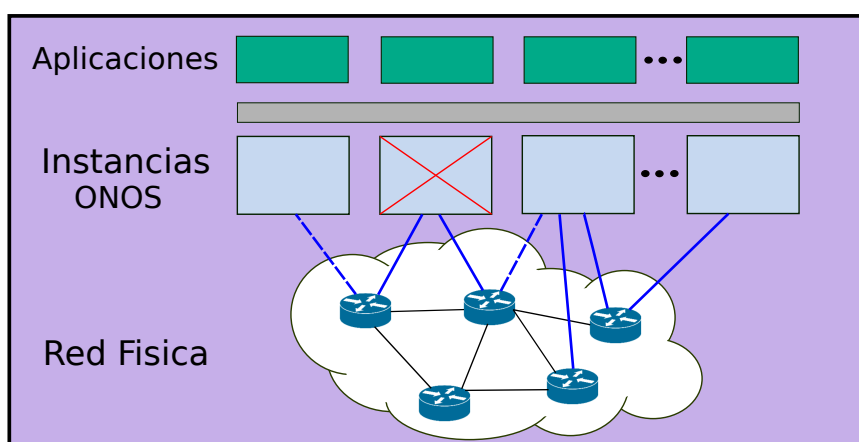


FIGURA 3.2: Arquitectura distribuida de ONOS.

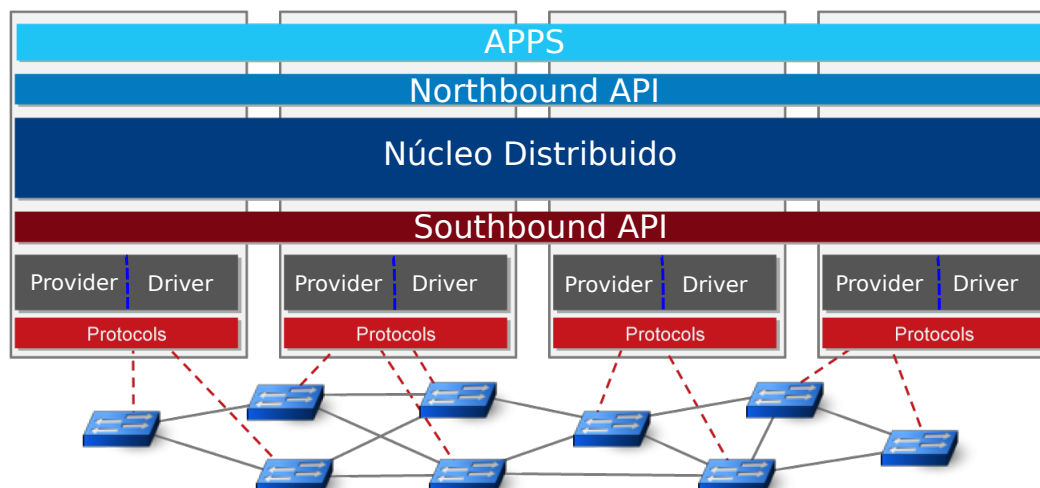


FIGURA 3.3: Arquitectura completa del controlador ONOS.

Interfaz Southbound en ONOS

Tal como se explicó anteriormente, el objetivo del proyecto es gestionar la configuración de un *muxponder* de 40Gb a través del protocolo *NETCONF*.

Para ello, se procede a explicar con más detalle la interfaz *Southbound* de ONOS. La misma, se encuentra dividida en una serie de componentes que se detallan a continuación:

- **Providers:** son aplicaciones independientes que residen en el núcleo de ONOS y que pueden activarse o desactivarse dinámicamente en tiempo de ejecución. El propósito principal de esta capa es abstraer al *core* las complejidades de los protocolos, brindando interfaces de las operaciones típicas y generales de los mismos. Un ejemplo de un *provider* en ONOS es el llamado “*NetconfAlarmProvider*”, encargado de transformar cada notificación de los dispositivos en una alarma registrada en ONOS.
- **Protocols:** es la capa de más bajo nivel en la interfaz *Southbound* y es la única que tiene contacto directo con los dispositivos conectados al controlador. Aquí se implementan los diferentes protocolos necesarios para la comunicación como ser *NETCONF*, *REST*, *SNMP*, etc. Comúnmente se utilizan librerías de terceros como *openflowj*, *snmp4j*, *thrift*, entre otras.
- **Drivers:** al igual que los *providers*, los *drivers* pueden cargarse dinámicamente al núcleo del controlador y proveen mecanismos para comunicarse con los diferentes dispositivos a través de algún protocolo. La diferencia principal con los *providers*, es que aquí no se implementan generalidades de los protocolos, sino comportamientos específicos de los dispositivos. Además, sirve de interfaz entre las aplicaciones que se encuentran en la capa *Northbound* y los diferentes equipos de red. El propósito principal de este subsistema es el de aislar el código específico del dispositivo, de tal manera de que el mismo no se extienda por el resto del núcleo de ONOS. Dado que dicho código será necesario para cualquier futuro previsible, este subsistema proporciona medios para contenerlo y permitir que otros subsistemas (por ejemplo, la capa de aplicación) interactúen con él a través de abstracciones independientes del protocolo

y del dispositivo. Por último, presenta una ventaja para los desarrolladores de hardware dado que al ser un componente modular, permite la herencia de funcionalidades de otros *drivers* con el fin de compartir características con una familia de dispositivos en común.

La figura 3.4 esclarece la participación que tiene cada componente tanto con el *core* como con el dispositivo.

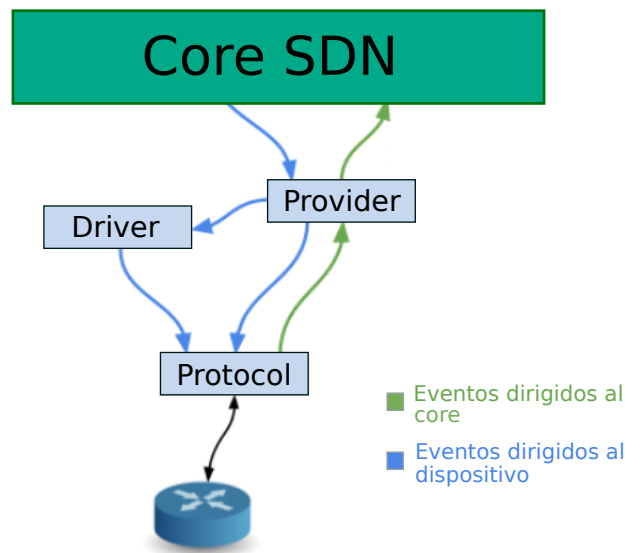


FIGURA 3.4: Interfaz *Southbound* en ONOS.

Justificación de la elección del controlador

En la actualidad, existe una diversidad de controladores *SDN*, como ser *Ryu* (Python), *Floodlight* (Java), *POX* (Python), e incluso implementaciones propietarias.

Se destaca *OpenDaylight* (Java), un controlador abierto que soporta una gran lista de protocolos y que, según [10], junto a *ONOS* es uno de los controladores más utilizados en la industria.

La razón determinante por la cual se optó por *ONOS* como controlador *SDN* radica en que el mismo cuenta con una documentación más clara y organizada. Además, se poseía experiencia previa trabajando con dicho controlador. Todo esto facilitó la curva de aprendizaje de las distintas herramientas, donde se pudo tener una rápida interacción con el controlador dada su facilidad de instalación y puesta en marcha.

Otro motivo reside en que las redes de los proveedores de servicio son complejas y multicapas, donde se requiere una separación clara de la capa de paquetes y de la capa de transporte, tal como se vió en el capítulo anterior. *ONOS*, ha logrado brindar soporte a las redes ópticas según lo demuestra el caso de uso aquí descripto [11].

3.2.2. Análisis de agentes *NETCONF*

Con el fin de poder gestionar la configuración del *muxponder* de 40Gb a través de *NETCONF*, se estudiará en esta sección dos implementaciones del protocolo: Sysrepo y Yuma123.

Las mismas son *open-source*, lo que facilita el estudio y comprensión de los agentes. Finalmente, se justificará la elección de Yuma123 como servidor *NETCONF* para el proyecto.

Sysrepo

El proyecto Sysrepo proporciona las funcionalidades de una base de datos lógica a las diferentes aplicaciones Unix-Linux. De esta forma, las aplicaciones pueden gestionar sus datos de configuración y de estado utilizando *YANG* como modelado de datos, a través de las *API's* e interfaces que expone Sysrepo [33]. Así, la implementación garantiza mediante *YANG* la consistencia de los datos y la correctitud de los mismos.

A su vez, Sysrepo integra *Netopeer2* [5] como agente *NETCONF*. *Netopeer2* es la evolución del proyecto *Netopeer* [4] (discontinuado) y ofrece tanto un cliente como un servidor *NETCONF*.

Sysrepo fue la primer implementación del protocolo instalada y manipulada en una máquina de propósito general. Tiene la ventaja de contar con una gran documentación, como así también una variedad de ejemplos y casos de usos. Además, otra ventaja que presenta es que el hecho de que Sysrepo exponga *API's* implica una posibilidad de adaptar cualquier aplicación Unix existente al protocolo *NETCONF*, sin mayores cambios.

Yuma123

En el 2011, el proyecto *open-source* YUMA, también conocido como OpenYUMA, sufrió un cambio en su licencia donde esta dejó de ser *BSD*. A partir de entonces, el proyecto tuvo dos ramificaciones: YumaPro [40], ahora perteneciente a YumaWorks, y Yuma123, su versión *open-source*.

Yuma123 nace a partir de la última *release BSD* del proyecto OpenYUMA, con el fin de continuar con el soporte de dicha implementación mientras se mantiene la licencia *BSD*. Al igual que Sysrepo, ofrece tanto un cliente (*yangcli*) como un servidor (*netconfd*) *NETCONF*. La diferencia con la implementación anterior es que aquí no se exponen *API's* a las aplicaciones, sino que las mismas son directamente compiladas como librerías *SIL* y son dependientes de Yuma123.

Según la documentación [37], se agregaron las siguientes funcionalidades con respecto a la versión original de OpenYUMA:

- Un sistema de compilación más eficiente, basado en las herramientas *autoconf* y *automake*.
- Se han corregidos *bugs* críticos reportados en OpenYUMA.
- Soporte de las nuevas funcionalidades del protocolo agregadas por la *IETF* (*ietf-nacm*, *ietf-system*, etc.).

Evaluación de las implementaciones

A la hora de efectuar una comparación entre ambos proyectos, se tendrán en cuenta los siguientes criterios: las diferencias relativas al protocolo *NETCONF*; las

herramientas y características extras que brinda cada una; y los recursos que demandan.

- **Diferencias relativas al protocolo *NETCONF*:** Como se detalló en el capítulo anterior, *NETCONF* define una serie de operaciones que no son obligatorias para las diferentes implementaciones del protocolo, sino que son opcionales y las mismas deberán ser explícitamente anunciadas en el mensaje *HELLO* del servidor. Es importante repasar cuáles de estas operaciones admite cada proyecto.

