Modelo en YANG del protocolo de encaminamiento RPL en redes LLN a través de RESTCONF

Douglas Andrés Ramírez Brujes

Trabajo de Grado para Optar el título de Ingeniero de Sistemas

Director
Pedro Javier Trujillo Tarazona
Mg. Informática

Universidad Industrial de Santander
Facultad de Ingenierías Fisicomecánicas
Escuela de Ingeniería de Sistemas e Informática
Bucaramanga
2020

Agradecimientos

En primera instancia a mis padres, por su apoyo moral y económico, siempre presente e incondicional, para poder culminar con mis estudios.

A mi director de Trabajo de Grado, Pedro Javier Trujillo Tarazona, por su guía en la redacción de los documentos, apoyo efectivo, interés durante el desarrollo del proyecto de grado, pronta asistencia respecto a los problemas presentados, y especialmente en orientación sobre las temáticas más complejas sobre el protocolo de encaminamiento RPL, Network Programming, Network DevOps y documentación de temáticas de redes de computadoras avanzadas.

A Lasdislav Lhotka, cocreador de varias de las herramientas usadas para el desarrollo de este proyecto, por su pronta y efectiva ayuda en la solución de un complejo problema presentado.

A la Escuela de Ingeniería de Sistemas por proveer las herramientas y recursos necesarios durante toda la carrera.

A todo el personal docente y administrativo de la escuela de Ingeniería de Sistemas, por permitirme y ayudarme a forjarme como profesional.

Tabla de Contenido

Introducción 17

1. Objetivos 19

1.1 Objetivo general 19

1.2 Objetivos específicos 19

2. Marco conceptual 20

2.1 Fundamentos teóricos 20

2.1.1 YANG. 20

2.1.2 RESTCONF. 23

2.1.3 RPL. 25

2.1.4 NMDA. 29

2.1.5 Modelo de datos YANG para la NMDA. 30

2.1.6 Biblioteca YANG. 33

2.1.7 Arquitectura cliente-servidor. 33

2.2 Herramientas implementadas 34

2.2.1 Pyang. 34

2.2.2 Yangson. 35

2.2.3 Oracle VM Virtual Box. 36

2.2.4 Cygwin. 37

2.2.5 Curl. 38

2.2.6 Jetconf. 39

2.2.7 Pyangbind. 41

2.3 Antecedentes 41

3. Metodología 42

3.1 Segmentos del protocolo RPL modelados 44

3.1.1 Segmentos compuestos. 47

3.2 Modelado YANG 47

3.2.1 Construcción de los segmentos independientes. 48

3.2.2 Segmentos compuestos. 49

3.2.3 RPC. 50

3.2.4 Cumplimiento con el estándar para la NMDA. 51

3.3 Validación del modelo YANG 53

3.3.1 Validación de sintaxis. 53

3.3.2 Validación de implementación. 54

3.3.3 Validación de datos. 55

3.3.4 Validación con Pyangbind. 56

3.4 Implementación del servidor RESTCONF 57

3.4.1 Backend de Jetconf. 57

3.4.2 Configuración del servidor Jetconf. 58

3.5 Diseño de pruebas para la interacción Modelo-RESTCONF por HTTP 59

3.5.1 Acceso y visualización de datos. 60

3.5.2 Creación de recursos (adición de datos). 60

3.5.3 Modificación de recursos (actualización de datos). 61

3.5.4 Llamada a procedimiento remoto. 61

3.6 Visión macroscópica del proyecto 61

4. Resultados 63

4.1 Modelo YANG del protocolo de encaminamiento RPL 63

4.1.1 Módulo uis-rpl. 63

4.1.2 Módulo uis-ofs-of0. 66

4.2 Ejecución del servidor Jetconf 67

4.2.1 Datos de simulación de enrutador en ejecución. 67

4.3 Pruebas RESTCONF 68

4.3.1 Pruebas de acceso y visualización de datos. 69

4.3.2 Pruebas de adición de datos. 73

4.3.3 Pruebas de actualización de datos. 75

4.3.4 Pruebas al RPC. 78

5. Conclusiones 79

6. Recomendaciones 80

Referencias Bibliográficas 81

Lista de Tablas

Tabla 1. Segmentos compuestos y su estructura. 50

Tabla 2. Formato para diseño de pruebas. 60

Tabla 3. Diseño de prueba de visualización de datos. 60

Tabla 4. Diseño de prueba de adición de datos. 60

Tabla 5. Diseño de prueba de actualización de datos. 61

Tabla 6. Diseño de prueba para llamadas a procedimientos remotos. 61

Lista de Figuras

Figura 1. Encabezado y definiciones de tipos de dato. 22

Figura 2. Agrupamiento de atributos y contenedor de instancias. 23

Figura 3. Pila de protocolos de RESTCONF 24

Figura 4. DAG 25

Figura 5. DODAG 25

Figura 6. Comunicación entre nodos 27

Figura 7. Tablas de rutas de cada nodo: modo storing 28

Figura 8. Tabla de rutas solo en el nodo raíz: modo no storing 28

Figura 9. Estructura parcial de árbol del modelo YANG ietf-routing. 31

Figura 10. Maquetado en YANG de los metadatos de una ruta. 32

Figura 11. Arquitectura cliente-servidor 33

Figura 12. Comando Pyang y resultado. 34

Figura 13. Validación de implementación independiente exitosa. 35

Figura 14. Validación de datos e implementación. 35

Figura 15. Direcciones IP del servidor (izquierda) y el cliente (derecha) 37

Figura 16. Petición construida con Curl y respuesta del servidor 38

Figura 17. Código escrito en Python para el RPC "get_route_stack". 40

Figura 18. Atributos globales métrica por defecto y distancia en el modelo ietf-rip 42

Figura 19. Diagrama de flujo de la metodología. 43

Figura 20. Segmento como conjunto abstracto de funcionalidades de un protocolo. 44

Figura 21. Diagrama de flujo del procedimiento de selección de segmentos relevantes. 45

Figura 22. Árbol jerárquico del módulo ietf-routing. 51

Figura 23. Instrucción para la extensión del módulo ietf-routing. 52

Figura 24. Sección del árbol de datos jerárquico generado con Pyang. 53

Figura 25. Dos instancias de módulos YANG, una en implement y otra en import. 55

Figura 26. Instancia de ietf-interfaces en un datastore en formato Json. 56

Figura 27. Agregación de un objeto de tipo función objetivo a un objeto de tipo RPL. 57

Figura 28. Clase para la generación de la lista de la tabla de rutas, de tipo datos de estado (ro). 58

Figura 29. Configuración principal del servidor Jetconf. 59

Figura 30. Diagrama del sistema. 62

Figura 31. Todos los segmentos independientes, exceptuando el segmento Estadísticas..... 64

Figura 32. Segmento estadísticas y RPC. 65

Figura 33. Instrucción de extensión al módulo uis-rpl 66

Figura 34. Módulo uis-ofs-of0. 66

Figura 35. Mensajes informativos de Jetconf al terminar la inicialización. 67

Figura 36. Topología planteada. 68

Figura 37. Código del script test_get_root.sh 69

Figura 38. Respuesta del servidor con datastore completo. 70

Figura 39. Código del script test_get_state_data.sh. 71

Figura 40. Tabla de padres elegidos como respuesta a la petición hecha al servidor. 71

Figura 41. Tabla de rutas como respuesta del servidor. 72

Figura 42. Lista de métricas definidas inicialmente en el datastore. 73

Figura 43. Código del script test_add_metric.sh. 74

Figura 44. Resultado devuelto por el servidor a la petición de adición de una métrica. 74

Figura 45. Respuesta del servidor al añadir una entrada cuyo nombre es Error. 75

Figura 46. Código del script test_update_metric.sh. 76

Figura 47. Respuesta del servidor a la petición de reemplazo de la entrada Error por HC. 76

Figura 48. Respuesta del servidor a una petición inválida. 77

Figura 49. Contenido del fichero get_route_stack.json 78

Figura 50. Código del script test_rpc_routes.sh. 78

Figura 51. Respuesta del servidor al RPC get_route_stack. 78

Lista de Apéndices

Ver apéndices adjuntos en el CD y pueden visualizarlos en base de datos de la Biblioteca UIS.

A. Apéndice A. Configuración cliente-servidor

B. Apéndice B. Configuración del backend “Empalme”

C. Apéndice C. Pruebas HTTP-RESTCONF

D. Apéndice D. Modulo YANG uis-rpl

E. Apéndice E. Modulo YANG uis-ofs-of0

Lista de siglas

API: Application Programming Interface (Interfaz de programación de aplicaciones).

DAG: Directed Acyclic Graph (Gráfico Acíclico Dirigido).

DODAG: Destination-Oriented DAG (DAG Orientado a un destino).

FIB: Forwarding Information Base (Base de información de reenvío).

HTTP: Hypertext Transfer Protocol (Protocolo de transferencia de hipertexto).

IEEE: Institute of Electrical and Electronics Engineers (Instituto de Ingenieros Eléctricos y Electrónicos).

IETF: Internet Engineering Task Force (Grupo de trabajo de ingeniería de internet).

IoT: Internet of Things (Internet de las cosas).

IP: Internet Protocol (Protocolo de internet).

IPv4: Internet Protocol version 4 (Protocolo de internet versión 4).

IPv6: Internet Protocol version 6 (Protocolo de internet versión 6).

JSON: JavaScript Object Notation (Notación de objeto de Javascript).

LLN: Low-Power and Lossy Network (Red de baja potencia y con pérdidas).

Network DevOps: Network Development and Operations (Desarrollo y operaciones sobre redes).

NMDA: Network Management Datastore Architecture (Arquitectura de almacenamiento para la administración de red).

REST: Representational State Transfer (Transferencia de estado representacional).

RFC: Request For Comments (Solicitud de comentarios).

RIB: Routing Information Base (Base de información de encaminamiento).

RIP: Routing Information Protocol (Protocolo de información de encaminamiento).

RPC: Remote Procedure Call (Llamada de procedimiento remoto).

RPL: Routing Protocol for LLNs (Protocolo de encaminamiento para LLNs).

SSH: Secure Shell (Consola segura).

WWW: World Wide WEB (Red informática mundial)**.**

XML: Extensible Markup Language (Lenguaje de mercado extensible).

YANG: Yet Another Next Generation (Otra nueva generación).

Resumen

TÍTULO: MODELO EN YANG DEL PROTOCOLO DE ENCAMINAMIENTO RPL EN REDES LLN A TRAVÉS DE RESTCONF[1]

AUTOR: DOUGLAS ANDRÉS RAMÍREZ BRUJES[2]

PALABRAS CLAVE: RPL, MODULO, YANG, REDES, LLN, RESTCONF

DESCRIPCIÓN:

Con el constante desarrollo y tendencia hacia la automatización de las redes, orientado al IoT, han surgido nuevas tecnologías y retos, por ejemplo, las redes LLN y los nuevos protocolos que trabajan para su funcionamiento. Estas tecnologías, en constante investigación, generan la oportunidad de desarrollar proyectos de ingeniería de software como el presente en este documento: realizar un modelo YANG, de los segmentos más relevantes del protocolo de encaminamiento RPL, que se pueda usar y configurar mediante el reciente protocolo de gestión de redes RESTCONF.

La elaboración del proyecto consiste en dos partes: la realización del modelo YANG del protocolo RPL y la implementación de este en un servidor que ejecute RESTCONF.

El desarrollo del modelo se realizó mediante la identificación de los segmentos del protocolo RPL a modelar, su respectiva codificación en YANG y tres validaciones, siguiendo los lineamientos y recomendaciones para cumplir con el estándar propuesto para la NMDA. Por otra parte, el servidor RESTCONF se instauró usando el software Jetconf, siendo este una implementación, escrita en Python 3 puro, del protocolo gestión de red RESTCONF.

Una vez finalizado el montaje del servidor RESTCONF, se realizaron pruebas de visualización, adición, y actualización de datos de simulación, cargados directamente desde un datastore, con resultados positivos y con el comportamiento esperado.

Finalmente, el modelo YANG del protocolo de encaminamiento RPL presenta las condiciones necesarias para cumplir con el propósito de ser un sólido punto de partida para un desarrollo posterior. Además, debido al diseño con el que se elaboró el modelo, este cumple con el estándar propuesto para la NMDA, extendiendo el módulo YANG ietf-routing.

[1] Trabajo de grado en Modalidad de Investigación.

[2] Facultad de ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática.
Director: Mg. Pedro Javier Trujillo Tarazona

Abstract

TITLE: YANG MODEL FOR LLNS ROUTING PROTOCOL RPL THROUGH RESTCONF[1]

AUTHOR: DOUGLAS ANDRÉS RAMÍREZ BRUJES[2]

KEYWORD: RPL, MODULE, YANG, NETWORKS, LLN, RESTCONF

DESCRIPTION:

With the constant development and trend towards network automation, oriented to the IoT, new technologies and challenges have emerged, for example, LLN networks and new protocols that work for their operation. These technologies, in constant investigation, generate the opportunity to develop software engineering projects such as the one presented in this document: make a YANG model, of the most relevant segments of the RPL routing protocol, which can be used and configured through the recent network management protocol RESTCONF.