Tanto Yuma123 [38] cómo Sysrepo [33] implementan el estándar *NETCONF* 1.0 y *NETCONF* 1.1, definidos en los *RFC 4741* [16] y *RFC 6241* [17] respectivamente.

Sin embargo, mientras que Sysrepo admite el transporte seguro mediante *SSH* y *TLS*, Yuma123 únicamente soporta *SSH*. Esto último, es una ventaja para Sysrepo ya que brinda flexibilidad y personalización al administrador sobre el protocolo de transporte seguro.

Por otra parte, Sysrepo admite únicamente la operación *commit* sobre la base de datos *candidate*, mientras que Yuma123 además de soportar dicha operación también incorpora las capacidades *confirmed-commit* y *validate*, lo que provee a esta última de potentes herramientas para corroborar la correctitud de los datos ingresados y a su vez restaurar la funcionalidad de la red en caso de ingresar una configuración incorrecta.

Para finalizar, cabe destacar que ambos proyectos soportan las bases de datos *startup* y *candidate*.

- **Herramientas y características extras al protocolo:** Ambas implementaciones integran tanto un cliente como un servidor *NETCONF*. Sin embargo, cada una incorpora una serie de herramientas que resulta de importancia mencionarlas.

- **Sysrepo**

- *sysrepoctl*: aplicación que permite administrar los módulos *YANG* desde una *CLI*. Brinda opciones para instalar, eliminar y listar los módulos que tiene activo el servidor.
- *sysrepocfg*: utilidad para exportar o importar datos de configuración de las diferentes bases de datos. De esta forma se podría editar, por ejemplo, el contenido de la base de datos *startup* desde un navegador web o editor de texto cualquiera, sin que sea necesario utilizar el protocolo *NETCONF* para dicho propósito.

- **Yuma123**

- *yangdiff*: herramienta que permite comparar dos revisiones de un mismo módulo *YANG*. El nivel de detalle con el cual se exponen las diferencias puede ajustarse hasta con tres niveles de reporte. Además, puede generar de forma automática la declaración “*revision*” del módulo con detalles de los cambios.
- *yangdump*: posibilita validar módulos *YANG* y convertirlos a otros formatos. De esta forma, mediante un módulo *YANG* la herramienta genera el esqueleto del código *SIL* (lenguaje C) que necesita para

relacionar la instrumentación del dispositivo con el modelado de los datos.

Para finalizar el análisis de este criterio, se menciona que ambas implementaciones permiten parametrizar opciones en el servidor *NETCONF*, como por ejemplo el número máximo de sesiones admitidas, el tiempo de espera para una respuesta *RPC* y el tiempo de espera de una sesión inactiva antes de finalizarla. Además, anteriormente se mencionó que mientras Sysrepo expone *API's* a las diferentes aplicaciones Unix, Yuma123 las integra como librerías *SIL* dependientes de la implementación. Esto último es una ventaja para Sysrepo, ya que tanto Sysrepo como la aplicación funcionarían como procesos diferentes que se comunican mediante interfaces, donde la falla de uno de estos procesos no necesariamente involucra el bloqueo completo del otro. Esto último no sucede en el caso de Yuma123, donde es el servidor quien realiza las llamadas a las librerías *SIL* previamente compiladas, formando un solo proceso.

- **Demanda de recursos:** Al inicio de este capítulo, se estudiaron las características técnicas del *muxponder* utilizado para este proyecto. Será de suma importancia que las implementaciones mencionadas se adapten a los recursos que dispone el equipo, por lo que se hará foco principal en demanda de la memoria *RAM* y de la memoria de almacenamiento.

Dicho esto, es importante mencionar que para el siguiente análisis se iniciaron los binarios con la configuración por defecto. Además, los datos obtenidos corresponden a la ejecución de los mismos en una máquina de escritorio, sin realizar algún tipo de optimización en recursos.

- **Sysrepo:** según la documentación [34], se requiere de una extensa lista de librerías de terceros para poder efectuar la compilación e instalación del proyecto. Teniendo en cuenta dichas librerías necesarias para el funcionamiento de Sysrepo, la implementación demanda un espacio total en memoria secundaria de 250Mb. Cabe destacar que en este análisis se incluye no solo el servidor Netopeer2 sino también el cliente, ya que Sysrepo necesita de ambos para funcionar. En el caso de memoria *RAM*, Sysrepo ocupa 270Mb.
- **Yuma123:** en este caso, la cantidad de librerías de terceros que requiere el proyecto [37] es menor. Además, se destaca que Yuma123 no necesita de ambos binarios (cliente y servidor) para funcionar, pudiendo iniciarse uno u otro según sea necesario. Teniendo en cuenta esto último, únicamente se analizan los recursos que demanda el servidor (llamado *netconfd*), ya que en el dispositivo no será necesario ejecutar un cliente *NETCONF*. Así, Yuma123 requiere en memoria secundaria un espacio de 50Mb, mientras que en memoria principal alcanza los 73Mb aproximadamente.

En figura 3.5, en la primer columna, puede verse una comparativa de la memoria *RAM* que demanda cada implementación, la expresión está dada en el orden de los *Kb*. El proceso "*netconfd*" corresponde al servidor *NETCONF* del proyecto Yuma123, mientras que el proceso "*sysrepor*" corresponde a Sysrepo e integra tanto el cliente como el servidor.

VIRT	RES	SHR	S	%CPU	%MEM	HORA+	ORDEN
72108	7856	6084	S	0,0	0,0	0:00.04	netconfd
263260	4484	3820	S	0,0	0,0	0:00.01	sysrepo

FIGURA 3.5: Demanda de recursos de las implementaciones analizadas.

Justificación de elección del agente

Presentado el análisis y las diferencias entre ambos proyectos, en esta sección se justificará la elección de Yuma123 como servidor que se instalará en el *muxponder* de 40Gb.

Como se mencionó anteriormente, Sysrepo fue la primer implementación con la que se tuvo contacto y manipulación del protocolo *NETCONF*. La razón por la que se optó empezar a familiarizarse con este, fue porque se encontró una gran cantidad de ejemplos y casos de uso a la hora de realizar los módulos *YANG* y relacionarlos con la instrumentación y las aplicaciones Unix. Además, la instalación del proyecto en una computadora de escritorio fue sencilla (debido a la extensa documentación y las diferentes alternativas de instalación que brinda como ser *dockers*, *scripts* de instalación, etc).

Sin embargo, no resultó de igual forma a la hora de realizar la compilación cruzada. La razón se debe a que Sysrepo tiene gran cantidad de dependencias como ser *libyang*, *Google Protocol Buffers*, *protobuf-c*, *libev*, entre otros. Específicamente, se tuvo problemas para compilar la librería “*protobuf-c*” para la arquitectura *NIOS*, por lo que se abandonó el uso de esta herramienta. Además, como se vio anteriormente, la demanda de memoria principal y secundaria en Sysrepo excede a los recursos disponibles en el *muxponder*.

En el caso de Yuma123 se logró compilar e instalar de manera correcta todas las librerías requeridas. Además, se realizaron scripts que facilitan dicha tarea para las siguientes arquitecturas: *ARM*, *NIOS* y *x86_64*. Cabe destacar que si bien los recursos que demanda Yuma123 son menores frente a Sysrepo, los mismos siguen siendo excesivos para el *muxponder*. Por lo tanto, se realizaron optimizaciones en la compilación del proyecto. Por ejemplo, se ha omitido la compilación de la librería *SSH*, ya que el *muxponder* ya la integra. Además, el proyecto incorpora una gran cantidad de módulos *YANG* a modo de ejemplo, estos no son necesarios para el funcionamiento del mismo, por lo que también fueron omitidos. Por último, se destaca la herramienta *yangdump* brindada por Yuma123, la cual facilita de forma significativa el desarrollo de las librerías *SIL* en C.

De esta forma, el factor determinante a la hora de elegir entre las distintas implementaciones *NETCONF*, fue tener en cuenta las limitaciones técnicas del equipo, siendo Yuma123 el agente que mejor se adaptó a las mismas.

Capítulo 4

Diseño e Implementación

Para el desarrollo del proyecto, será necesario implementar un sistema donde se puedan realizar pruebas y evaluar de este modo la correctitud del mismo.

De esta forma, la primera sección de este capítulo comprende tanto el análisis del sistema como la topología utilizada. Específicamente, se estudiará su estructura, su comportamiento y sus requerimientos.

En las secciones siguientes, se detallarán las diferentes aplicaciones desarrolladas que permiten cumplir con el objetivo del proyecto. En primer lugar, se hará un estudio del módulo YANG implementado, el cual caracteriza y modela los datos del equipo. A continuación, se detallará el driver implementado para lograr la comunicación entre el controlador y el dispositivo. Por último, se analizará la interfaz REST y la GUI desarrollada.

4.1. Entorno de trabajo

En esta sección se detallan las características del sistema a implementar, descritas en el lenguaje de especificación de sistemas SysML [20]. Se optó por este lenguaje de modelado ya que brinda una extensión a UML permitiendo combinar elementos del mundo físico (*hardware*) con elementos del mundo lógico (*software*).

4.1.1. Topología

Con el fin de poder identificar en una primera instancia los requerimientos que debe cumplir el sistema, se conformará una topología. La misma, está basado en un caso de uso simple, en el cual se quiere brindar conectividad a dos clientes a través de un enlace óptico, este último conformado por dos *muxponders*.

Además, es importante identificar la presencia del controlador, el cual gestionará la configuración de los equipos mediante el protocolo *NETCONF*. En la figura 4.1 se muestra de forma general la topología planteada.

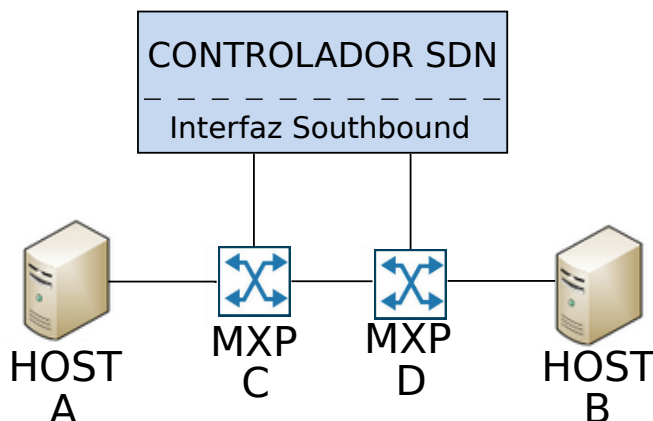


FIGURA 4.1: Topología implementada en el proyecto.

Los dispositivos terminales A y B, se encuentran conectados cada uno a los puertos cliente del *muxponder* de 40Gb. A su vez, el transmisor del *muxponder* C se encuentra conectado con el receptor del *muxponder* D, y viceversa. Esto permite una comunicación bidireccional, donde ambos clientes pueden tener conectividad.

A continuación, en la figura 4.2 se muestra como está dispuesta la conexión físicamente en los dispositivos. A la izquierda se tiene el *muxponder* C, y a la derecha se encuentra el *muxponder* D. Las interfaces de control de ambos dispositivos están conectadas al controlador ONOS, mientras que los puertos clientes se encuentran conectados a los clientes A y B respectivamente. Por otra parte, las interfaces de línea se encuentran conectados entre los *muxponder*, como se explicó anteriormente, con el fin de poder tener una comunicación bidireccional.

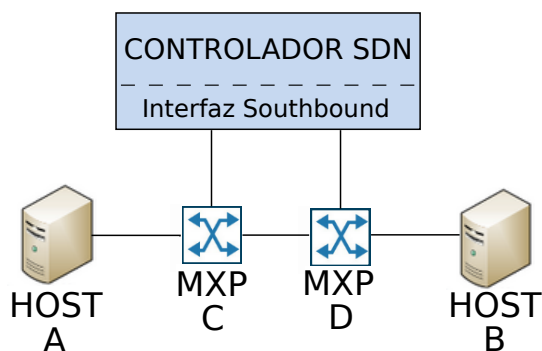


FIGURA 4.2: Conexión física de la topología.

4.1.2. Requerimientos del sistema

Para establecer los requerimientos funcionales del sistema, se presentará en primer lugar, diagramas de casos de usos. Como se puede observar en el diagrama de la figura 4.3, el objeto principal del sistema es poder brindar al administrador un entorno donde el mismo pueda gestionar a los *muxponders* mediante sus aplicaciones.

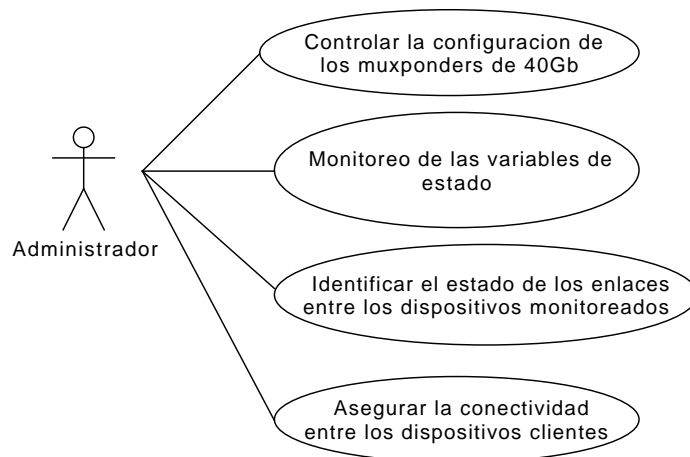


FIGURA 4.3: Caso de uso desde la perspectiva del administrador.

En base a esto, se pueden definir los requerimientos funcionales a nivel de sistema. La figura 4.18 muestra una lista de los mismos.

Como requerimiento no funcional, se puede identificar la adición de nodos virtuales, los cuales no afectan a la funcionalidad del sistema pero brindan más flexibilidad y permiten conformar una topología más compleja. Dichos nodos consistirán en switches virtuales conectados a los muxponders, y a su vez conectados con host virtuales.

<p>«Requerimiento» Controlador SDN</p> <p>id: R-01 El sistema debe brindar el servicio de un controlador SDN</p>	<p>«Requerimiento» Conexión con el controlador</p> <p>id: R-02 Se debe establecer una conexión entre los muxponders y el controlador</p>
<p>«Requerimiento» Servidor web</p> <p>id: R-03 El sistema debe proveer el servicio de una interfaz web para administrar los dispositivos.</p>	<p>«Requerimiento» Soporte NETCONF</p> <p>id: R-04 Los dispositivos y el controlador deben soportar el protocolo NETCONF.</p>
<p>«Requerimiento» Nodos físicos</p> <p>id: R-05 El sistema debe soportar la conexión de nodos físicos.</p>	<p>«Requerimiento» Nodos virtuales</p> <p>id: R-06 El sistema debe soportar la conexión de nodos virtuales.</p>

FIGURA 4.4: Requerimientos del sistema.

Además, a partir del diagrama de caso de uso de la figura 4.3, se pueden identificar tres diagramas de actividades relacionadas.

La figura 4.5, muestra el flujo de actividad típico que tendrá una operación de configuración por parte del administrador. Como se puede ver, el evento inicial es la adición de una configuración desde la aplicación web. Esta configuración, una vez procesada, se traduce en una solicitud POST HTTP que se envían hacia la Northbound interface del controlador. Una vez recibida la solicitud, se realiza su procesamiento y se envía el mensaje de configuración NETCONF a los dispositivos involucrados a través de la Southbound interface.

Por otra parte, la actividad de monitoreo se observa en la figura 4.5, donde la aplicación realiza consultas periódicas al controlador para obtener información sobre los dispositivos. Se envía un mensaje HTTP (esta vez una operación GET), el controlador procesa la petición y responde nuevamente la consulta.

Por último, en la figura 4.5 se tiene el flujo de actividades donde el controlador registra una alarma mediante una notificación NETCONF que envía un dispositivo.

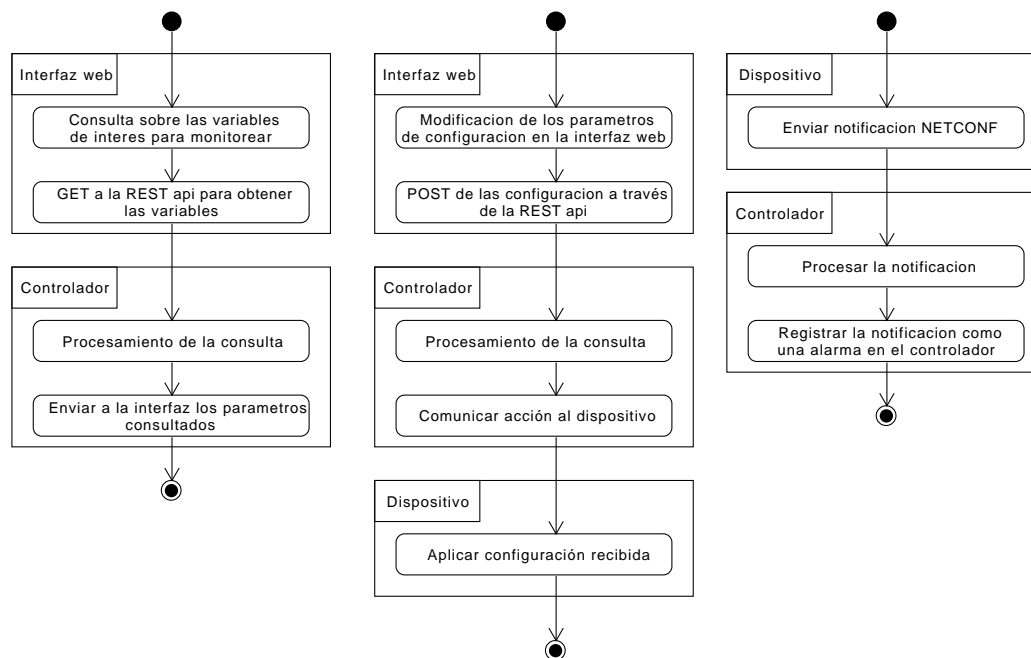


FIGURA 4.5: Topología implementada en el proyecto.

4.2. Integración del protocolo NETCONF al muxponder

La sección anterior, deja claro que uno de los requerimientos de sistema será que tanto el controlador ONOS como los dispositivos a monitorear, soporten NETCONF como protocolo de gestión de la configuración.

Como se vio en el capítulo anterior, el controlador soporta dicho protocolo, por lo que para cumplir con este requisito de sistema, únicamente será necesario adaptar NETCONF al dispositivo.

4.2.1. Requerimientos

Para cumplir con los requerimientos vistos en 4.18, se confeccionaron los requerimientos que se observan en la figura 4.6.

<p>«Requerimiento» Adaptar NETCONF al equipo</p> <p>id: R-07 Se debe instalar algún agente del protocolo en el dispositivo.</p>	<p>«Requerimiento» Monitoreo de datos de estado</p> <p>id: R-08 Se debe poder monitorear cualquier variable de memoria compartida del dispositivo a través de NETCONF.</p>
<p>«Requerimiento» Alarmas mediante notificaciones</p> <p>id: R-09 Las alarmas emitidas por el dispositivo deberán notificarse mediante el protocolo NETCONF.</p>	<p>«Requerimiento» Gestionar datos de configuración</p> <p>id: R-10 Se debe poder realizar cambios en la configuración del equipo mediante el protocolo NETCONF.</p>

FIGURA 4.6: Topología implementada en el proyecto.

4.2.2. Compilación instalación del agente

Siendo YUMA123 el agente que se eligió para integrar el protocolo NETCONF en el equipo, en esta sección se detalla el procedimiento que se llevó a cabo para la compilación e instalación del agente en el muxponder, con el fin de poder cumplir el requerimiento R-07.

En primer lugar, toda la tarea de compilación se realizó en una computadora de propósito general, debido a los recursos limitados con los que cuenta el dispositivo. Para facilitar la compilación e instalación, se realizaron tres scripts los cuales se describen a continuación:

- **Dockerfile:** el objetivo de este script, es el de realizar la compilación del proyecto YUMA123, dejando todas las librerías y los binarios en una carpeta que luego deberá copiarse en el dispositivo. Dockerfile [] es un archivo de texto que contiene los pasos e instrucciones que la herramienta Docker deberá seguir para construir una imagen. En el mismo, se indica una imagen de referencia (ubuntu:16.04) que servirá como base para la construcción. Luego, se descargan todas las librerías requeridas por el proyecto YUMA123 junto con los compiladores necesarios para realizar la compilación cruzada. A partir de aquí, se compila cada una de las librerías para la arquitectura objetivo (por ejemplo, NIOS II) y finalmente se compila también el proyecto YUMA123. Todos los binarios, librerías y cabeceras resultantes se encuentran en la carpeta `"/root/usrapp"` de la imagen Docker construida. Cabe destacar que de esta forma se realizaron tres versiones del script, con el objetivo de poder compilar para las arquitecturas NIOS II, ARM y 86 64.
- **remote_install_yuma.sh:** script bash que tiene la tarea de realizar la instalación del protocolo compilado previamente con Docker y Dockerfile. Para ello, requiere de tres parámetros: usuario del dispositivo remoto, dirección IP del dispositivo remoto y la arquitectura deseada. Con estos parámetros, el script realiza la instalación del agente mediante SSH y SCP [], copiando el contenido necesario de `"/root/usrapp"` que se generó con la construcción de la imagen Docker, al directorio `"/root/usrapp"` del dispositivo. Es importante mencionar que muchas

de las librerías requeridas por YUMA123 son necesarias únicamente para la compilación y no para el funcionamiento del protocolo, por lo que este script omitirá dichas librerías con el fin de reducir el tamaño que ocupa el agente en memoria.

- *remote_ninstall_{yuma}.sh*: requiere dos parametros los cuales son el usuario del dispositivo remoto y la dirección ip del mismo. El objetivo de este script es facilitar la desinstalación de todas las librerías relacionadas a YUMA123 del dispositivo.