The development of the project consists of two parts: the realization of the YANG model of the RPL protocol and the implementation of it in a server running RESTCONF.

The development of the model was carried out by identifying the segments of the RPL protocol to be modeled, their respective YANG coding and three strict validations, following the guidelines and recommendations to comply with the standard proposed for NMDA. On the other hand, the RESTCONF server was installed using Jetconf software, which is an implementation, written in pure Python 3, of the RESTCONF network management protocol.

After mounting the RESTCONF server, test of visualization, addition, data updating, and a remote procedure call were performed. The data are simulated and loaded directly from a datastore. The results had the expected behavior, within the possible scenario for each of them.

Finally, the YANG model of the RPL routing protocol presents the necessary conditions to fulfill the purpose of being a solid starting point for further development. Furthermore, due to the design with which the model was developed, it complies with the proposed NMDA standard, extending the YANG ietf-routing module.

[1] Undergraduate final Project, research modality.

[2] Department of Physical-mechanical Engineering. School of System Engineering and Computer Science. Advisor: Mg. Pedro Javier Trujillo Tarazona.

Introducción

En la actualidad, el concepto de Internet de las cosas (IoT) está en un constante crecimiento y desarrollo, debido a que este es un paradigma cuya misión es interconectar todo tipo de objetos o dispositivos a Internet, por ejemplo: pequeños nodos que recolecten información útil en granjas de cultivos, sensores y actuadores en la infraestructura de un edificio de oficinas o de una planta industrial, hasta redes de nodos ubicados estratégicamente en una ciudad o región con el fin de obtener diversos índices o modificar el comportamiento de procesos en esos entornos.

Las redes “Low Power and Lossy Networks” (LLN) se caracterizan por las limitaciones de los nodos en potencia, memoria y energía, y a las condiciones restringidas en los enlaces, puesto que se requiere una alta fiabilidad para la transmisión de los datos a pesar de que generalmente se trate de enlaces con alta posibilidad de pérdida de paquetes.

Por lo tanto, las condiciones para el correcto funcionamiento de estas redes son drásticamente diferentes a los de las redes tradicionales. Ergo, los protocolos de red no pueden ser los mismos a los de uso típico en las redes comunes de hoy en día.

En el estado del arte, se encuentra que la arquitectura de protocolos para IoT en redes LLN que utilizan IPv6 y RPL como protocolos esenciales es relevante en la medida que es estandarizada por la IETF (Internet Engineering Task Force).

De otra parte, hay una fuerte tendencia hacia la automatización de las redes, incluyendo las que se usen para IoT, por lo que sería conveniente disponer de un modelo del protocolo RPL que posibilite la configuración local y remota de los parámetros del protocolo propiciando elementos para la automatización.

Por ejemplo, ya existe una propuesta de modelo para el protocolo RIP (Routing Information Protocol), que se tomó como referente para el desarrollo de la propuesta de modelo que se elaboró para el protocolo RPL.

En la automatización de redes se han desarrollado estándares por la IETF que son YANG, NETCONF y RESTCONF. Con YANG se pueden elaborar modelos de datos de diferentes recursos, especialmente para redes de computadores. NETCONF y RESTCONF se utilizan para el proceso de la lectura y configuración de datos de los dispositivos y sistemas de red modelados con YANG. NETCONF es un estándar propuesto en la IETF mediante la RFC 4741 (Rob Enns, Bjorklund, Schoenwaelder, & Bierman, 2011) y actualizado mediante la RFC 6241 (Rob Enns et al., 2011) y posibilita elementos para la automatización del funcionamiento y administración de redes (por ejemplo, en la configuración remota programada de dispositivos y sistemas de red) a través de SSH. RESTCONF es un estándar propuesto mediante la RFC 8040 (A. Bierman, Bjorklund, & Watsen, 2017) para la automatización del funcionamiento y administración de redes (por ejemplo, en la configuración remota programada de dispositivos y sistemas de red) a través de servicios WEB tipo REST (Representational state transfer).

Por lo anterior, se encontró la oportunidad de desarrollar como aplicación de la ingeniería de software en el campo de las redes de computadores un modelo de datos YANG de los componentes principales del protocolo RPL que trabaje con el estándar RESTCONF para que se pueda utilizar como punto de partida o de aporte para el desarrollo de un modelo de datos YANG de todos los componentes del protocolo RPL necesarios para la automatización de redes LLN que implementen este protocolo.

1. Objetivos

1.1 Objetivo general

Elaborar un modelo en YANG del protocolo de encaminamiento RPL para redes LLN (Low Power and Lossy Networks) a través de RESTCONF.

1.2 Objetivos específicos

- Seleccionar los componentes principales del protocolo RPL a modelar.
- Codificar el modelo YANG del protocolo RPL con el fin de realizar llamadas de procedimiento remotos (RPC) a través de RESTCONF.
- Adaptar el modelo YANG del protocolo RPL extendiéndolo acorde a la propuesta de estándar del modelo de datos en YANG para la administración de enrutamiento estipulada por la IETF.
- Validar el funcionamiento del modelo YANG a través de Pyang y Pyangbind.

2. Marco conceptual

2.1 Fundamentos teóricos

2.1.1 YANG. Yang es un lenguaje de modelado de datos para la definición de datos enviados a través de protocolos de administración de red tales como NETCONF y RESFTCONF. (Björklund (Ed.), 2016)

Consiste en un lenguaje modular que representa las estructuras de datos a modo de árbol jerárquico. El lenguaje incluye diversos tipos de datos básicos, además de permitir la creación de otros tipos de datos concretos o con un uso específico. (Martin Björklund, 2010)

Surge de la necesidad de modelar, independientemente del protocolo, los datos de configuración y de estado de los diversos dispositivos de red disponibles.

YANG tiene su propia terminología, como se puede ver en el RFC 6020 (Martin Björklund, 2010) o, en el más reciente, RFC 7950 (Björklund (Ed.), 2016) sección 3. A continuación, un listado con la terminología más relevante y común al desarrollo de modelos en YANG:

- **Data node:** Es un nodo en un esquema que puede ser instanciado. Puede ser un container, leaf, list, etc.
- **Augment:** Añade nuevos nodos de un esquema o módulo a un nodo de otro esquema o módulo previamente definido.
- **Data model:** Un modelo de datos describe cómo los datos son representados y accedidos.
- **Data tree**: Es el árbol instanciado de datos de configuración y de estado de un dispositivo.
- **Container:** Es un nodo de datos cuyo objetivo es simplemente contener datos y que puede existir en máximo un árbol de datos. No tienen ningún valor y solo pueden contener nodos hijos.

- **Identifier:** Se usa para identificar diferentes tipos de ítems YANG por un nombre.
- **Derived type**: Es un tipo de dato derivado por un tipo de dato prexistente en YANG, o de algún otro tipo de dato derivado.
- **Leaf:** Es un nodo de datos que existe en máximo una instancia de un árbol de datos. Las leaf tienen un valor, pero no tienen nodos hijo.
- **List:** Es un nodo de datos similar a un container, con la diferencia de que éste sí puede existir en múltiples instancias de un árbol de datos.
- **Module:** Un módulo YANG define la jerarquía de nodos que pueden ser usador por operaciones NETCONF o RESTCONF.
- **State data:** Son los datos adicionales de un sistema que no corresponden a datos de configuración, por ejemplo, datos de información del estado del dispositivo de solo lectura o estadísticas recolectadas.

En YANG, cada modelo es representado como un módulo (module) y éste contiene nodos de datos. Los nodos de datos, como se mencionó anteriormente en la lista de terminología, puede ser cualquier tipo de nodo. Se le llama nodo a todo tipo de esquema presente dentro de un módulo.

Todos los módulos pueden ser extendidos mediante Augment, con el fin de tener a disposición los nodos de un módulo externo completo o de una parte de este.

A continuación, un ejemplo de modelo YANG de los datos más básicos de un switch de capa dos (2).

```
module douglas-mini-switch {
  yang-version 1.1;
  namespace "https://www.uis.edu.co";
  prefix "mini-switch";
  description
    "Este es un módulo de ejemplo, el cuál modela un switch
     con características básicas.";

  /**
  * Nuevo tipo de dato que representa el número máximo de vlan asignado
  **/
  typedef vlan {
    type uint16 {
      range 1..4094; //Número máximo de vlans
    }
  }

  /**
  * Nuevo tipo de dato que representa el número de puertos del switch
  * que están habilitados.
  **/
  typedef puertos-habilitados {
    type uint16 {
      range 1..32; //Número de puertos presentes en el switch
    }
  }

```

Figura 1. Encabezado y definiciones de tipos de dato.

```
27
28    /**
29     * Agrupamiento de los datos más básicos de un switch.
30     **/
31    grouping datos-switch {
32      leaf vlan {
33        type vlan;
34      }
35      leaf nombre {
36        type string;
37      }
38      leaf puertos-habilitados {
39        type puertos-habilitados;
40      }
41    }
42
43    /**
44     * Contenedor que almacena una lista de switches instanciables.
45     **/
46    container switches {
47      list switch {
48        // Cada nuevo switch instanciado se debe identificar un nombre único.
49        key "nombre";
50        uses datos-switch; // Implementa el agrupamiento.
51      }
52    }
53  }
```

Figura 2. Agrupamiento de atributos y contenedor de instancias.

2.1.2 RESTCONF. RESTCONF es un protocolo basado en HTTP (Hypertext transfer protocol) que provee una interfaz programable para acceder a datos modelados en YANG.

Surge de la necesidad de estandarizar la forma en la que las aplicaciones web acceden o modifican los datos de configuración y estado, los llamados de procedimientos remotos y notificaciones, dadas por un dispositivo de red, de forma extensible y modular (Bjorklund, Schoenwaelder, Shafer, Watsen, & Wilton, 2019), de modo que se apliquen los principios y prácticas modernas de la reutilización de código y utilización de interfaces de programación de aplicaciones (APIs, por sus siglas en inglés).

El protocolo RESTCONF aprovecha los verbos planteados por la transferencia de estado representacional (REST, por sus siglas en inglés), para el intercambio de información a través de peticiones cuyos datos se transmiten en formato JSON o XML. Los datos solicitados o entregados por una petición van de acuerdo con el modelo de datos planteado en YANG, es decir, los datos de configuración y estado disponibles se definen a través de un modelo de datos en YANG.

En la figura 3 se muestra la pila de protocolos usada por el protocolo RESTCONF para la transmisión de información.

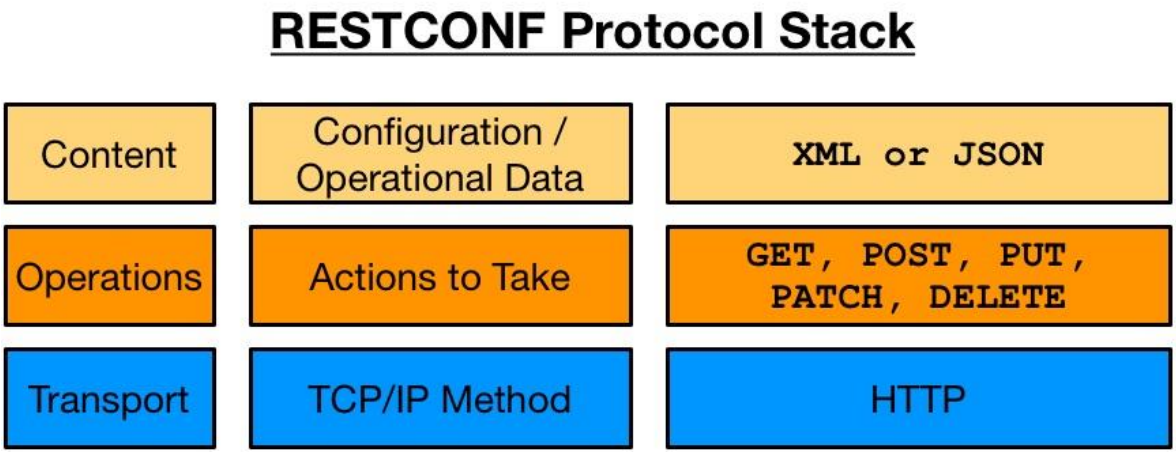


Figura 3. Pila de protocolos de RESTCONF. Cisco Live 2018. (Byrne, 2018)

2.1.3 RPL. El protocolo de enrutamiento para redes de baja potencia y alta pérdida (RPL, por sus siglas en inglés), es un protocolo de enrutamiento para redes inalámbricas de bajo consumo de energía y típicamente susceptibles a pérdidas de paquetes basado en la creación de vectores distancia (Alexander et al., 2012). Opera sobre IEEE 802.15.4.

RPL funciona mediante la construcción de vectores distancia a través de grafos acíclicos dirigidos (DAGs, por sus siglas en inglés) orientados a un destino (DODAGs, por sus siglas en inglés), donde el destino principal de toda la red de nodos se le denomina nodo raíz.

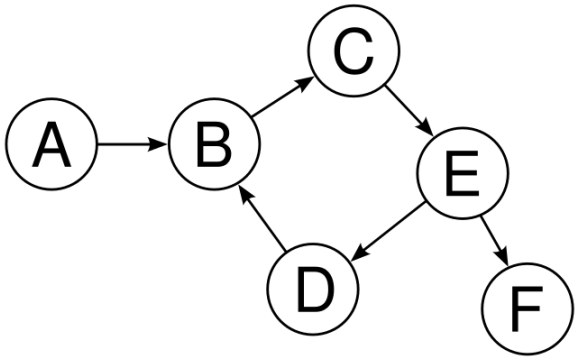


Figura 4. DAG. Wikimedia Commons. (Coetzee)

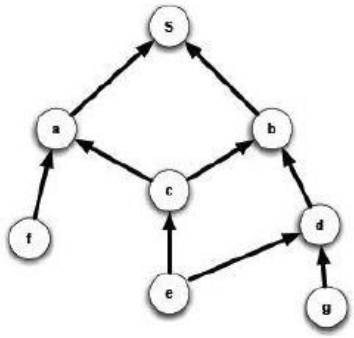


Figura 5. DODAG. Wikimedia Commons. (Khalili, 2016)

La selección de nodos vecinos se da a través de la asignación y comparación de rangos (ranks) definidos por una función objetivo (OF, por sus siglas en inglés), donde, generalmente, a mayor sea el rango, significa que está más abajo jerárquicamente en el DODAG, por lo tanto, la elección del nodo vecino será el que menor rango tenga, dado que el envío de información de un nodo al nodo raíz conllevaría un menor número de saltos.

El protocolo RPL cuenta con 4 tipos de mensajes para la formación y mantenimiento del DODAG:

1. DIO (DODAG Information object) objeto de información del DODAG.
2. DAO (Destination advertisement object) objeto de actualización al destino.
3. DAO-Ack (Destination advertisement object acknowledgment) objeto de confirmación de actualización al destino.
4. DIS (DODAG Information solicitation) solitud de información del DODAG.

DIS: El objeto de solicitud de información es el mensaje utilizado para solicitar, a nodos cercanos, información respecto al DODAG. Este tipo de mensaje es el que los nodos utilizan para el conocimiento de la existencia de un DODAG existente o, en caso de ser el nodo raíz, iniciar con la formación de un DODAG.

DIO: El objeto de información del DODAG es el mensaje utilizado por los nodos para transmitir la información respecto al estado del DODAG (en caso de estar vinculado a uno). Generalmente se usan para responder a un DIS de un nodo desconocido y para mantener actualizada la red de nodos.

DAO: El objeto de actualización al destino es el mensaje utilizado por los nodos para informar a su nodo padre sobre un cambio en la topología del DAG subsecuente. Este tipo de mensaje se puede dar, por ejemplo, al momento de que un nuevo nodo vecino se una al DODAG.

DAO-Ack: El objeto de confirmación de actualización al destino es el mensaje que se envía un nodo como respuesta, generalmente, a modo de confirmación de que la información se recibió correctamente y además fue aceptada.

El procedimiento que se lleva a cabo cuando un nuevo nodo es descubierto (o este descubre a una red de nodos ya formada) es el siguiente:

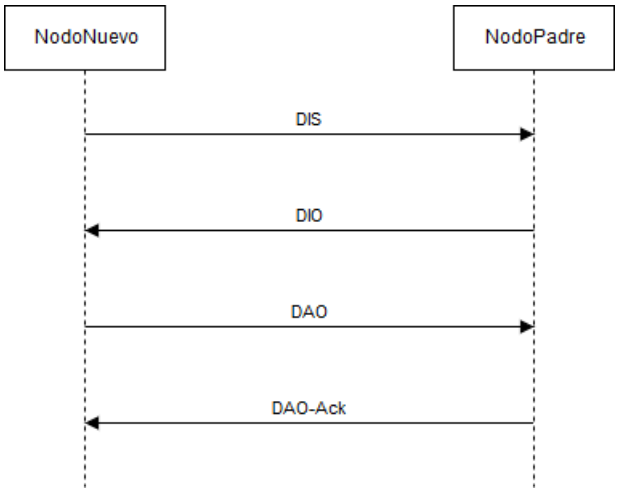


Figura 6. Comunicación entre nodos

1. El nuevo nodo (A) envía un DIS al nodo al cual quiere solicitarle la información (B).
2. El nodo B puede (o no) responder, a través de un DIO, el cual contiene toda la información necesaria sobre el DODAG.
3. El nodo A puede (o no) decidir unirse al DODAG, mediante un mensaje DAO, el cual informa al nodo B la petición de unirse al DODAG.
4. El nodo B, con base en la información del DAO del nodo A, envía un DAO-Ack informando si fue admitido o rechazado al nodo A.

El protocolo de encaminamiento RPL cuenta con dos posibles modos de ejecución, dependiendo de la implementación con la que se haya programado: Storing y No storing; La diferencia de estos modos radica en la forma en la que un paquete es encaminado de forma descendente (desde el nodo raíz hacia los nodos hijos).

El modo storing consiste en que cada nodo de la red almacena y mantiene su propia tabla de encaminamiento, mientras que en el modo no storing esto no ocurre, por lo tanto, el nodo raíz es el único que conoce y mantiene la tabla de encaminamientos, por lo tanto, conoce toda la topología del DODAG y todos los paquetes deben pasar a través de él para llegar al destino.

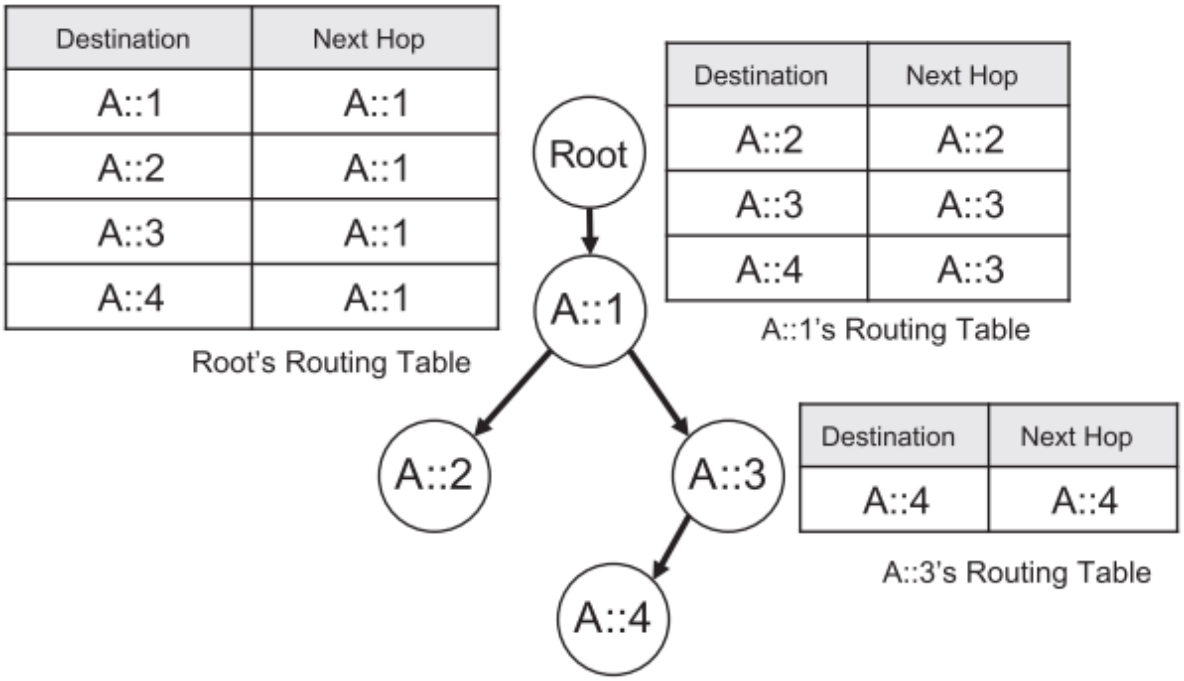


Figura 7. Tablas de rutas de cada nodo: modo storing. (Sukho, DongYeop, Kim, & Ki-Hyung, 2018)

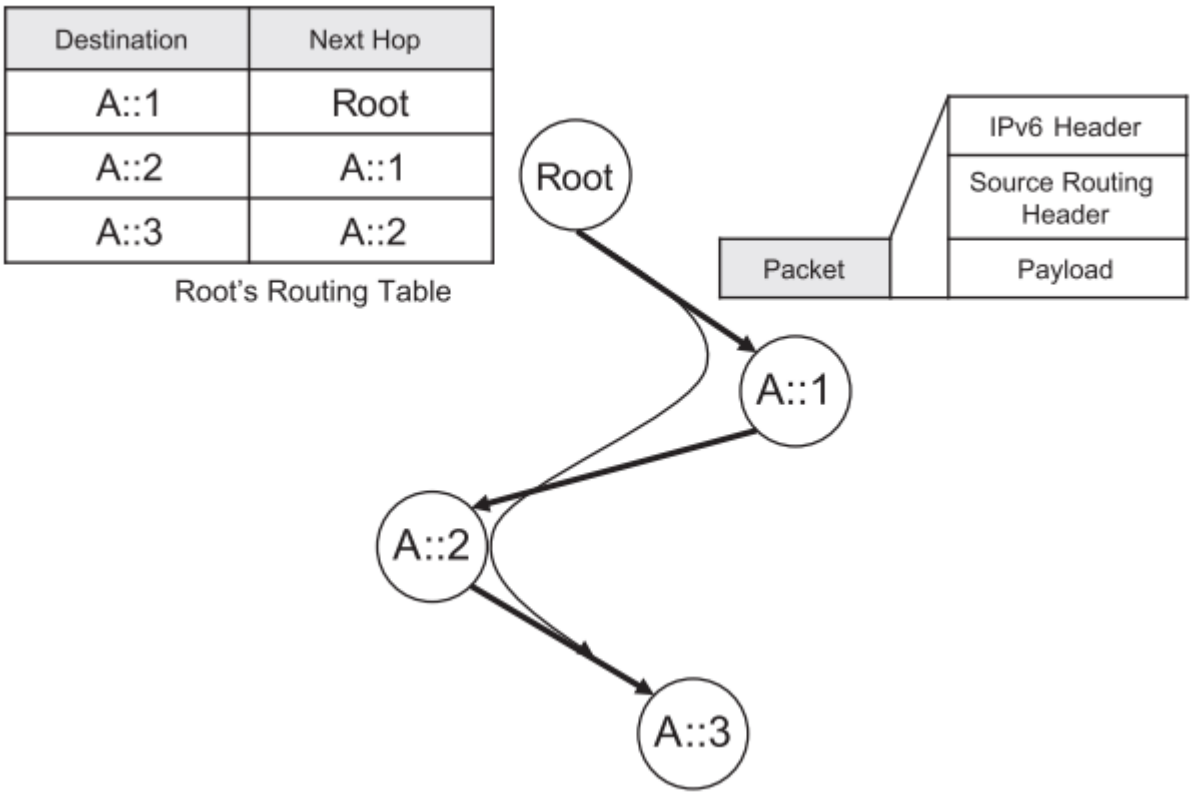


Figura 8. Tabla de rutas solo en el nodo raíz: modo no storing. (Sukho et al., 2018)

2.1.4 NMDA. NMDA, traduce a “arquitectura del datastore de gestión de red”, por sus siglas en inglés.

La IETF en el RFC 8342 (M. Björklund, Schoenwaelder, Shafer, Watsen, & Wilton, 2018) plantea una plataforma arquitectónica para almacenamiento, acceso y modelado de datos (definido en inglés como Datastores) que, generalmente, son usados por protocolos de gestión de redes, tales como el protocolo de configuración de red NETCONF (por sus siglas en inglés), RESTCONF, y el lenguaje de modelado YANG.

El objetivo de esta plataforma arquitectónica es identificar un conjunto de datastores conceptuales, útiles para el desarrollo de modelos YANG para protocolos de gestión de redes o, en este caso, para desarrollar un modelo YANG del protocolo RPL que luego sea implementado y evaluado sobre RESTCONF.

Cabe aclarar que este conjunto de datastores conceptuales brindan o sirven como punto de partida para que los modelos puedan ser escritos de forma tal que sean agnósticos al protocolo de gestión de red, es decir, para que su implementación pueda darse, por ejemplo, tanto en RESTCONF como en NETCONF. No obstante, no es obligatorio la adopción de todo el conjunto de datastores al momento de desarrollar un modelo YANG.