4.2.3. Diseño del módulo YANG

Para poder cumplir con los requerimientos R-08, R-09 y R-10 que se muestra en la figura 4.6, se diseñó un módulo YANG que contiene cinco secciones bien definidas, las cuales se describen a continuación:

- **Cabecera del módulo y declaraciones:** contiene la estructura inicial de un módulo YANG. En él, se define un nombre y un prefijo, se realiza una descripción del mismo y por último se realiza la definición de los datos utilizados por el módulo. Cabe destacar que se definieron tres tipos de datos: *restricted-tipo-trafico*, *restricted-tipo-fec-linea* y *restricted-tipo-fec-cliente*, donde se especifica a través de la directiva *'enum'*, cuales son los valores aceptados que puede tomar dicho tipo de dato. Por ejemplo, dado que la configuración del tipo de tráfico para este dispositivo únicamente admite dos valores (*otu2* y *xge*), es importante restringir el ingreso de algún otro valor ya que podría ocasionar errores en la configuración. Se puede observar un fragmento del módulo realizado en la figura 4.1, donde se detalla la cabecera del módulo y sus declaraciones.

LISTING 4.1: Cabecera del módulo YANG.

```
module cli-mxp {

    namespace "http://fulgor.com/ns/cli-mxp";
    prefix "cli-mxp";

    description
        "CLI para configurar el muxponder de 40G";

    revision "2018-06-24" {
        description
            "Version 0.1.0";
    }

    typedef restricted-tipo-trafico {
        type enumeration {
            enum "otu2";
            enum "xge";
        }
    }

    ...
    ...
    ...
}
```

- **Container YANG de configuración:** el módulo contiene también una sección donde se declara un container llamado 'mux-config', el cual es el único que admite datos de configuración del dispositivo. En él, se describen todos los parámetros que admiten una configuración (por ejemplo, el tipo de fec de línea), a través de las declaraciones 'leaf'. Un fragmento de esta sección puede observarse en la figura 4.2, donde además se ve el uso de los tipos de datos definidos en la cabecera del módulo.

LISTING 4.2: Container de configuración.

```
container mux-config {  
    description "Parametros de la CLI";  
  
    leaf tipo_trafico {  
        description  
            "[otu2|xge] especifica el tipo de tráfico.";  
        type restricted-tipo-trafico;  
    }  
  
    ...  
    ...  
    ...  
  
    list ports {  
        key "port";  
        leaf port {  
            type int16{  
                range "0 .. 6";  
            }  
            mandatory true;  
        }  
    }  
  
    leaf neighbor {  
        mandatory true;  
        type string;  
    }  
  
    leaf port_neighbor {  
        mandatory true;  
        type string;  
    }  
}
```

- **Container YANG de estado:** de igual forma, se realizaron containers para los datos de estado, los cuales no admiten una escritura de valores y son necesarios para monitoreo del dispositivo. La figura 4.3 muestra una parte de esta sección del módulo, donde puede apreciarse la directiva "config false", la cual indica que el container no admitirá datos de configuración.

LISTING 4.3: Container de estado.

```

container mux-state {
    description "Representa a datos de estado del dispositivo.";

    config false;

    leaf fpga_temperature_state {
        description "Temperatura de la FPGA";
        type decimal64 {
            fraction-digits 2;
        }
    }

    leaf device_boardId {
        description "Identificador unico del dispositivo";
        type string;
    }
    ...
    ...
    ...

```

- **Definición de RPC:** Como se estudió en capítulos anteriores, NETCONF permite definir RPC propias de un módulo, con el fin de extender la funcionalidad de los dispositivos. Se define así una RPC cuya utilidad será la de poder indicar al agente, cuándo debe aplicar la configuración que contiene el container “mux-config” en el dispositivo. En la figura 4.4, se muestra dicha sección del módulo. En ella, se puede notar que la RPC admite una respuesta de la operación solicitada, la cual está contenida en el leaf “respuesta-mux-apply-config” y es de tipo String. En este mensaje se indica el resultado de la operación.

LISTING 4.4: Declaración de RPC.

```

rpc mux-apply-config {
    description "RPC que aplica los cambios de configuracion";
    output {
        leaf respuesta-mux-apply-config {
            type string;
        }
    }
}

```

- **Definición de notificación:** por último, el módulo tiene una sección donde se declara una notificación. Dicho mensaje será utilizado para indicar a las sesiones conectadas y suscritas, las diferentes alarmas que produzca el dispositivo. Estos mensajes se transportan mediante notificaciones del protocolo NETCONF. La idea de estos mensajes es la de, por ejemplo, poder notificar mediante una alarma si un enlace con un dispositivo vecino se cayó, si el dispositivo supera la temperatura umbral, etc. La figura 4.5 muestra la declaración de dicha notificación en el módulo, donde se puede ver que el mensaje estará contenido dentro de la leaf “INFO”, en la cual se especifica de forma obligatoria con la directiva “mandatory”, cuál será el mensaje que se enviará como notificación.

tipo de tráfico otu2 a través de la CLI. Se ejemplifica lo mencionado anteriormente en la figura [X].

<p>«Requerimiento» Controlador SDN</p> <p>id: R-01 El sistema debe brindar el servicio de un controlador SDN</p>	<p>«Requerimiento» Conexión con el controlador</p> <p>id: R-02 Se debe establecer una conexión entre los muxponders y el controlador</p>
<p>«Requerimiento» Servidor web</p> <p>id: R-03 El sistema debe proveer el servicio de una interfaz web para administrar los dispositivos.</p>	<p>«Requerimiento» Soporte NETCONF</p> <p>id: R-04 Los dispositivos y el controlador deben soportar el protocolo NETCONF.</p>
<p>«Requerimiento» Nodos físicos</p> <p>id: R-05 El sistema debe soportar la conexión de nodos físicos.</p>	<p>«Requerimiento» Nodos virtuales</p> <p>id: R-06 El sistema debe soportar la conexión de nodos virtuales.</p>

FIGURA 4.8: Topología implementada en el proyecto.

Teniendo en cuenta las aplicaciones mencionadas, se procede a explicar el diseño de la librería en el lenguaje C. Para ello, se utilizó la herramienta yangdump del proyecto YUMA123, la cual genera un esqueleto de la aplicación a partir de un módulo YANG dado. Se forman así dos archivos, uno con extensión ".h"(headers) y otro con extensión ".c"(código fuente).

En el primero, la herramienta declara todas las variables, funciones y los tipos de datos que va a utilizar la librería. Este archivo es incluido en el archivo con extensión ".c" mediante la directiva "include" en C.

El segundo archivo contiene la estructura de la aplicación en sí. En ella, se encuentran implementadas todas las funciones, las cuales llamará el agente en caso de que ingrese un mensaje referido al módulo YANG en cuestión. Todo el desarrollo de la aplicación y la relación entre la instrumentación del dispositivo con el módulo YANG, se encuentra en este archivo.

Con el fin de explicar cómo se desarrolló esta aplicación, se distinguen dos flujos de actividades bien definidos, uno para las operaciones que son sincrónicas con los mensajes que envía el cliente, y otro para aquellas que sean asíncronas a los mensajes del mismo.

Para el primer grupo, se tiene entonces las operaciones como obtención de un dato de estado o de configuración, modificación de un dato de configuración y ejecución de RPC, mientras que el segundo grupo contempla el envío de notificaciones, las cuales son asíncronas a las operaciones del cliente.

Se muestra el comportamiento de este primer grupo, en el diagrama de actividad de la figura 4.9. Cada vez que llega un mensaje NETCONF al agente YUMA123, el mismo procesa y verifica a qué módulo YANG hace referencia el mensaje y qué tipo de operación requiere el cliente.

Si la operación es una consulta por una variable de estado o de configuración, el agente realiza una llamada a una función relacionada a la variable consultada. En dicha función, lo que se hace es tomar el valor de memoria compartida del dispositivo, castear el mismo según indique el módulo YANG (string, int, uint, etc) y por último, emitir una respuesta al cliente con el valor del dato consultado.

Por otra parte, si la operación es la de editar una variable de configuración, el agente llama a una función de la librería C relacionada al módulo en cuestión, donde se actualiza el valor de dicha variable. Además, el agente emite un mensaje con el resultado de la operación.

Por último, si la operación trata de una RPC definida en el módulo YANG, el agente llama a la función relacionada a la RPC para efectuar la tarea solicitada. En este caso, se tiene una RPC que indica cuándo se deberá aplicar la configuración en el dispositivo, por lo que al momento de llamar a esta RPC, el agente copia los valores de los datos de configuración necesarios (tipo de tráfico, tipo fec de cliente, tipo fec línea, etc) y los aplica haciendo uso del binario ññññññ explicado anteriormente.

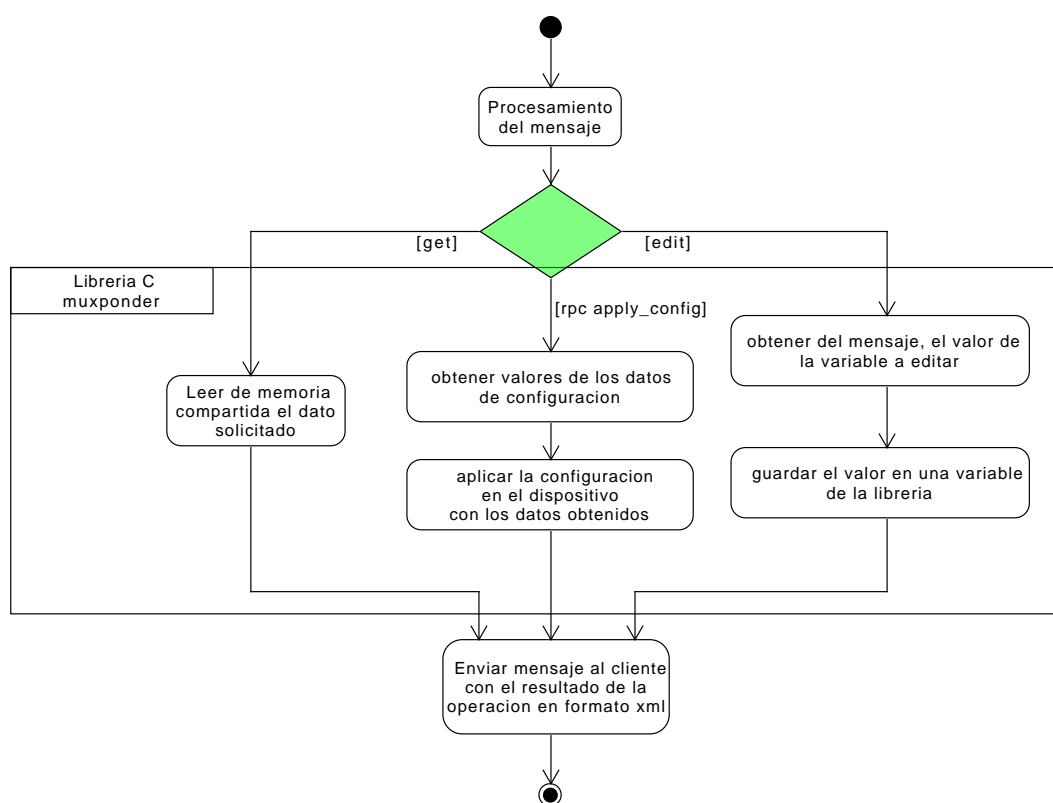


FIGURA 4.9: Diagrama de actividad de las operaciones sincronas con el cliente.