Los datastores pueden ser implementados, por ejemplo, mediante el uso de ficheros, una base de datos, o combinaciones de estos. Un datastore mapea una instancia del árbol de un modelo YANG, es decir, los datos contenidos en un datastore se ven accedidos y modificados a través de la interfaz proporcionada por el árbol de un modelo YANG, esto último es la forma jerárquica de ver la implementación de un modelo YANG. (M. Björklund et al., 2018)

2.1.5 Modelo de datos YANG para la NMDA. La IETF en el RFC 8349 (Lhotka, Lindem, & Qu, 2018) plantea tres (3) módulos y un (1) submódulo, escritos en YANG, que sirven como plataforma para cualquier extensión de la arquitectura de almacenamiento de datos para la administración de redes (NMDA, por sus siglas en inglés), en el ámbito de la configuración y administración de subsistemas de enrutamiento (Lhotka et al., 2018).

Los módulos incluidos actúan como núcleo para modelos de datos para el encaminamiento subsecuentes, dado que provee segmentos comunes para ese tipo de extensiones tales como: las rutas, la información de las rutas y protocolos de control.

El primer módulo, llamado “ietf-routing” (Ladislav, Lindem, & Qu, 2017), define los componentes genéricos de un sistema de enrutamiento. Los otros dos módulos, “ietf-ipv4-unicast-routing” e “ietf-ipv6-unicast-routing” corresponden a una extensión del módulo “ietf-routing” anteriormente mencionado, implementando nodos adicionales que son necesarios para el enrutamiento unicast IPv4 e IPv6, respectivamente.

El módulo “ietf-ipv6-unicast-routing” contiene el submódulo “ietf-ipv6-router-advertisements”, el cual, extiende al módulo “ietf-interfaces” e “ietf-ip”.

En la siguiente figura se presenta parte de la estructura de árbol generada a partir del módulo “ietf-routing”, el cual aumenta el módulo ietf-routing.

```
module: ietf-routing
  +--rw routing
  |  +--rw router-id?                yang:dotted-quad {router-id}?
  |  +--ro interfaces
  |  |  +--ro interface*   if:interface-ref
  |  +--rw control-plane-protocols
  |  |  +--rw control-plane-protocol* [type name]
  |  |     +--rw type             identityref
  |  |     +--rw name             string
  |  |     +--rw description?     string
  |  |     +--rw static-routes
  |  +--rw ribs
  |     +--rw rib* [name]
  |        +--rw name              string
  |        +--rw address-family    identityref
  |        +--ro default-rib?      boolean {multiple-ribs}?
  |        +--ro routes
  |        |  +--ro route* []
  |        |     +--ro route-preference?   route-preference
  |        |     +--ro next-hop
  |        |     |  +--ro (next-hop-options)
  |        |     |     +--:(simple-next-hop)
  |        |     |     |  +--ro outgoing-interface?   if:interface-ref
  |        |     |     +--:(special-next-hop)
  |        |     |     |  +--ro special-next-hop?     enumeration
  |        |     |     +--:(next-hop-list)
  |        |     |        +--ro next-hop-list
  |        |     |           +--ro next-hop* []
  |        |     |              +--ro outgoing-interface?   if:interface-ref
  |        |     +--ro source-protocol     identityref
  |        |     +--ro active?             empty
  |        |     +--ro last-updated?       yang:date-and-time
  |        +---x active-route
  |        |  +--ro output
  |        |     +--ro route
  |        |        +--ro next-hop
  |        |        |  +--ro (next-hop-options)
  |        |        |     +--:(simple-next-hop)
  |        |        |     |  +--ro outgoing-interface?   if:interface-ref
  |        |        |     +--:(special-next-hop)
  |        |        |     |  +--ro special-next-hop?     enumeration
  |        |        |     +--:(next-hop-list)
  |        |        |        +--ro next-hop-list
  |        |        |           +--ro next-hop* []
  |        |        |              +--ro outgoing-interface?   if:interface-ref
  |        |        +--ro source-protocol    identityref
  |        |        +--ro active?            empty
  |        |        +--ro last-updated?      yang:date-and-time
  |        +--rw description?      string
  o--ro routing-state
```

Figura 9. Estructura parcial de árbol del modelo YANG ietf-routing.

A continuación, el código YANG del agrupamiento de datos respecto a los metadatos de rutas, almacenadas en el contenedor “active-route”.

```
310    grouping route-metadata {
311      description
312        "Common route metadata.";
313      leaf source-protocol {
314        type identityref {
315          base routing-protocol;
316        }
317        mandatory "true";
318        description
319          "Type of the routing protocol from which the route
320           originated.";
321      }
322      leaf active {
323        type empty;
324        description
325          "Presence of this leaf indicates that the route is preferred
326           among all routes in the same RIB that have the same
327           destination prefix.";
328      }
329      leaf last-updated {
330        type yang:date-and-time;
331        description
332          "Time stamp of the last modification of the route.  If the
333           route was never modified, it is the time when the route was
334           inserted into the RIB.";
335      }
336    }
```

Figura 10. Maquetado en YANG de los metadatos de una ruta.

2.1.6 Biblioteca YANG. En el constante desarrollo e implementación de las herramientas necesarias en un futuro cercano para el mantenimiento y automatización de las redes, con el surgimiento de roles y disciplinas tales como el Network DevOps, es de vital importancia definir los estándares y bases sobre las cuales se construirán los nuevos conceptos y métodos para el correcto desempeño de estos. Debido a esto, la IETF en el RFC 8525 (Andy Bierman, Björklund, Schönwälder, Watsen, & Wilton, 2019) define el concepto llamado Biblioteca YANG.

La Biblioteca YANG tiene como objetivo general suplir la necesidad de tener un mecanismo estándar para exponer los módulos YANG y Datastores que se estén usando en un servidor para la gestión de redes. (Andy Bierman et al., 2019)

Hoy en día, en el misma RFC (8525) se presenta un módulo YANG llamado “ietf-yang-library”, este módulo es compatible con el previamente definido NMDA.

2.1.7 Arquitectura cliente-servidor. La arquitectura cliente-servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores y los demandantes, llamados clientes. (Palacios & Rodriguez, 2019)

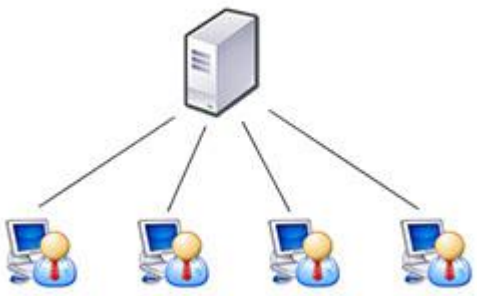


Figura 11. Arquitectura cliente-servidor

En el presente proyecto se implementa esta arquitectura, donde el servidor corresponde a la distribución de Linux Ubuntu 16.04, alojado dentro de una máquina virtual. El cliente corresponde

a una consola Cygwin en la máquina anfitrión, aunque también podría ser cualquier otro dispositivo con capacidad de generar peticiones al servidor usando como protocolo de aplicación a HTTP versión 2.

2.2 Herramientas implementadas

En la presente sección, se presentarán las herramientas principales, necesarias e implementadas, para el desarrollo del proyecto.

2.2.1 Pyang. Pyang es la herramienta principal para la validación sintáctica (la información detallada de esto se encuentra en la sección de 3.3 de este documento) de los módulos desarrollados, además de proveer la utilidad de generar los diagramas de árbol jerárquicos que resultan a partir del código YANG escrito o de transformar modelos YANG a modelos en otro formato (Martin Björklund, n.d.). Además, permite agregar plugins que mejoren o agreguen funcionalidades nuevas o existentes, un ejemplo de plugin es Pyangbind, un plugin que permite transformar módulos YANG en clases de Python (Shakir, n.d.).

En la figura de a continuación, se presenta el diagrama de árbol del módulo “uis-ofs-of0”, junto con el comando necesario para la generación de este.

```
douglas@restconf-svr:~/Backend/Modulos-yang$ pyang -f tree uis-ofs-of0@2020-03-0
4.yang

module: uis-ofs-of0
  augment /rt:routing/rt:control-plane-protocols/rt:control-plane-protocol/rpl:r
pl/rpl:objective-functions/rpl:objective-function:
    +--rw of0
       +--ro variables
       |  +--ro step-of-rank?    uint16
       |  +--ro rank-increase?   uint16
       +--rw parameters
          +--rw stretch-of-rank?   uint16
          +--rw rank-factor?       uint16
douglas@restconf-svr:~/Backend/Modulos-yang$
```

Figura 12. Comando Pyang y resultado.

2.2.2 Yangson. Yangson es la herramienta principal para la validación de implementación y validación de datos de un datastore junto con una instancia de biblioteca YANG, es decir, para realizar la validación de implementación y datos de forma simultánea, se requiere disponer de un datastore, y de una instancia válida del modelo ietf-yang-library. No obstante, es posible realizar la validación de implementación por separado a la validación de datos.

Para el desarrollo del proyecto, se hizo uso de ambas validaciones, tanto de forma separada como de forma simultánea.

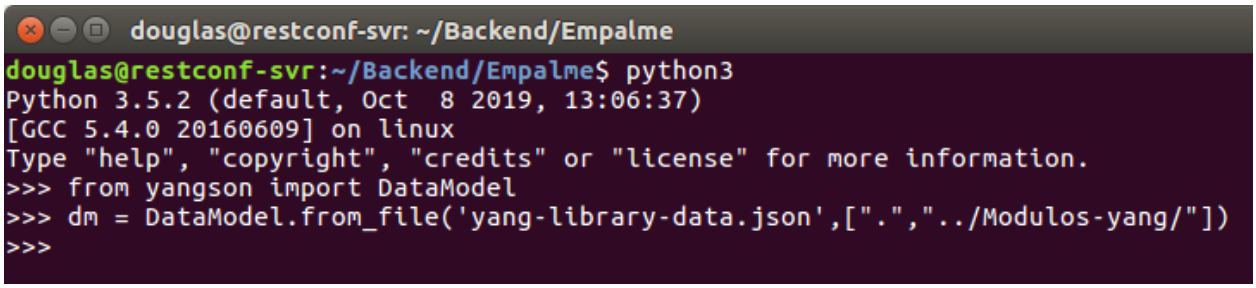


Figura 13. Validación de implementación independiente exitosa.

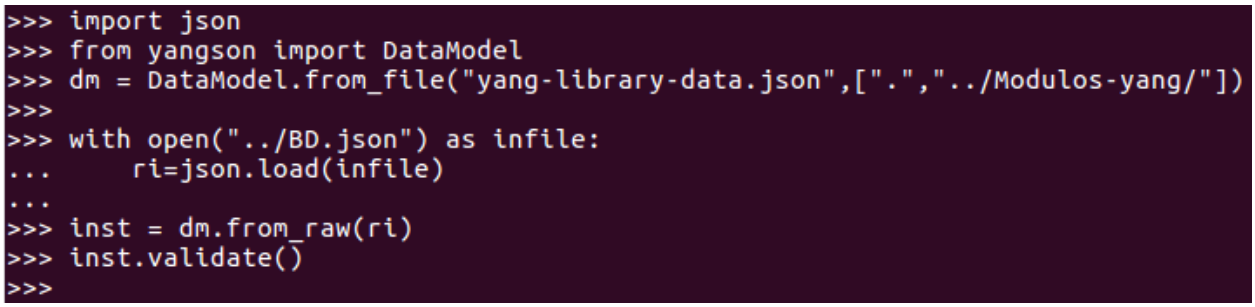


Figura 14. Validación de datos e implementación.

El funcionamiento de esta herramienta, dicho de forma general, consiste en analizar gramaticalmente (parse, en inglés) un modelo YANG y un datastore en formato Json, para luego encajar los datos presentes en el datastore dentro del modelo YANG. En caso de que algún dato

dentro del datastore no presente las características estipuladas por el modelo YANG, se arroja un error indicando el problema.

2.2.3 Oracle VM Virtual Box. Oracle VM VirtualBox (conocido generalmente como VirtualBox) es un software de virtualización para arquitecturas x86/amd64. Actualmente es desarrollado por Oracle Corporation como parte de su familia de productos de virtualización. Por medio de esta aplicación es posible instalar sistemas operativos adicionales, conocidos como "sistemas invitados", dentro de otro sistema operativo "anfitrión", cada uno con su propio ambiente virtual. Entre los sistemas operativos soportados (en modo anfitrión) se encuentran GNU/Linux, Mac OS X, OS/2 Warp, Genode, Windows y Solaris/OpenSolaris, y dentro de ellos es posible virtualizar los sistemas operativos FreeBSD, GNU/Linux, OpenBSD, OS/2 Warp, Windows, Solaris, MS-DOS, Genode y muchos otros (Wikipedia, 2020).

En el presente proyecto se implementa esta herramienta mediante la virtualización del sistema operativo Ubuntu 16.04, una de las distribuciones de Linux. Esta máquina virtual tiene las siguientes características:

- RAM: 3GB
- Almacenamiento: 12GB
- NIC: Adaptador puente

Es importante que la NIC esté configurada en modo adaptador puente. Esto hace que Virtual Box configure la virtualización del adaptador de red usando otra instancia del adaptador de red de la máquina anfitriona, de esta forma, la máquina virtual toma una dirección IP asignada por la misma red local en la que se encuentra la máquina anfitriona, permitiendo una fácil, rápida y efectiva comunicación entre ambas máquinas. Ideal para implementar la arquitectura de cliente

servidor, en especial porque cualquier otro dispositivo conectado o presente dentro de la misma red local, también se podría conectar al servidor como cliente.

En la siguiente figura, se muestra en ejecución la máquina virtual, junto con la dirección IP privada asignada en ese momento y la dirección IP asignada a la máquina anfitrión.

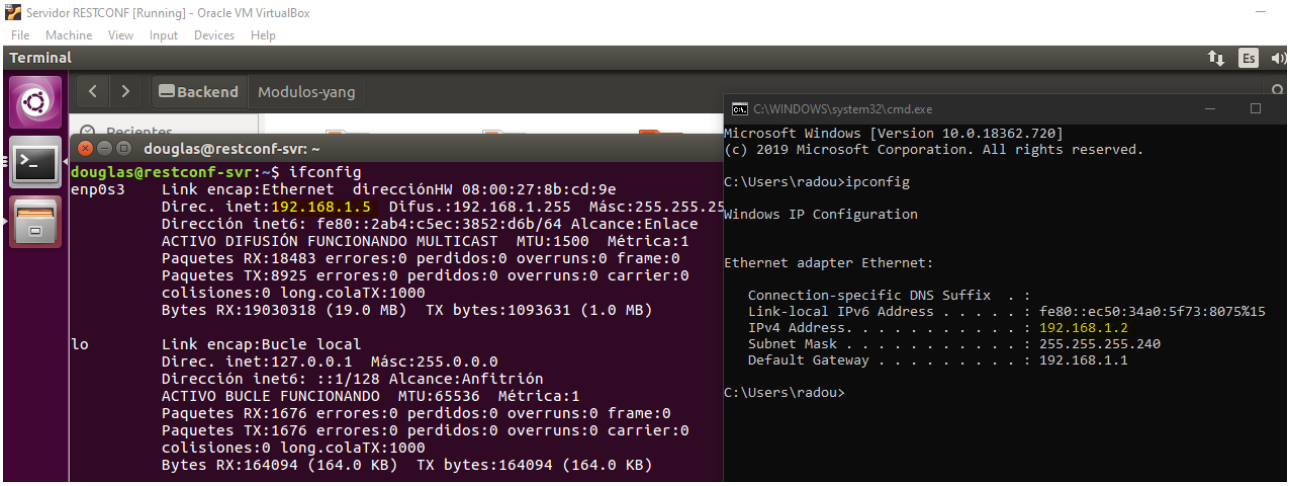


Figura 15. Direcciones IP del servidor (izquierda) y el cliente (derecha)

2.2.4 Cygwin. Cygwin es una colección de herramientas desarrollada por Cygnus Solutions para proporcionar un comportamiento similar a los sistemas Unix en Microsoft Windows. Su objetivo es portar software que ejecuta en sistemas POSIX a Windows con una recompilación a partir de sus fuentes (Noer, 1998).

En el presente proyecto, se utiliza Cygwin gracias a la posibilidad de implementar una consola que tenga la capacidad de ejecutar software hecho originalmente para sistemas basados en Unix. De esta forma, se facilita la instalación de la herramienta Curl en dispositivos cuyo sistema operativo sea Windows. Esto es ventajoso debido a que Curl es la herramienta que se usa para hacer las peticiones al servidor, es decir, es la herramienta que proporciona las utilidades necesarias para que un dispositivo pueda ser usado como cliente.

2.2.5 Curl. Curl es un proyecto de software consistente en una biblioteca (libcurl) y un intérprete de comandos (curl) orientado a la transferencia de archivos. Soporta los protocolos FTP, FTPS, HTTP, HTTPS, TFTP, SCP, SFTP, Telnet, DICT, FILE y LDAP, entre otros (Stenberg, 2017).

La implementación de esta herramienta es crucial para la creación de peticiones que luego son enviadas al servidor, con el fin de realizar y monitorizar las pruebas pertinentes al comportamiento del servidor y, en concreto y con especial énfasis, para ver el funcionamiento del modelo YANG del protocolo RPL sobre el protocolo RESTCONF. En la figura de a continuación se muestra una petición hecha con Curl al servidor, donde este retorna la información solicitada, en una consola Cygwin.

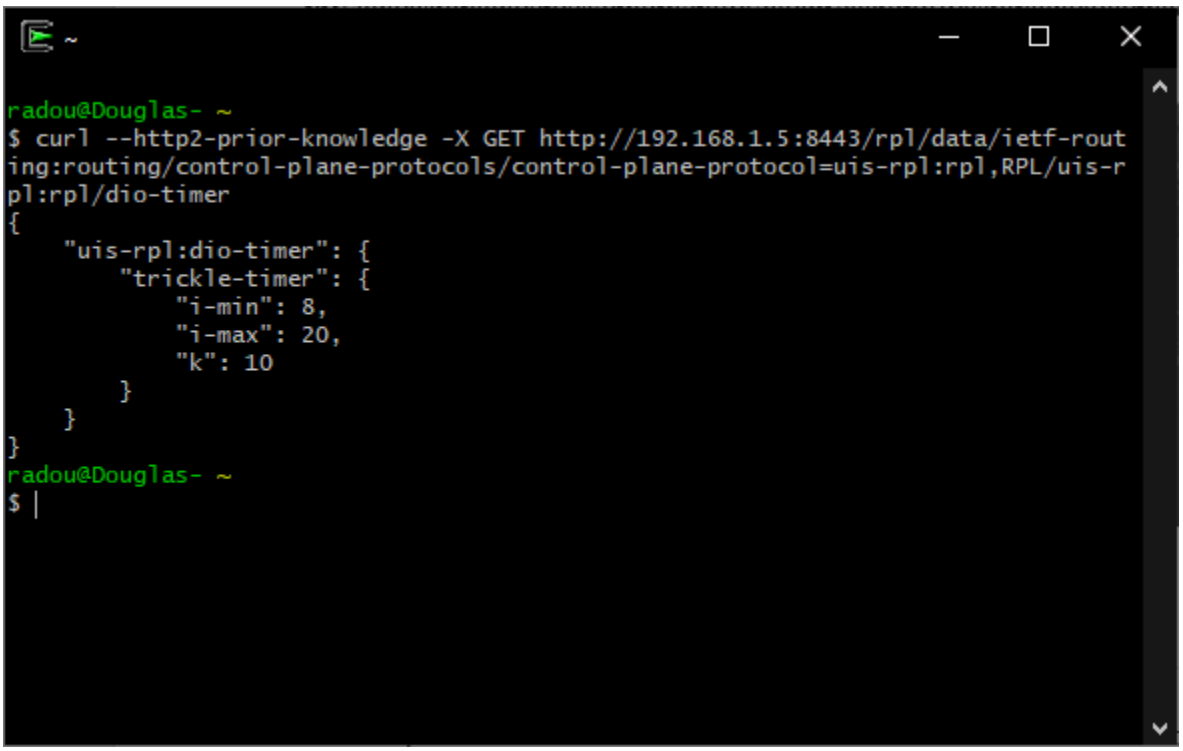


Figura 16. Petición construida con Curl y respuesta del servidor

2.2.6 Jetconf. Jetconf es una implementación del protocolo RESTCONF escrita en Python3 (NIC.CZ, 2019). Es la principal herramienta que se usa en el presente proyecto, dado que proporciona la implementación de RESTCONF junto con una API para el incruste de backends personalizados, ideal para el desarrollo de tecnologías emergentes que hagan uso del protocolo RESTCONF, tal como se ve en la sección 3.4 de este documento.

La API que proporciona Jetconf está diseñada para cubrir cualquiera de los casos de uso en los que sea necesario la implementación del protocolo RESTCONF en un servidor cuyo rol sea el de servidor de gestión de red. Por tanto, las aplicaciones que se requieran desarrollar usando Jetconf, deben hacer uso de la API proporcionada, a estas aplicaciones se les denomina Backends de Jetconf.

Todo backend de Jetconf cumple con la arquitectura planteada por la API de este, dividiendo las funciones necesarias para la gestión de un datastore y de las operaciones y acciones RESTCONF en módulos por separado, con un nombre en común. Adicionalmente, si un backend de Jetconf requiere datos o módulos extra, se pueden agregar a dicho backend, siempre y cuando respete la API.

La API requiere que la implementación de un backend de Jetconf cumpla con los siguientes módulos, junto con su respectiva función:

- usr_conf_data_handlers (Manejador de los datos de configuración)
- usr_state_data_handlers (Manejador de los datos de estado)
- usr_op_handlers (Manejador para las operaciones RESTCONF - RPCs)
- us_action_handlers (Manejador de las acciones RESTCONF – operaciones de un nodo)
- usr_datastore (Personalización del funcionamiento al guardar o cargar datastores)
- usr_init (Operaciones a realizar al momento de iniciar o finalizar el servidor Jetconf)

En la siguiente figura se puede ver el código para el RPC “get-route-stack” del módulo “uis-rpl”, dentro del módulo Python usr_op_handlers de un backend de Jetconf.

usr_op_handlers.py

```
from yangson.instance import InstanceRoute
from jetconf.helpers import JsonNodeT
from jetconf.data import BaseDatastore

from . import sim_inst_rpl
from colorlog import info

RPL = sim_inst_rpl.RPL
RUTA_RPL = "/ietf-routing:routing/control-plane-protocols/" + \
            "control-plane-protocol/uis-rpl:rpl"

class OpHandlersContainer:
    def __init__(self, ds: BaseDatastore):
        self.ds = ds

    def RPl_get_route_stack(self, input_args: JsonNodeT, username: str) -> JsonNodeT:
        def busqueda_recursiva(stack, destino):
            for ip in RPL['TOPOLOGIA']:
                if ip == destino:
                    stack.append(ip)
                    busqueda_recursiva(stack,RPL['TOPOLOGIA'][ip]['PADRE'])
                    break
        info('Generando stack de enrutamiento...')
        stack = []
        target = input_args["uis-rpl:target-ipv6-address"]
        busqueda_recursiva(stack,target)
        stack.reverse()
        info('Enviando...')
        return stack
```

Figura 17. Código escrito en Python para el RPC "get_route_stack".

2.2.7 Pyangbind. Pyangbind es un plugin para Pyang que genera una jerarquía de clases de Python a partir de uno o varios módulos YANG. Las clases resultantes pueden ser usadas e interactuadas directamente con Python. Esto permite la evaluación de segmentos de modelos YANG específicos o módulos YANG completos (Shakir, n.d.). Particularmente, Pyangbind brinda las siguientes funciones:

- Crear nuevas instancias de datos.
- Cargar instancias de datos desde fuentes externas.
- Serializar objetos de tal forma que puedan ser almacenados.

2.3 Antecedentes

En el estado del arte, dada la base para el modelado de datos para la administración de enrutamiento proporcionado por la IETF, en el campo de las redes LLN, existe un modelo en YANG para el protocolo de enrutamiento RIP (Protocolo de información de enrutamiento, por sus siglas en inglés) presentado como un borrador ante la IETF el 4 de febrero de 2018. Este borrador, llamado “A YANG Data Model for Routing Information Protocol (RIP)”, expiró el 8 de agosto de 2018, sin embargo, dejó a disposición general el modelo en YANG del protocolo del enrutamiento RIP, el cual, sirve como guía para la correcta extensión y desarrollo para el modelado de datos para la administración de enrutamiento, siguiendo el estándar planteado por la IETF en el RFC 8349.

En la siguiente figura se presentan dos atributos globales del modelo YANG, “ietf-rip”, del protocolo RIP: “default-metric” y “distance” (Liu, Sarda, & Choudhary, 2018).

```
480    grouping global-attributes {
481      description
482        "Global configuration and state attributes.";
483      uses originate-default-route-container;
484
485      leaf default-metric {
486        type uint8 {
487          range 0..16;
488        }
489        default 1;
490        description
491          "Set the default metric.";
492      }
493
494      leaf distance {
495        type uint8 {
496          range 1..255;
497        }
498        default 120;
499        description
500          "The administrative distance of the RIPv2 or RIPng for the
501           current RIPv2 or RIPng instance.";
502      }
```

Figura 18. Atributos globales métrica por defecto y distancia en el modelo ietf-rip

El modelo YANG para el protocolo de enrutamiento RIP está diseñado para funcionar con los protocolos NETCONF y RESTCONF. Este modelo es de gran utilidad para comprender la integración con el módulo “ietf-routing”, con el fin de ajustarse al estándar propuesto de la NMDA.