Por otra parte, el grupo relacionado a las operaciones asíncronas con los mensajes del cliente, no es otra cosa que la operación de envío de notificaciones mediante

el protocolo NETCONF. Para ello, la librería desarrollada en C, crea un hilo que examina periódicamente cada tres segundos la memoria compartida del dispositivo.

La razón por la cual se examina cada tres segundos, está relacionada con la aplicación “monitor”, la cual actualiza los valores de memoria compartida con esa frecuencia.

Al examinar los valores de las alarmas, las compara con la información antigua que se tenía almacenada sobre las mismas. Si la información es igual, no se envían notificaciones a las sesiones. En cambio, si la información actual es diferente a la información anterior, quiere decir que existe un cambio de estado de la misma y por lo tanto será necesario notificar a las sesiones suscritas este nuevo estado de las alarmas. Así, tanto el nombre de la alarma como su nuevo estado, son enviados a través de la notificación definida en la figura 4.5, haciendo uso de la leaf INFO.

Un diagrama de actividad de estas operaciones se puede observar en la figura 4.10.

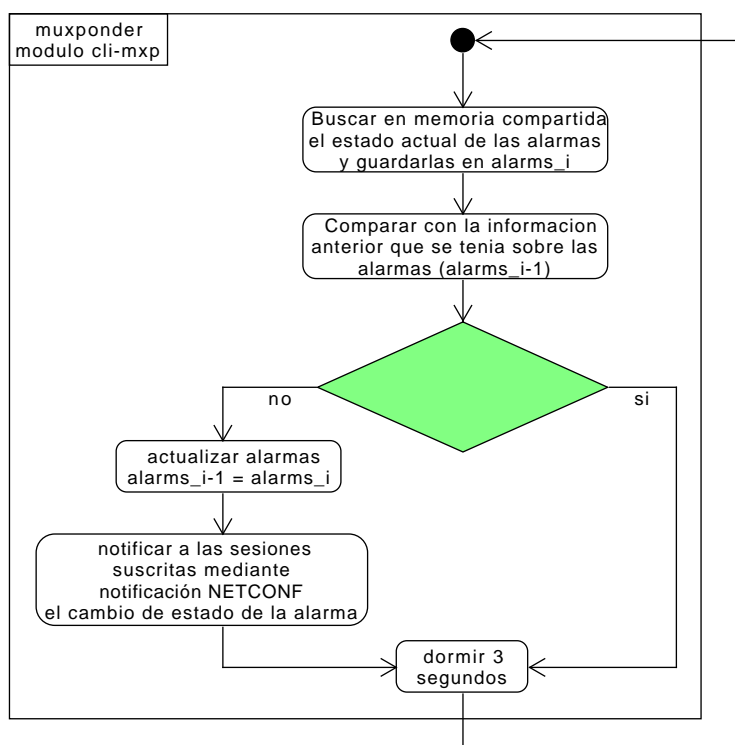


FIGURA 4.10: Diagrama de actividad de las notificaciones.

4.3. Diseño del Driver

Como se vio en capítulos anteriores, ONOS se comunica con los dispositivos a través de tres componentes de la interfaz Southbound: Providers, Protocols y Drivers.

Así, para poder indicar al controlador cuáles serán las operaciones y los comportamientos específicos del muxponder de 40Gb, será necesario desarrollar un Driver (Java) en la interfaz Southbound del controlador.

4.3.1. Requerimientos

A fin de cubrir las necesidades del administrador, visto en el caso de uso de la figura 4.3, el driver desarrollado deberá cumplir con los requerimientos funcionales del a figura 4.11.

<p>«Requerimiento» Descubrimiento del dispositivo</p> <p>id: R-11 El driver desarrollado debera ser capaz de identificar las características básicas del equipo en el momento de la conexión</p>	<p>«Requerimiento» Descubrimiento de enlaces</p> <p>id: R-12 El driver debe proveer mecanismos e indicaciones al controlador sobre como estan conformado los enlaces con los dispositivos vecinos</p>
<p>«Requerimiento» Soporte a las RPC</p> <p>id: R-13 El driver debera brindar soporte a todas las RPC definidas en el modulo YANG del dispositivo</p>	<p>«Requerimiento» Comandos CLI de ONOS</p> <p>id: R-14 El driver debe exponer comandos CLI a las operaciones típicas del dispositivo con el fin de poder administrar el mismo desde la consola de ONOS</p>

FIGURA 4.11: Requerimientos para el driver desarrollado.

4.3.2. Descubrimiento del dispositivo

El controlador ONOS reconoce la presencia de un nuevo dispositivo a través de un mensaje en formato JSON, el cual contiene información como la dirección IP del equipo, el driver que describe sus comportamientos, el protocolo que utiliza, entre otra información de utilidad.

Un ejemplo de este mensaje se muestra en la figura 4.6. Dicho mensaje, es enviado al controlador haciendo uso del comando de ONOS “onos-netcfg” [].

LISTING 4.6: Mensaje JSON con información del dispositivo.

```
{
  "devices": {
    "netconf:172.16.0.141:830": {
      "netconf": {
        "ip": "172.16.0.141",
        "port": 830,
        "username": "user",
        "password": "pass",
      },
      "basic": {
        "driver": "altura-netconf"
      }
    }
  }
}
```

Al momento de indicar al controlador ONOS la presencia de un nuevo dispositivo, el mismo hace una llamada por única vez a la función DeviceDescription-Discovery, la cual se encuentra implementada en el Driver indicado por el archivo JSON.

Esta función, tiene la tarea de descubrir las características más generales del equipo, como ser la versión de software y de hardware del mismo, el número de puertos disponibles, el identificador único del dispositivo, etc.

Con más detalle, lo que realiza la función es iniciar la sesión SSH del protocolo NETCONF, esperar a que termine el intercambio de capacidades entre cliente y servidor y por último, enviar un mensaje NETCONF al servidor solicitando con la operación "GET", los siguientes datos de estado: información del fabricante, versión del hardware, software, identificador único del equipo y las alarmas que estén activas.

Como se aclaró, en esta función también se debe realizar el descubrimiento de los puertos del dispositivo, pero como estos no cambian a lo largo de la vida del equipo, no se envía un mensaje al servidor solicitando información de los mismos, sino que la información se encuentra especificada en el propio Driver.

La figura 4.12 muestra el flujo de actividad típico que tendría el controlador ONOS al momento de agregarse un nuevo dispositivo administrado por el Driver desarrollado.

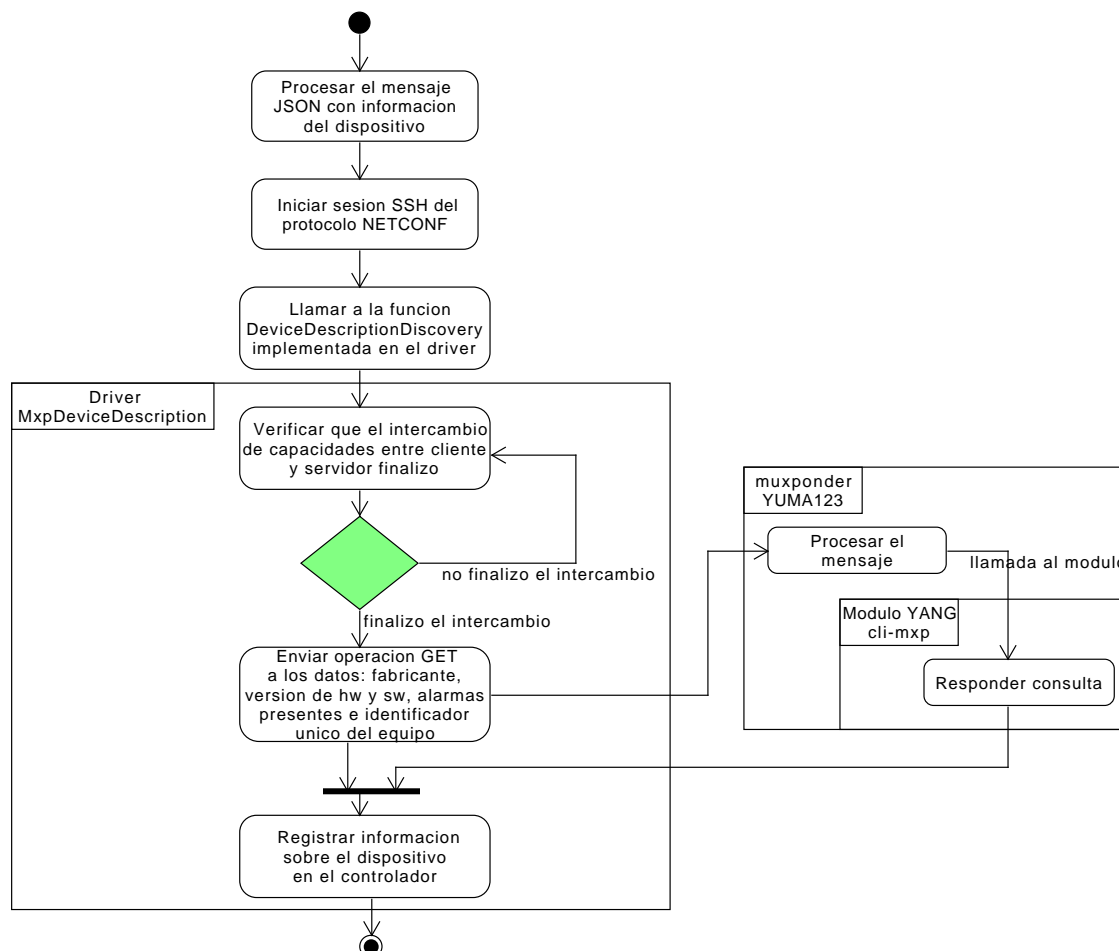


FIGURA 4.12: Topología implementada en el proyecto.

4.3.3. Descubrimiento de Enlaces

El Driver también debe proveer un mecanismo para indicar al controlador cómo se componen los enlaces entre los diferentes dispositivos administrados. Para ello, el controlador ONOS brinda una interfaz llamada "LinkDiscovery", la cual se deberá implementar en el driver desarrollado.

Así, el controlador llama periódicamente a esta función (cada 30 segundos de forma predeterminada, pudiéndose cambiar este tiempo desde la CLI) para corroborar el estado de los enlaces.

Con más detalle, lo que realiza esta función es enviar periódicamente a los dispositivos un mensaje NETCONF con la operación GET-CONFIG, consultando por los datos "port", "neighbor" y "port-neighbor" del container mux-config, estos datos pueden verse representados en el módulo YANG, en la figura 4.2. A continuación, se explica de forma breve la función de cada uno de estos datos:

- **port**: indica el puerto del dispositivo local al cual se conectará un vecino.
- **neighbor**: contiene el identificador único del dispositivo vecino al que se hace referencia en "port".
- **port**: indica el puerto del dispositivo vecino con el que se deberá formar el enlace.

Con esta información, el controlador comprueba si el dispositivo local o el vecino tienen alarmas registradas respecto al enlace de línea del muxponder, concretamente lo hace mediante las alarmas "RXS" y "Rx LOCK ERR".