3. Metodología

Para el desarrollo del presente proyecto, la metodología a seguir va de acuerdo con los dos componentes principales del proyecto: el desarrollo del modelo YANG del protocolo RPL y la implementación del servidor RESTCONF; Posterior a la finalización de los componentes, se

diseñan las pruebas pertinentes, las cuales evalúan simultáneamente el modelo YANG y la implementación del servidor. En la siguiente figura, se presenta la metodología:

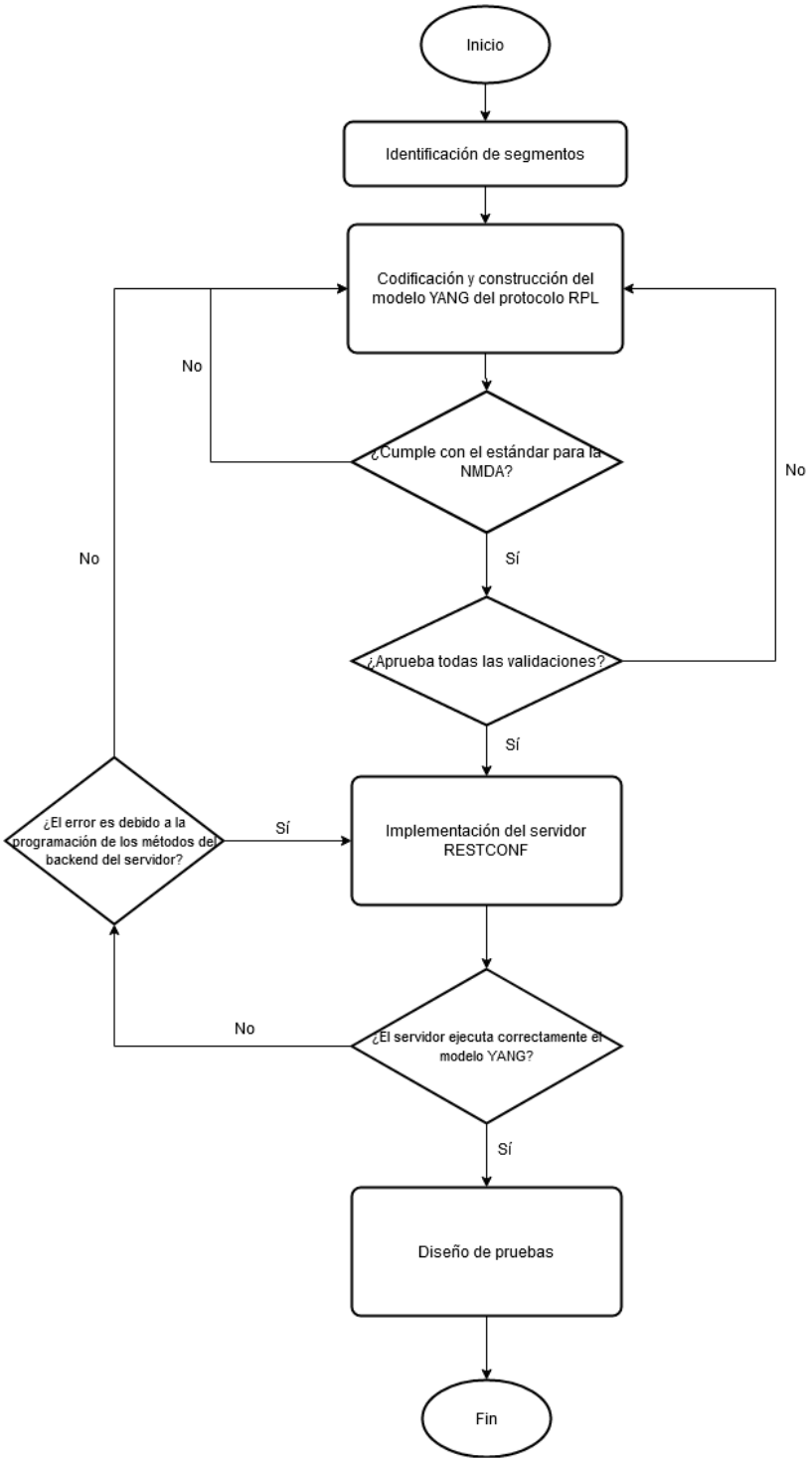


Figura 19. Diagrama de flujo de la metodología.

3.1 Segmentos del protocolo RPL modelados

Debido a la complejidad cada día creciente del protocolo RPL, fue necesario acotar la cobertura del modelo YANG de este protocolo que se iban a modelar en el presente estudio, de tal forma que este pueda ser expandido, aumentado o continuado por terceras partes, tomando como base el modelo YANG resultante de este proyecto, es por esto por lo que una necesidad imperativa fue definir la unidad de trabajo a modelar en YANG. A esta unidad conceptual se le asignó el nombre de “segmento”.

Segmento, bajo la definición conceptual dada para el presente proyecto, es la de un fragmento abstracto del funcionamiento, en este caso, de un protocolo de encaminamiento, es decir, corresponde a un trozo lógico que es independiente de la implementación de algoritmos, código o datos propios del protocolo de encaminamiento.

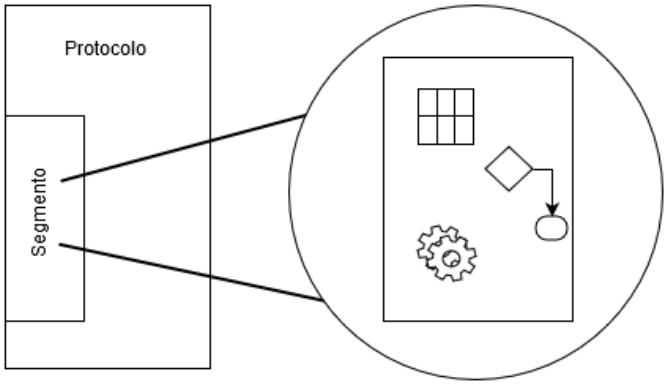


Figura 20. Segmento como conjunto abstracto de funcionalidades de un protocolo.

Para el desarrollo del presente proyecto se estipularon un total de once (11) segmentos del protocolo de encaminamiento RPL, de los cuales seis (6) de estos corresponden a segmentos relevantes y propios del anteriormente mencionado protocolo de encaminamiento. Los cinco (5) segmentos restantes corresponden a segmentos de carácter general para todo protocolo de encaminamiento. La elección de segmentos se realizó siguiendo el siguiente procedimiento:

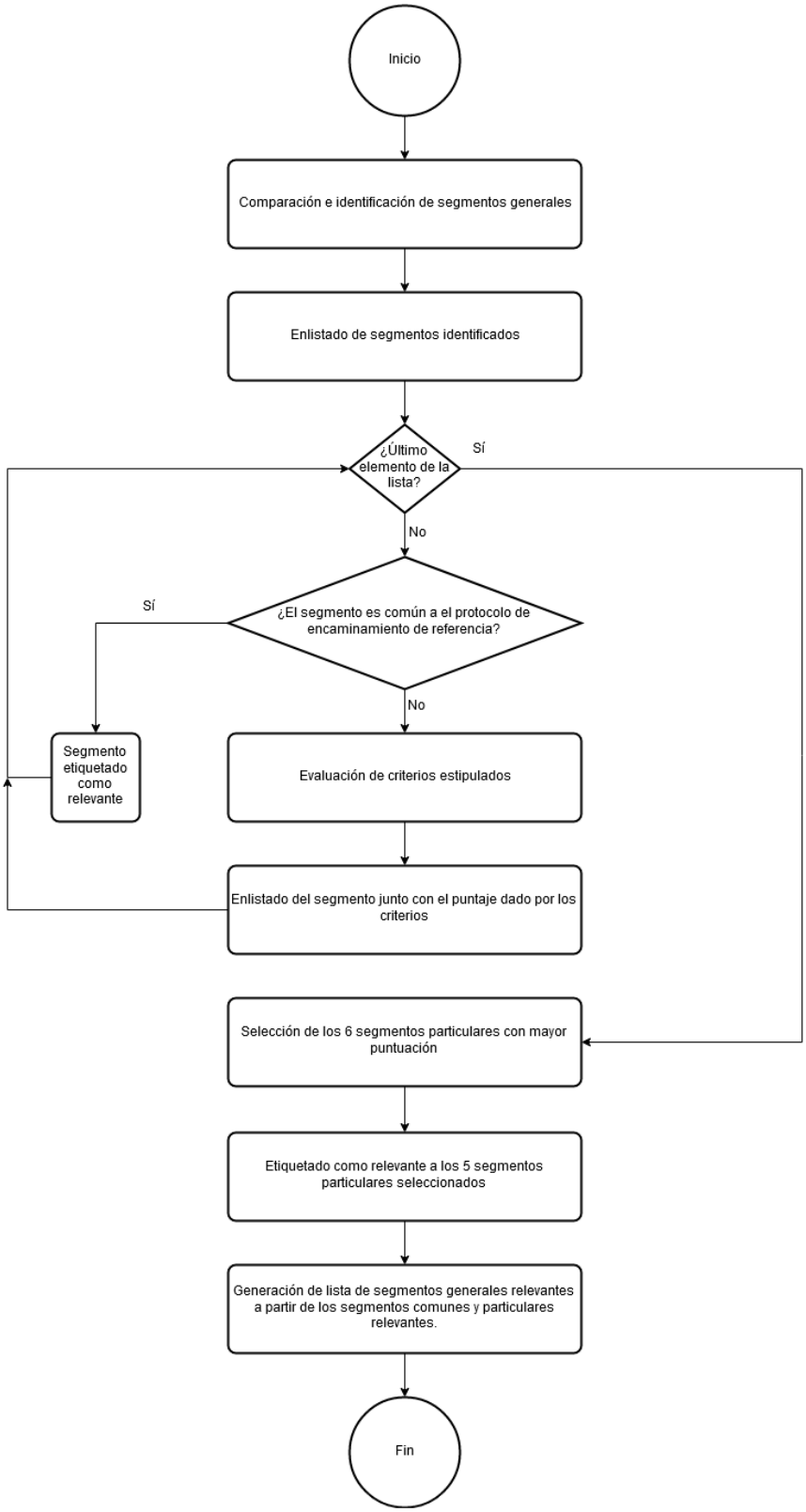


Figura 21. Diagrama de flujo del procedimiento de selección de segmentos relevantes.

Donde los criterios estipulados a seguir para la elección son los de a continuación:

- **Importancia del segmento como dependencia de otros segmentos:** Necesidad de la existencia previa del segmento para el correcto funcionamiento e implementación de segmentos derivados de este.
- **Papel desempeñado para el funcionamiento del protocolo RPL:** Relevancia del rol o tarea a desarrollar por el segmento para el funcionamiento general del protocolo RPL.
- **Frecuencia de las solicitudes al contenido del segmento:** Necesidad independiente de segmentos alternos al contenido del segmento particular.
- **Almacenamiento de datos generales del protocolo RPL:** El segmento almacena datos generales de uso, monitoreo, seguridad, estado, funcionamiento, etc. de forma medianamente prolongada.

Los segmentos de carácter general son los siguientes:

- Tabla de rutas
- Tabla de vecinos
- Direccionamiento IP
- Métrica definida
- Estadísticas

Los segmentos propios (y relevantes) del protocolo RPL son los siguientes:

- Función objetivo
- Construcción de topologías
- Trickle
- Enrutamiento descendente

- Tabla de padres elegidos
- Enrutamiento ascendente
- Interfaces

El segmento Interfaces fue añadido posteriormente a la selección de segmentos a modelar, debido a que la información que este contiene. presentaba características suficientes para separarlo del segmento donde se iba a incluir originalmente (tabla de rutas). Este segmento está encargado de contener la información relacionada con las interfaces (tanto físicas como lógicas) de un enrutador, habilitadas para funcionar con el protocolo RPL.

3.1.1 Segmentos compuestos. Durante el desarrollo del proyecto, al momento de modelar los segmentos estipulados, se dio lugar a que algunos de los segmentos anteriormente listados, debido a su fuerte interrelación o entrelazamiento, fuesen compuestos parcial o totalmente por otros segmentos. A esta situación se le nombró “composición de segmento” y al segmento donde se aplicaba esta situación como “segmento compuesto”. Esto mismo dio lugar a que se pudiese definir otro concepto, llamado “segmento independiente”, esto viene siendo un segmento cuya naturaleza o características le impide ser un segmento compuesto, es decir, el contenido de estos es atómico.