Si alguno de los dispositivos involucrados contiene una de estas alarmas, el enlace no se forma. De lo contrario, si no tienen estas alarmas, el driver informa al controlador que forme un enlace óptico entre ambos dispositivos. El diagrama de actividad de la figura 4.13 muestra lo explicado anteriormente.

Es importante notar que en esta instancia, no se consulta al dispositivo por sus alarmas con un mensaje NETCONF, ya que las mismas son enviadas asincrónamente mediante notificaciones NETCONF por el dispositivo, y registradas por el controlador como alarmas. Por lo tanto, para verificar el estado de las alarmas de un dispositivo, solo se consulta internamente en el core de ONOS.

4.3.4. Operaciones definidas en el Driver

El driver desarrollado brinda, además de las implementaciones de LinkDiscovery y DeviceDescriptionDiscovery explicadas anteriormente, la posibilidad de ejecutar cualquier operación admitida por el módulo del dispositivo, entre ellas la RPC que se describió en la figura 4.4.

Además, se especifican comandos CLI para que el administrador pueda interactuar con los dispositivos a través del driver, haciendo uso de la consola de ONOS.

Se explica así el funcionamiento de la interfaz implementada para la operación RPC "mux-apply-config". En el driver, esta función tiene una utilidad más además de indicar al dispositivo que tiene que aplicar la configuración.

Dado que se puede indicar la presencia de dispositivos vecinos, es necesario corroborar que la configuración aplicada entre ellos sea la misma para garantizar la conectividad entre los clientes conectados a los muxponders. Así, si se aplica una

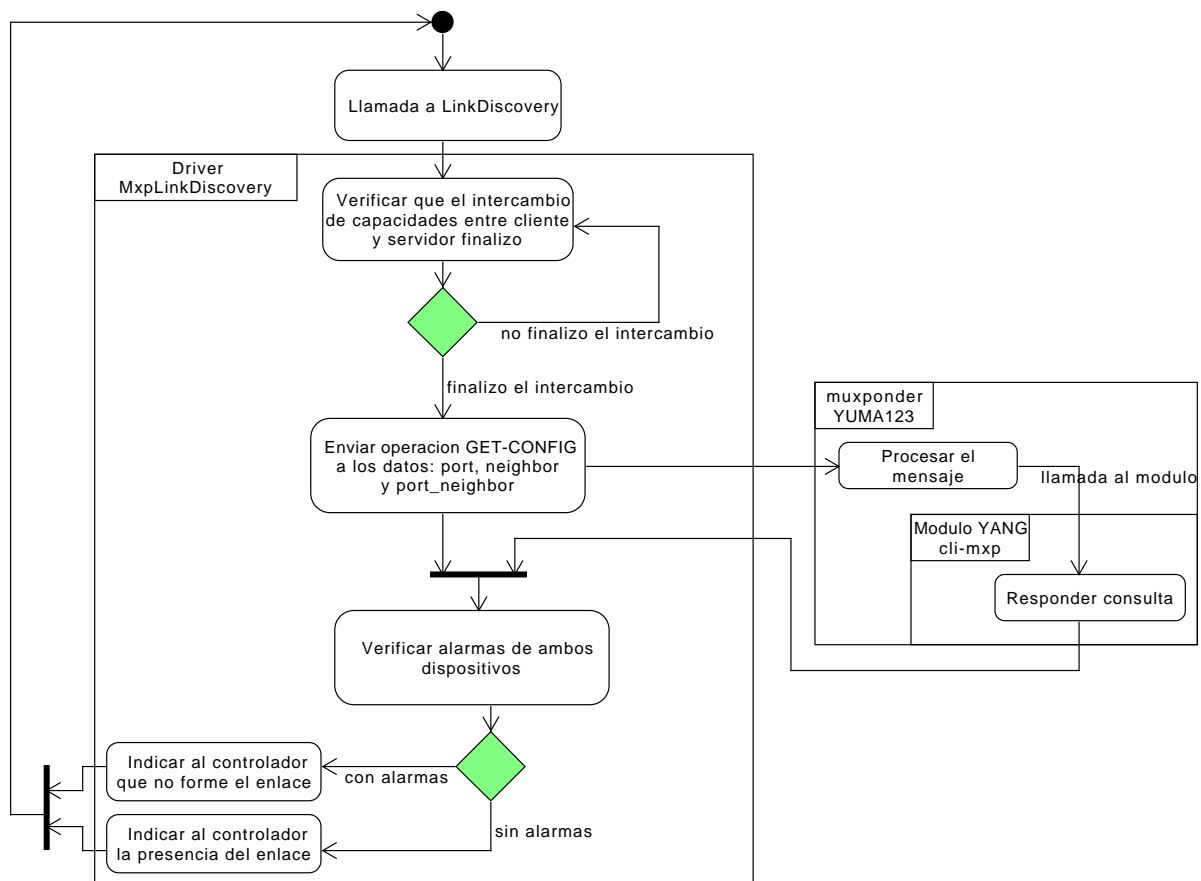


FIGURA 4.13: Topología implementada en el proyecto.

configuración en un muxponder, el controlador deberá corroborar que los dispositivos vecinos conectados tengan la misma configuración aplicada, de lo contrario se deberá generar y registrar una alarma en el controlador.

Teniendo en cuenta esto, el diagrama de la figura 4.14 muestra el flujo de actividad que sigue el driver cuando recibe una llamada a esta RPC para el muxponder A. Para este primer caso, el muxponder A no tiene especificado ningún vecino, por lo que no se generará alguna alarma sobre configuración inconsistente, de hecho, se verifica que no tenga alarmas de este tipo y de tenerlas se eliminan.

En primer lugar, cuando se llama a la función `rpcApplyConfig`, el driver envía un mensaje al dispositivo con la operación RPC que define el módulo YANG. El agente YUMA123 recibe este mensaje, lo procesa y aplica la configuración en el equipo haciendo uso de los datos de configuración que tenga el datastore running. Por último, se envía un mensaje con la respuesta de esa operación.

Luego, el driver consulta al mismo muxponder si tiene algún vecino conectado, como en este caso no se tiene ningún vecino conectado, se eliminan las alarmas (si existe alguna) relacionadas al muxponder A con configuración inconsistente entre los vecinos.

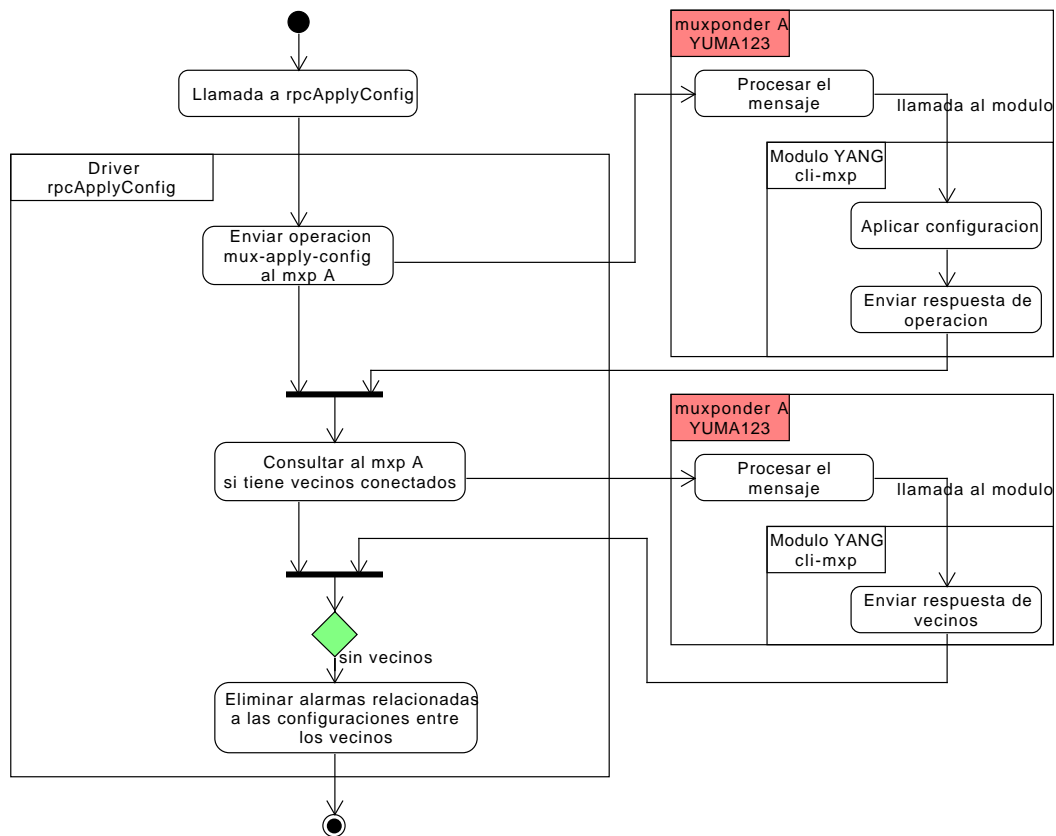


FIGURA 4.14: Topología implementada en el proyecto.

En caso de que el muxponder A tenga vecinos conectados, el flujo de actividad es el que se muestra en la figura 4.15. Así, se consulta al dispositivo vecino (muxponder B) su configuración aplicada y se la compara con la configuración aplicada recientemente al dispositivo local (muxponder A). Si las configuraciones resultan ser las mismas, se buscan las alarmas relacionadas a configuración inconsistente entre el muxponder A y B, y de existir, se eliminan. Por el contrario, si la configuración es distinta, se crea una alarma y se la registra en el controlador.

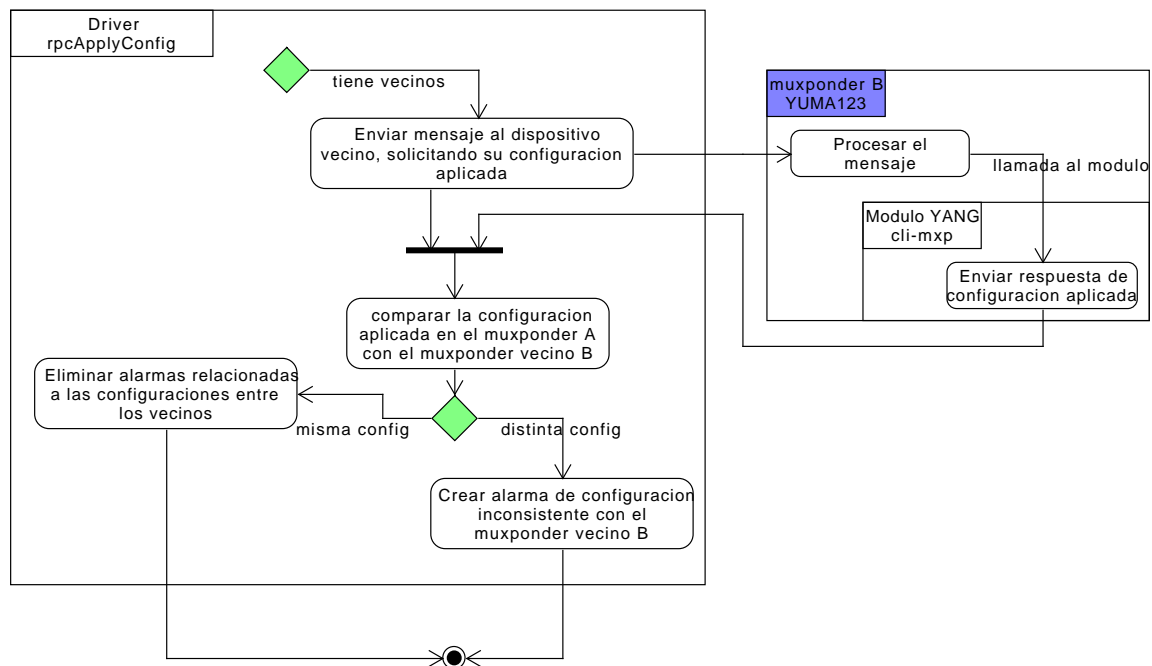


FIGURA 4.15: Topología implementada en el proyecto.

4.4. Diseño de la interfaz Northbound e Interfaz de usuario

Para cumplir con los requerimientos del sistema, será necesario crear en primera instancia, una interfaz REST API a las aplicaciones externas, para que las mismas puedan comunicarse con los dispositivos administrados (muxponders) por el controlador. Como se estudió en capítulos anteriores, ONOS utiliza una interfaz llamada Northbound para comunicarse con la capa de aplicación, por lo que la aplicación REST estará ubicada en dicha interfaz.

También, se deberá diseñar y crear una aplicación web, que sirva como interfaz de usuario al administrador. A diferencia de la aplicación REST, la GUI desarrollada residirá en la capa de aplicación. La figura 4.16 esclarece la ubicación de las aplicaciones mencionadas anteriormente.

«Requerimiento» Controlador SDN id: R-01 El sistema debe brindar el servicio de un controlador SDN	«Requerimiento» Conexión con el controlador id: R-02 Se debe establecer una conexión entre los muxponders y el controlador
«Requerimiento» Servidor web id: R-03 El sistema debe proveer el servicio de una interfaz web para administrar los dispositivos.	«Requerimiento» Soporte NETCONF id: R-04 Los dispositivos y el controlador deben soportar el protocolo NETCONF.
«Requerimiento» Nodos físicos id: R-05 El sistema debe soportar la conexión de nodos físicos.	«Requerimiento» Nodos virtuales id: R-06 El sistema debe soportar la conexión de nodos virtuales.

FIGURA 4.16: Topología implementada en el proyecto.

4.4.1. Requerimientos

A continuación, se listan en la figura 4.17 los diferentes requerimientos que deberán cumplir la interfaz REST y la GUI. Los requerimientos R-15, R-16 y R-17 corresponden a la interfaz REST, mientras que los requerimientos R-18 a R-23 pertenecen a la GUI desarrollada.

<p>«Requerimiento» Controlador SDN</p> <p>id: R-01 El sistema debe brindar el servicio de un controlador SDN</p>	<p>«Requerimiento» Conexión con el controlador</p> <p>id: R-02 Se debe establecer una conexión entre los muxponders y el controlador</p>
<p>«Requerimiento» Servidor web</p> <p>id: R-03 El sistema debe proveer el servicio de una interfaz web para administrar los dispositivos.</p>	<p>«Requerimiento» Soporte NETCONF</p> <p>id: R-04 Los dispositivos y el controlador deben soportar el protocolo NETCONF.</p>
<p>«Requerimiento» Nodos físicos</p> <p>id: R-05 El sistema debe soportar la conexión de nodos físicos.</p>	<p>«Requerimiento» Nodos virtuales</p> <p>id: R-06 El sistema debe soportar la conexión de nodos virtuales.</p>

FIGURA 4.17: Topología implementada en el proyecto.

4.4.2. Implementación de la REST

Para el desarrollo de la aplicación que se ejecuta en la interfaz Northbound del controlador, se utilizó la herramienta onos-create-app [1]. La misma, crea un esqueleto de una aplicación simple con una interfaz REST, a partir de la cual se realizaron modificaciones para poder cumplir con los requerimientos R-15, R-16 y R-17.

Así, la aplicación REST API se encuentra dividida en cinco clases de Java, las cuales se detallan a continuación.

- **AppComponent:** esta clase resulta del uso de la herramienta onos-create-app. Aquí, se define el comportamiento que tendrá la aplicación al momento de su activación y desactivación. En este caso, cuando se activa la aplicación en el controlador, la misma inicia un objeto Listener para poder imprimir mensajes de log y debug.
- **AppWebApplication:** también resulta del uso de la aplicación mencionada anteriormente. El objetivo de esta clase, es la de indicar cuáles serán las funciones y las clases de la aplicación que se exponerán en la Northbound interface de ONOS.
- **GetWebResource:** en esta clase se definen las operaciones de consulta que son expuestas, a través de la clase AppWebApplication, a la interfaz Northbound

del controlador. En ella, se definen funciones que tienen operaciones GET de HTTP, las cuales aceptan ciertos parámetros dependiendo de la operación (por ejemplo, indicar a qué dispositivo se quiere realizar la consulta). Seguidamente, la función llama al Driver del dispositivo con los parámetros que recibió, y devuelve una respuesta a las aplicaciones que la llamaron.

- **RpcWebResource**: de forma similar, esta clase expone una interfaz REST API a la RPC mux-apply-config definida en el módulo YANG. Así, las aplicaciones externas especifican el id de un dispositivo, para que luego la interfaz REST se comunique con el mismo mediante el Driver desarrollado.
- **SetWebResource**: por último, se expone una interfaz con operaciones PUT de HTTP, con las cuales se posibilita que las aplicaciones externas puedan realizar cambios en las bases de datos running, candidate o startup.

Se muestra en la figura 4.17 un ejemplo de la interfaz REST desarrollada. En la misma, se puede observar las diferentes operaciones expuestas explicadas anteriormente.

<p>«Requerimiento» Controlador SDN</p> <p>id: R-01 El sistema debe brindar el servicio de un controlador SDN</p>	<p>«Requerimiento» Conexión con el controlador</p> <p>id: R-02 Se debe establecer una conexión entre los muxponders y el controlador</p>
<p>«Requerimiento» Servidor web</p> <p>id: R-03 El sistema debe proveer el servicio de una interfaz web para administrar los dispositivos.</p>	<p>«Requerimiento» Soporte NETCONF</p> <p>id: R-04 Los dispositivos y el controlador deben soportar el protocolo NETCONF.</p>
<p>«Requerimiento» Nodos físicos</p> <p>id: R-05 El sistema debe soportar la conexión de nodos físicos.</p>	<p>«Requerimiento» Nodos virtuales</p> <p>id: R-06 El sistema debe soportar la conexión de nodos virtuales.</p>

FIGURA 4.18: Topología implementada en el proyecto.

4.4.3. Implementación de la GUI

A fin de cumplir con los requerimientos R-18 a R-23, se realizó una aplicación WEB basada en Flask []. La misma, se desarrolla en Python y hace uso de archivos HTML, JavaScript y CSS para presentar la interfaz de usuario. Cada sección listada en los requerimientos de la figura 4.17, corresponde a un archivo HTML que contiene la estructura de la aplicación y los datos que se presentan en la interfaz.

Es importante mencionar que todas las vistas realizan una tarea común. La misma consiste en consultar periódicamente al controlador, las alarmas activas que tiene

el mismo. Se puede observar el diagrama de actividad de la tarea mencionada en la figura 4.19.

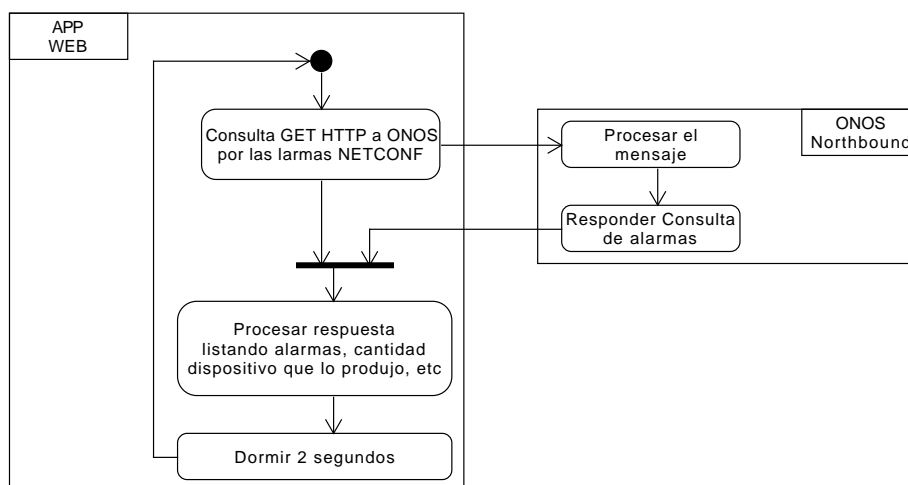


FIGURA 4.19: Topología implementada en el proyecto.

Como ya se explicó en este capítulo, las alarmas son enviadas por los dispositivos a través de las notificaciones NETCONF. Luego, el controlador las registra internamente, por lo que no es necesario hacer uso de la interfaz Southbound para consultar a los equipos por el estado de las alarmas. De esta forma, los mismos se ven aliviados al no tener que procesar periódicamente estas consultas.

Teniendo en cuenta esta actividad común, el desarrollo de la GUI se divide en las siguientes secciones:

- **Vista principal:** esta vista muestra una información resumida sobre las alarmas, como ser la cantidad de las mismas. A su vez, en la parte inferior de esta vista se permite agregar nuevos dispositivos a la topología, indicando su dirección IP y el puerto. Por último, en la zona superior se brinda un campo donde puede seleccionarse un conjunto de equipos para posteriormente aplicar una configuración con un perfil dado.

Así, en el diagrama de la figura 4.20, se muestra como la aplicación web conforma un mensaje JSON con la información del equipo (dirección IP, puerto) y la envía al controlador a través de la interfaz REST. Luego, el controlador procesa el mensaje dando inicio al descubrimiento del dispositivo.

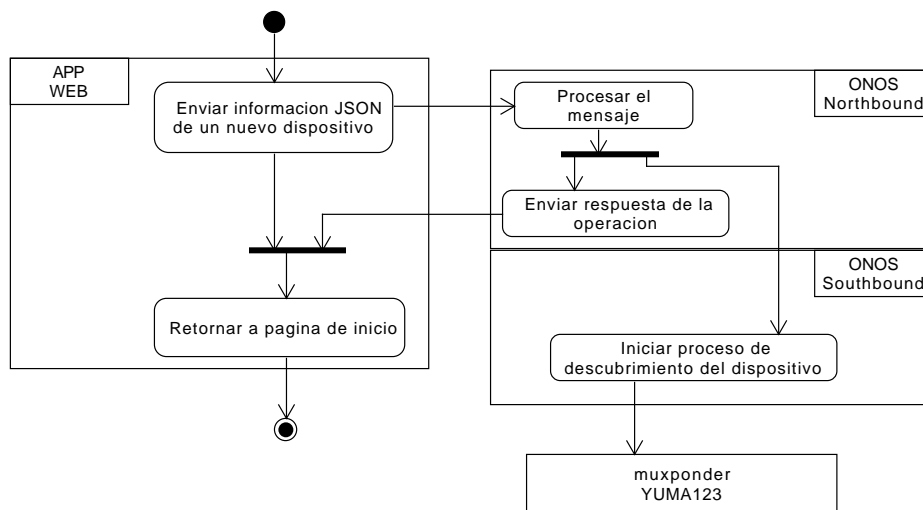


FIGURA 4.20: Topología implementada en el proyecto.