3.2 Modelado YANG

Para un correcto modelo YANG del protocolo de encaminamiento RPL se siguieron unos lineamientos y especificaciones estipuladas en el RFC 8407 para el desarrollo de módulos YANG donde se maximice la interoperabilidad y usabilidad a través de RESTCONF (Andy Bierman, 2018).

A partir de los segmentos enlistados en la sección anterior, se inició en primera instancia la identificación de segmentos compuestos debido a que a estos es posible aplicar la técnica de divide

y vencerás, logrando así obtener los segmentos independientes. Este procedimiento se realizó teniendo en cuenta que el objetivo final, en donde se implementaría el módulo YANG, sería un nodo raíz en modo **NO** storing, o en su defecto, un enrutador de frontera que tenga comunicación con un nodo raíz de una red que haga uso del protocolo de encaminamiento RPL.

Una vez identificado el tipo de segmentos de todos los segmentos a modelar, se procedió a desarrollar cada uno de ellos, empezando por la construcción básica del esqueleto de todo módulo YANG, procediendo a hacer el aumento al módulo “ietf-routing”, indicando cada contenedor de los segmentos compuestos y finalmente desarrollando cada uno de los segmentos independientes, que, en conjunto, conforman los segmentos compuestos.

El conjunto de segmentos compuestos dentro del contenedor principal después de la extensión del modelo ietf-routing corresponden al modelo del protocolo RPL, cumpliendo con el estándar propuesto de la NDMA.

Posteriormente, con el modelo del protocolo RPL desarrollado, es posible plantear las llamadas de operaciones remotas (RPC, por sus siglas en inglés). En el presente documento se muestra la construcción de un RPC en la sección 3.2.3.

3.2.1 Construcción de los segmentos independientes. A continuación, la lista de segmentos de tipo independiente:

- Trickle
- Estadísticas
- Interfaces
- Función objetivo
- Métricas

Cada uno de estos segmentos fue modelado de acuerdo con las especificaciones mostradas en el RFC 6550, correspondiente a la petición de comentarios del protocolo RPL y usando como apoyo el capítulo 17 del libro Interconnecting Smart Objects with IP (Dunkels & Vasseur, 2010).

Para el segmento de función objetivo se implementó el patrón de diseño de software de fábrica abstracta. Este consiste en proveer una interfaz común, en este caso, para crear funciones objetivo, que no necesariamente tienen relación entre sí, pero cumplen con las características para enlistarse dentro del modelo YANG del protocolo RPL, además de cumplir con el estándar propuesto de NMDA. Esto se logra a partir de la interfaz propiciada, que consiste en aumentar el módulo YANG del protocolo RPL, similar a como el módulo del protocolo de encaminamiento en sí aumenta el módulo de ietf-routing.

El segmento independiente de métricas fue el único segmento que no se modeló de acuerdo al RFC debido a que el protocolo RPL permite tener más de una métrica en una implementación, donde cada una cambia el algoritmo o fórmula matemática que la genera, a diferencia del segmento de función objetivo, donde cada función objetivo maneja sus propias variables y parámetros claramente diferenciados pero con una interfaz común, por tanto, se planteó una arquitectura de patrón de fachada, donde a través del datastore presente en el dispositivo, se establece una hoja (leaf) a modo de interfaz, donde el valor de la hoja representa el valor retornado por la métrica en ejecución.

3.2.2 Segmentos compuestos. La tabla 1 muestra los segmentos compuestos, junto con los segmentos que lo componen:

Tabla 1.

Segmentos compuestos y su estructura.

Segmentos compuestos	Segmentos que lo componen
Encaminamiento descendente	Tabla de rutas
Encaminamiento ascendente	Tabla de padres elegidos, tabla de vecinos, construcción de topologías
Direccionamiento IP	Encaminamiento descendente, Encaminamiento ascendente

Los segmentos independientes que no se encuentren dentro de un segmento compuesto son segmentos que están en el contenedor raíz del módulo YANG de RPL, es decir, no componen a ningún segmento más que al módulo principal.

Como se puede ver en la tabla, el segmento de direccionamiento IP está compuesto de otros segmentos compuestos. En este caso de los segmentos de encaminamiento ascendente y descendente.

3.2.3 RPC. Una llamada a procedimientos remotos (RPC por sus siglas en inglés) es un programa o instrucción que utiliza una computadora para ejecutar código en una máquina remota sin tener que preocuparse por las comunicaciones entre ambas (Birrell & Nelson, 1984).

En el presente proyecto, se desarrolló un RPC que permite obtener el conjunto de direcciones IP a seguir para enviar un paquete a un nodo, desde el nodo raíz (debido a que el modelo YANG del protocolo RPL se está haciendo para el modo **no** storing).

El conjunto de direcciones IP corresponde a una lista de direcciones IP, cuyo orden importa, en el que se presenta utilizando el sistema LIFO (last input, first output), es decir, el último elemento se presenta como el primero, el penúltimo como el segundo y de forma sucesiva con los demás

elementos. Para lograr esto, al generar la lista de direcciones IP, esta se invierte y posteriormente es enviada al cliente.

Como entrada (input) requiere la dirección IP del nodo destino. A partir de esta dirección, se hace una búsqueda recursiva, apilando las direcciones IP de todos los saltos (si existen) para llegar hasta el nodo destino, teniendo siempre como punto de partida al nodo raíz.

Se retorna un listado de hojas (leaf-list) de cada dirección IP hasta el destino, en el orden correspondiente.

Este RPC recibe el nombre de “get_route_stack”. La figura 15 muestra el código fuente de este método.

3.2.4 Cumplimiento con el estándar para la NMDA. Como se explica en la sección 2.1.4, la IETF brinda un módulo YANG llamado ietf-routing, el cual, todo módulo YANG de un protocolo de encaminamiento debe de extender para pertenecer a los protocolos con plano de control en un dispositivo que implemente la NMDA. En la figura de a continuación se presenta el árbol jerárquico del módulo ietf-routing sin la extensión del módulo uis-rpl.

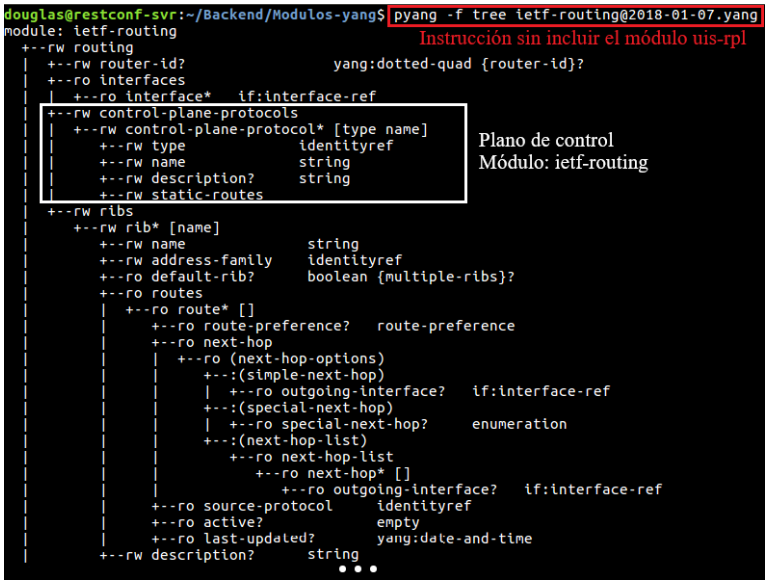


Figura 22. Árbol jerárquico del módulo ietf-routing.

Una vez ensamblados los segmentos para la construcción completa del contenedor principal del protocolo RPL, en este caso llamado rpl, se añade este contenedor a la extensión del módulo ietf-routing, para cumplir con el requisito de proveer la capacidad de mantener varios protocolos de encaminamiento en el plano de control para estos, propuesto por la IETF, en un dispositivo de gestión de red, por ejemplo, un servidor.

```
augment "/rt:routing/rt:control-plane-protocols/"
  + "rt:control-plane-protocol"{

    when "derived-from-or-self(rt:type, 'rpl:rpl')"{
      description
        "This augment is only valid for a routing protocol instance
        of RPL.";
    }

    description "RPL augmentation.";
    container rpl {
      description "RPL data.";
```

Figura 23. Instrucción para la extensión del módulo ietf-routing.

Adicionalmente, los datos de estado y de configuración no están separados los unos de los otros, es decir, no hay duplicación de contenedores o estructuras para discernir entre estos datos. En la figura de a continuación se puede evidenciar el cumplimiento del estándar propuesto por la IETF.

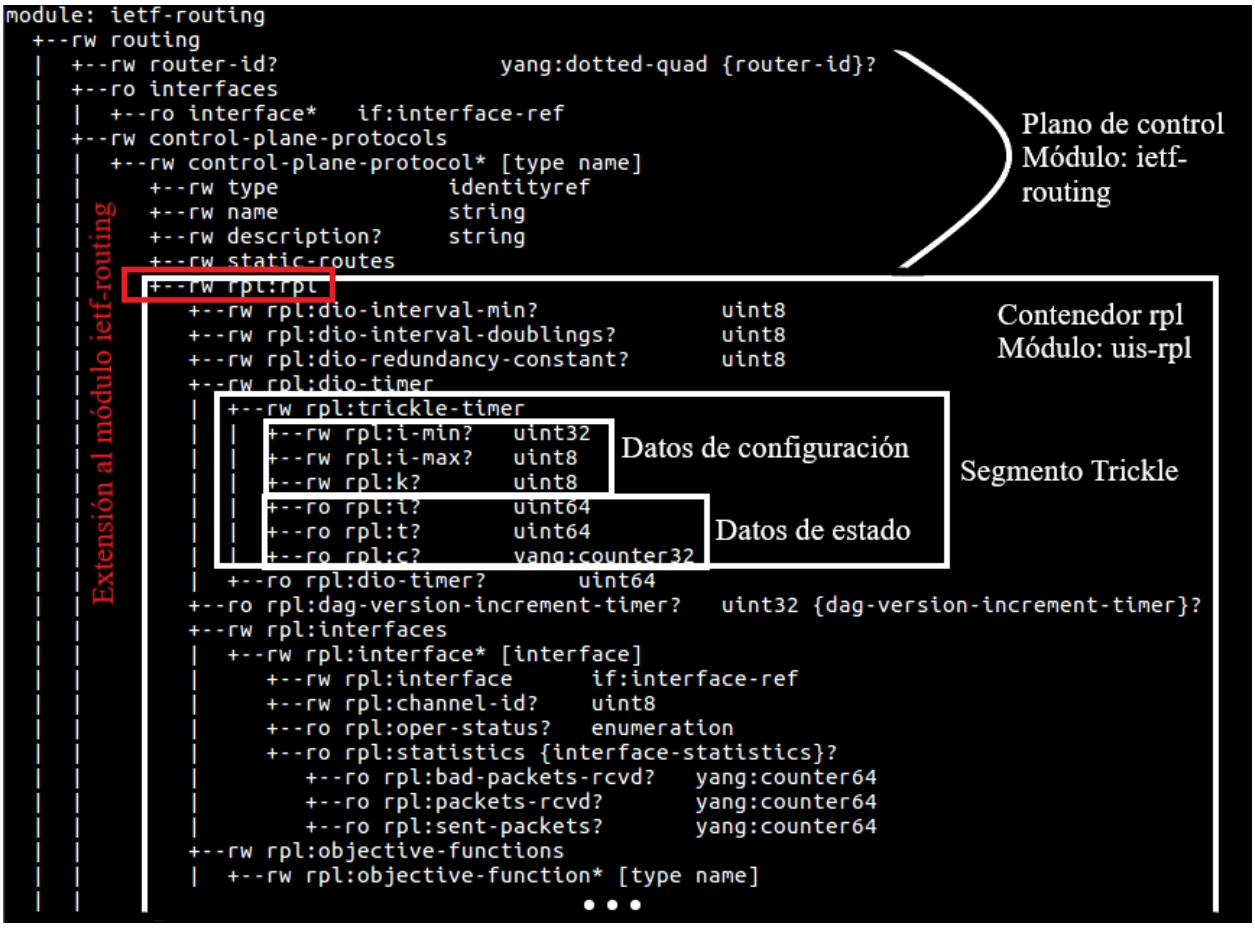


Figura 24. Sección del árbol de datos jerárquico generado con Pyang.

3.3 Validación del modelo YANG

Al modelo YANG del protocolo RPL durante todo su desarrollo se le aplican tres (3) tipos de validaciones diferentes, cada una de ellas tiene un propósito específico y se deben realizar en orden.

3.3.1 Validación de sintaxis. La validación de sintaxis consiste en el análisis sintáctico del código YANG escrito, con el fin de buscar los errores de escritura, de sintaxis o de los XPath estipulados.