Por otra parte, la figura 4.21 muestra como es el proceso de configuración de un muxponder a través de la GUI. Primeramente, se envía el perfil de configuración, una vez que la información es almacenada en el datastore running, se envía la RPC mux-apply-config para que se apliquen los cambios en el dispositivo.

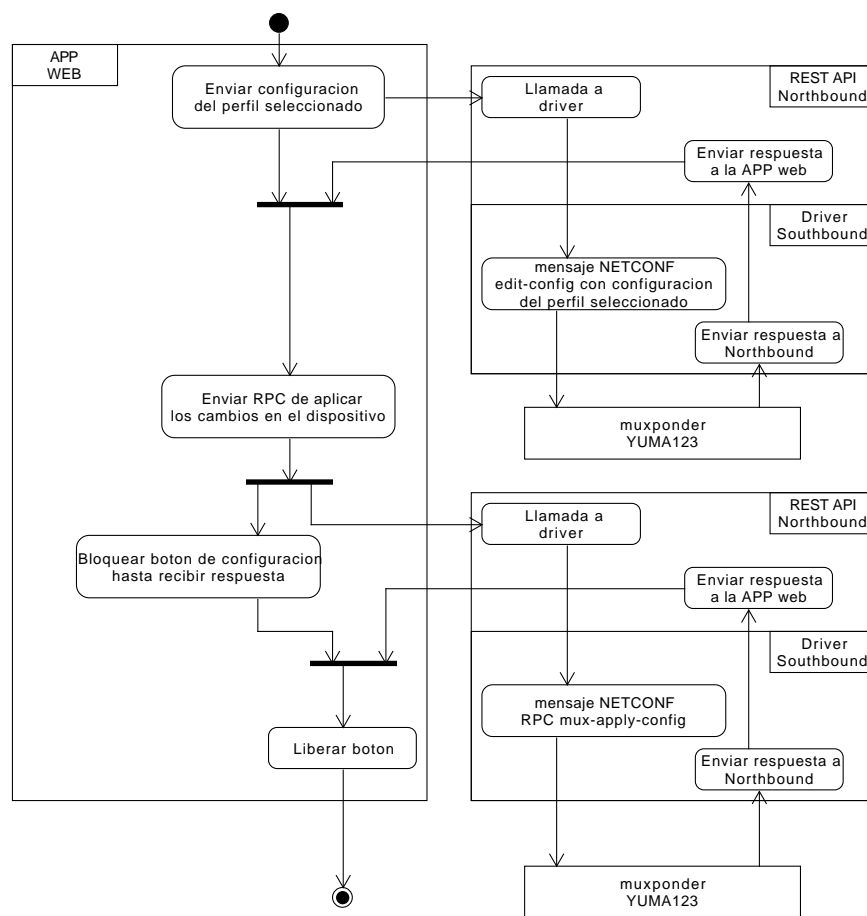


FIGURA 4.21: Topología implementada en el proyecto.

Por último, la figura 4.22 muestra la interfaz gráfica que presenta esta vista.

«Requerimiento» Controlador SDN	«Requerimiento» Conexión con el controlador
id: R-01 El sistema debe brindar el servicio de un controlador SDN	id: R-02 Se debe establecer una conexión entre los muxponders y el controlador
«Requerimiento» Servidor web	«Requerimiento» Soporte NETCONF
id: R-03 El sistema debe proveer el servicio de una interfaz web para administrar los dispositivos.	id: R-04 Los dispositivos y el controlador deben soportar el protocolo NETCONF.
«Requerimiento» Nodos físicos	«Requerimiento» Nodos virtuales
id: R-05 El sistema debe soportar la conexión de nodos físicos.	id: R-06 El sistema debe soportar la conexión de nodos virtuales.

FIGURA 4.22: Topología implementada en el proyecto.

- **Vista de alarmas:** muestra una información más detallada de las alarmas. El administrador puede identificar el tipo de alarma, el dispositivo que lo originó y la cantidad de alarmas totales que presenta la red.
- **Vista de datos de configuración:** presenta una sección donde se puede observar una lista de los equipos que conforman la topología con sus configuraciones aplicadas.
- **Vista de datos de estado:** de forma similar a la vista anterior, se muestra en esta sección información relacionada a los datos de estado de los equipos.
- **Vista de topología:** permite realizar cambios en la topología de la red, a través de mensajes NETCONF a los dispositivos. Así, se puede especificar una pareja de equipos para que el controlador, a través de la función LinkDiscovery, pueda formar los enlaces entre los mismos.
- **Vista de perfiles:** muestra una sección donde se permite agregar o eliminar perfiles de configuración. A continuación, se presentan diagramas de actividad para ambas tareas.

Bibliografía

- [1] *A Simple Network Management Protocol (SNMP)*. URL: <https://tools.ietf.org/html/rfc1157>.
- [2] *A Simple Network Management Protocol (SNMP)*. URL: <https://tools.ietf.org/html/rfc1157#section-3>.
- [3] M. Bjorklund. *A YANG Data Model for System Management*. URL: <https://www.ietf.org/rfc/rfc7317.txt>.
- [4] CESNET. *Netopeer*. URL: <https://github.com/CESNET/netopeer>.
- [5] CESNET. *Netopeer2*. URL: <https://github.com/CESNET/Netopeer2>.
- [6] Nimesh Dubey. *From Static Networks to Software-driven Networks*. URL: <https://www.isaca.org/Journal/archives/2016/volume-4/Pages/from-static-networks-to-software-driven-networks-an-evolution-in-process.aspx>.
- [7] Nick Feamster. *The Road to SDN - An intellectual history of programmable networks*. URL: <https://queue.acm.org/detail.cfm?id=2560327>.
- [8] Open Networking Foundation. *Software-Defined Networking: The New Norm for Networks*. Inf. téc. ONF, 2013.
- [9] *G.709 : Interfaces para la red óptica de transporte*. URL: <https://www.itu.int/rec/T-REC-G.709/es>.
- [10] Paul Göransson, Chuck Black y Timothy Culver. *Software Defined Networks A Comprehensive Approach*. Second edition. Elseiver, 2017.
- [11] *Introducing ONOS -a SDN network operating system for Service Providers*. URL: <http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf>.
- [12] *License, Patents, and Contributor Agreement*. URL: <https://onosproject.org/agreement/>.
- [13] SDNCentral LLC. *2017 Network Virtualization Report SDN Controllers, Cloud Networking and More*. Inf. téc. SDx Central, 2017.
- [14] Carl Moberg. *A 30-minute Introduction to NETCONF and YANG*. URL: <https://www.slideshare.net/cmoberg/a-30minute-introduction-to-netconf-and-yang>.
- [15] *Muxponder: Take a Fresh Look at 100G*. URL: <http://www.fiberopticshare.com/muxponder-take-fresh-look-100g.html>.
- [16] *NETCONF Configuration Protocol*. URL: <https://tools.ietf.org/html/rfc4741>.
- [17] *Network Configuration Protocol (NETCONF)*. URL: <https://tools.ietf.org/html/rfc6241>.

- [18] *Nios® II Processors*. URL: <https://www.intel.com/content/www/us/en/programmable/products/processors/support.html>.
- [19] Karim Okasha. *Network Automation and the Rise of NETCONF*. URL: <https://medium.com/k.okasha/network-automation-and-the-rise-of-netconf-e96cc33fe28>.
- [20] Object Management Group OMG. *What is SysML?* URL: <http://www.omg.sysml.org/what-is-sysml.html> (visitado 02-01-2018).
- [21] ONOS: *Towards an Open, Distributed SDN OS*. URL: <https://onosproject.org/wp-content/uploads/2014/11/HotSDN-paper-2014-ONOS-Towards-an-Open-Distributed-SDN-OS.pdf>.
- [22] *Open Networking Foundation*. URL: <https://www.opennetworking.org/>.
- [23] *Optical Network - Definition - What does Optical Network mean?* URL: <https://www.techopedia.com/definition/23643/optical-network>.
- [24] *Organizations supporting the Open Network Operating System*. URL: <https://onosproject.org/members/>.
- [25] *Overview of the 2002 IAB Network Management Workshop*. URL: <https://www.ietf.org/rfc/rfc3535.txt>.
- [26] *OWASP Top Ten Cheat Sheet*. URL: https://www.owasp.org/index.php/OWASP_Top_Ten_Cheat_Sheet.
- [27] Sterling Perrin. *SDH Network Modernization With Multiservice OTN*. URL: <http://www-file.huawei.com/~media/CORPORATE/PDF/white%20paper/sdh-network-modernization-with-multiservice-otn.pdf>.
- [28] Christos Rizos. *Why use NETCONF/YANG when you can use SNMP and CLI?* URL: <https://snmpcenter.com/why-use-netconf/>.
- [29] Margaret Rouse. *The Data Plane*. URL: <https://searchnetworking.techtarget.com/definition/data-plane-DP>.
- [30] Brent Salisbury. *The Control Plane, Data Plane and Forwarding Plane in Networks*. URL: <http://networkstatic.net/the-control-plane-data-plane-and-forwarding-plane-in-networks/>.
- [31] *SDN architecture*. URL: https://www.opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf.
- [32] *Software-Defined Multilayer Networks*. URL: <https://www.ecitele.com/wp-content/uploads/2018/10/trains-planes-and-more-fibre-systems-spring-2017.pdf>.
- [33] Sysrepo. *Sysrepo*. URL: <https://github.com/sysrepo/sysrepo>.
- [34] Sysrepo. *Sysrepo*. URL: <https://github.com/sysrepo/sysrepo/blob/master/INSTALL.md>.
- [35] *Understanding NETCONF and YANG*. URL: <https://www.networkworld.com/article/2173842/understanding-netconf-and-yang.html>.
- [36] *Understanding the SDN Architecture – SDN Control Plane and SDN Data Plane*. URL: <https://www.sdxcentral.com/networking/sdn/definitions/inside-sdn-architecture/>.
- [37] Vladimir Vassilev. *Yuma123*. URL: <https://github.com/vlvassilev/yuma123>.

-
- [38] Vladimir Vassilev. *Yuma123*. URL: http://yuma123.org/wiki/index.php/Yuma_netconfd_Manual#Features.
 - [39] YANG - *A Data Modeling Language for the Network Configuration Protocol (NETCONF)*. URL: <https://tools.ietf.org/html/rfc6020>.
 - [40] YumaWorks. *YumaPro*. URL: <https://www.yumaworks.com/features/open-source/>.