Para realizar esta validación se usa la herramienta Pyang. Esta validación se puede realizar de dos formas, realizando una validación de sintaxis simple o mediante la construcción de un árbol

jerárquico a partir del código del módulo YANG. Esta última opción es, generalmente, más útil debido a que permite visualizar los cambios realizados sobre el módulo, respecto a la adición de segmentos, de forma visual y directa. *Ver figura 12, en la sección 2.2.1.*

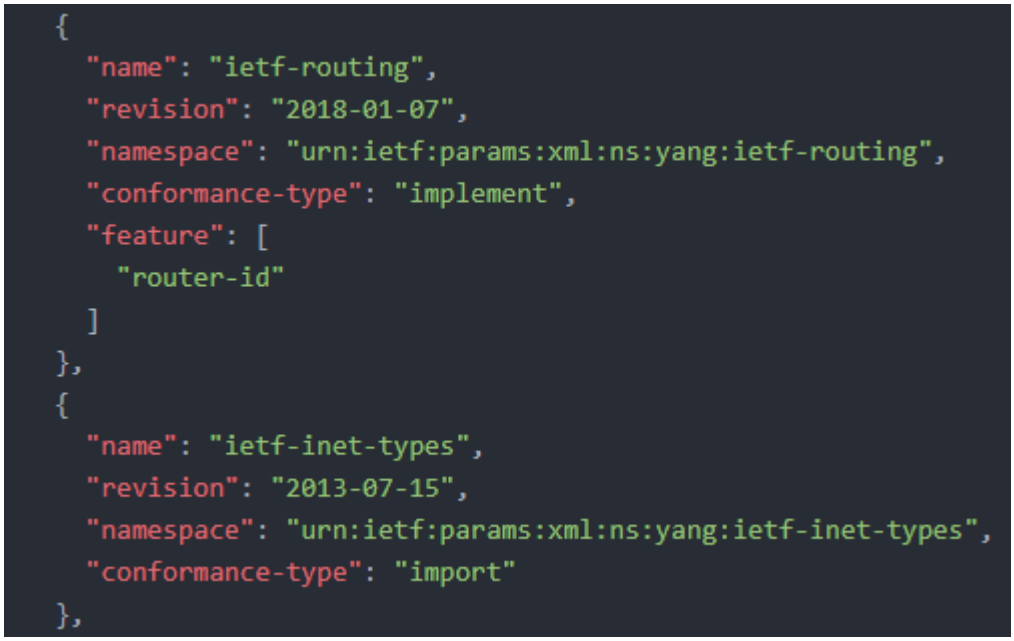
Pyang, además permite validar la sintaxis con varios módulos en conjunto, pero que tengan algún tipo de relación entre sí, por ejemplo, de aumento (extensión). Esto permite validar los XPath de los módulos que extiendan a otros o, por otra parte, visualizar el árbol jerárquico de todos los módulos extendidos.

3.3.2 Validación de implementación. La validación de implementación consiste en validar los módulos YANG existentes y necesarios para la implementación en un servidor RESTCONF. Estos módulos deben ser nombrados y listados en un datastore en formato Json que correspondan a una instancia del módulo YANG ietf-yang-library. La herramienta que se encarga de hacer este tipo de validación es Yangson.

Al momento de realizar la validación de implementación, Yangson realiza un análisis de sintaxis al datastore en formato Json que corresponda a la instancia del módulo YANG ietf-yang-library, en busca de errores en la digitación del archivo. Si esta validación se realiza sin problemas, la herramienta procede a interpretar los parámetros asignados a cada instancia de los módulos expuestos en el datastore de la biblioteca.

Uno de los parámetros más relevantes es el tipo de conformidad (conformance-type), este parámetro puede recibir como entrada lo siguiente: import, implement; El primer ítem consiste en que el módulo YANG solo se importe, con el fin de que los demás módulos que tengan dependencias de este funcionen correctamente. El segundo ítem consiste en que Yangson cargará el módulo YANG a la RAM, permitiendo realizar operaciones de configuración y de lectura de

estado sobre los datos configurados. Esta opción, por su naturaleza, al mismo tiempo realiza las veces de la opción import.



```
{
  "name": "ietf-routing",
  "revision": "2018-01-07",
  "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing",
  "conformance-type": "implement",
  "feature": [
    "router-id"
  ]
},
{
  "name": "ietf-inet-types",
  "revision": "2013-07-15",
  "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
  "conformance-type": "import"
},
```


Figura 25. Dos instancias de módulos YANG, una en implement y otra en import.

3.3.3 Validación de datos. La validación de datos consiste en, al haber validado la implementación de los módulos YANG, validar los datos planteados en un datastore en formato Json, que instancie los módulos YANG que están configurados en modo implement, a través del módulo ietf-yang-library, como se explicó en la sección anterior.

La herramienta que se encarga de realizar esta validación nuevamente es Yangson. La forma en la que se realiza esta validación consiste en intentar acoplar los datos presentes en el datastore propuesto por el usuario o sistema en el esquema generado por los módulos cargados en la RAM. Si alguno de los datos presentes en el datastore no cumple con las características estipuladas en el módulo YANG sobre el cual se esté iterando en ese momento, se arroja un error y la validación falla. Lo mismo ocurre si, en el datastore, se presentan datos que correspondan a segmentos no existentes dentro del módulo YANG sobre el que se esté iterando.

Si la validación de datos es exitosa, queda cargado en la RAM el esquema de los módulos YANG que estuviesen en tipo de conformancia implement, junto con los datos que se le asignaron a través del datastore.

Figura 26. Instancia de ietf-interfaces en un datastore en formato Json.

3.3.4 Validación con Pyangbind. Debido a la funcionalidad que ofrece Pyangbind para interactuar directamente con los módulos YANG, durante el desarrollo del proyecto se implementó una versión temprana del modelo YANG del protocolo RPL, donde se usó Pyangbind para validar la extensión del módulo ietf-routing y el patrón de diseño de fábrica abstracta, con el fin rectificar la robustez y correcto funcionamiento de esas características del modelo YANG.

A continuación, se presenta el código Python de la interacción de los objetos creados con las clases generadas por Pyangbind para los módulos uis-rpl y uis-ofs-of0.

```
rpl = router.routing.control_plane_protocols.control_plane_protocol.add(
    type = 'rpl',
    name = 'RPL'
)
```

```
rpl.rpl.trickle.i_min = 60*1000 # 1 minute to milliseconds
rpl.rpl.trickle.i_max = 4
rpl.rpl.trickle.k = 2
rpl.rpl.interfaces.interface.add('wireless1')

# OBJECTIVE FUNCTION
of = rpl.rpl.objective_functions.objective_function.add(
    type = 'of-of0',
    name = 'of0'
)
```

Figura 27. Agregación de un objeto de tipo función objetivo a un objeto de tipo RPL.

El resultado de esta validación confirmó la correcta extensión de los módulos, de forma jerárquica, siguiendo el orden correcto: ietf-routing, uis-rpl, uis-ofs-of0; Respectivamente. Además, verifica el adecuado funcionamiento del patrón de diseño anteriormente mencionado, fábrica abstracta, para la creación y agregación de módulos de funciones objetivo.

Por la naturaleza de esta validación, solo es necesario realizase una vez durante todo el desarrollo del modelo YANG, debido a que las demás validaciones pertinentes se hacen con la herramienta Yangson de forma más rápida y eficaz.

3.4 Implementación del servidor RESTCONF

3.4.1 Backend de Jetconf. Jetconf, como se mencionó en la sección 2.2.6, permite a través de la API que proporcionan, diseñar y desarrollar backends personalizados para las diversas implementaciones que se requieran realizar.

Para el presente proyecto, se desarrolló un backend de Jetconf personalizado al cual se le nombró “Empalme”. Este backend permite interactuar directamente con el modelo YANG del protocolo de encaminamiento RPL a través del servidor Jetconf, el cual brinda la implementación

del protocolo RESTCONF, es decir, a través del backend Empalme, es posible realizar peticiones REST mediante RESTCONF directamente al modelo YANG del protocolo RPL.

El backend, desarrollado en Python, cumple con todos los requerimientos de implementación de la API de Jetconf, por lo tanto, basta con empaquetarlo, junto con los ficheros necesarios adicionales y asignarlo en el fichero de configuración del servidor Jetconf.

En la figura de a continuación se presenta el código fuente de la clase que genera la tabla de rutas sobre la marcha, cuando una petición REST de tipo GET solicita el recurso, a través de RESTCONF.

```
class RPLRoutingTable(StateDataListHandler):
    def generate_list(self, node_ii: InstanceRoute, username: str, staging: bool) -> JsonNodeT:
        rutas = []
        for ip in RPL['TOPOLOGIA']:
            par = {
                "dest-ip" : ip,
                "next-hop-ip" : RPL['TOPOLOGIA'][ip]['PADRE']
            }
            rutas.append(par)
        return rutas
```

Figura 28. Clase para la generación de la lista de la tabla de rutas, de tipo datos de estado (ro).

3.4.2 Configuración del servidor Jetconf. Finalmente, una vez teniendo el backend desarrollado, es necesario configurar el servidor antes de su ejecución. Para esto, se usa un fichero en formato YAML (Yet Another Modeling Language, por sus siglas en inglés) donde se especifican y asignan ciertos parámetros respecto a la ejecución del servidor y sus submódulos.

De estos parámetros destacan el nombre de la raíz que tendrá la URL al momento de ejecutar el servidor, la habilitación de certificados SSL para realizar conexiones autenticadas y cifradas, la ruta al datastore principal, el paquete del backend a implementar y la ruta al directorio de módulos yang (donde se deben guardar todos los módulos YANG listados en la biblioteca YANG).

```
GLOBAL:
  TIMEZONE: "America/Bogota"
  LOGFILE: "-"
  PIDFILE: "/tmp/jetconf.pid"
  PERSISTENT_CHANGES: false
  LOG_LEVEL: "debug"
  LOG_DBG_MODULES: ["usr_conf_data_handlers", "usr_state_data_handlers"]
  YANG_LIB_DIR: "./Modulos-yang"
  DATA_JSON_FILE: "./BD.json"
  BACKEND_PACKAGE: "Empalme"

HTTP_SERVER:
    DOC_ROOT: "./Datos/doc-root"
    DOC_DEFAULT_NAME: "index.html"
    API_ROOT: "/rpl"
    SERVER_NAME: "restconf-rpl"
    DBG_DISABLE_CERT: true
    DBG_DISABLE_CERTS: true
    DISABLE_SSL: true
    SERVER_SSL_CERT: "./certs/server_servidor.crt"
    SERVER_SSL_PRIVKEY: "./certs/server_servidor.key"
    CA_CERT: "./certs/ca.pem"
```

Figura 29. Configuración principal del servidor Jetconf.

3.5 Diseño de pruebas para la interacción Modelo-RESTCONF por HTTP

En este capítulo se presentan las pruebas a realizar, y su respectivo planteamiento de diseño, sobre el proyecto en pleno funcionamiento, partiendo con la base de que el software consiste en una API REST alojada en un servidor que trabaja con el protocolo RESTCONF, por lo tanto, el punto de partida es el CRUD.

El formato para cada prueba consiste en una entrada de datos, sus parámetros y el resultado esperado, como se puede ver en la siguiente tabla:

Tabla 2.

Formato para diseño de pruebas.

Nombre de prueba	
Entrada:	**Parámetros:**
Resultados esperados:	

3.5.1 Acceso y visualización de datos.

Tabla 3.

Diseño de prueba de visualización de datos.

Acceso y visualización de datos	
Entrada:	**Parámetros:**
GET URL /recurso	IP o dominio del servidor (Opcional) Identificador del recurso a acceder
Resultados esperados:	
Retorno por parte del servidor de los datos del recurso accedido o de una lista de recursos.	

3.5.2 Creación de recursos (adición de datos).

Tabla 4.

Diseño de prueba de adición de datos.

Adición de datos	
Entrada:	**Parámetros:**
POST Datos en formato Json URL /recurso	IP o dominio del servidor (Opcional) Identificador del recurso a agregarle otros recursos
Resultados esperados:	
Creación de un nuevo recurso en la dirección especificada.	

Visualización del recurso añadido.

3.5.3 Modificación de recursos (actualización de datos).

Tabla 5.

Diseño de prueba de actualización de datos.

Actualización de datos	
Entrada:	**Parámetros:**
PATCH o PUT Datos en formato Json URL /recurso	IP o dominio del servidor Identificador del recurso a modificar (Opcional) Nuevo identificador del recurso
Resultados esperados:	
Modificación del recurso en la dirección especificada. Visualización del recurso modificado.	

3.5.4 Llamada a procedimiento remoto.

Tabla 6.

Diseño de prueba para llamadas a procedimientos remotos.

RPC	
Entrada:	**Parámetros:**
POST Datos en formato Json URL /procedimiento	IP o dominio del servidor
Resultados esperados:	
Retorno por parte del servidor el contenido, en formato Json, del resultado presentado por la ejecución del procedimiento.	

3.6 Visión macroscópica del proyecto

En la figura 30 se presenta un diagrama del sistema, a partir de una visión holística y macroscópica.

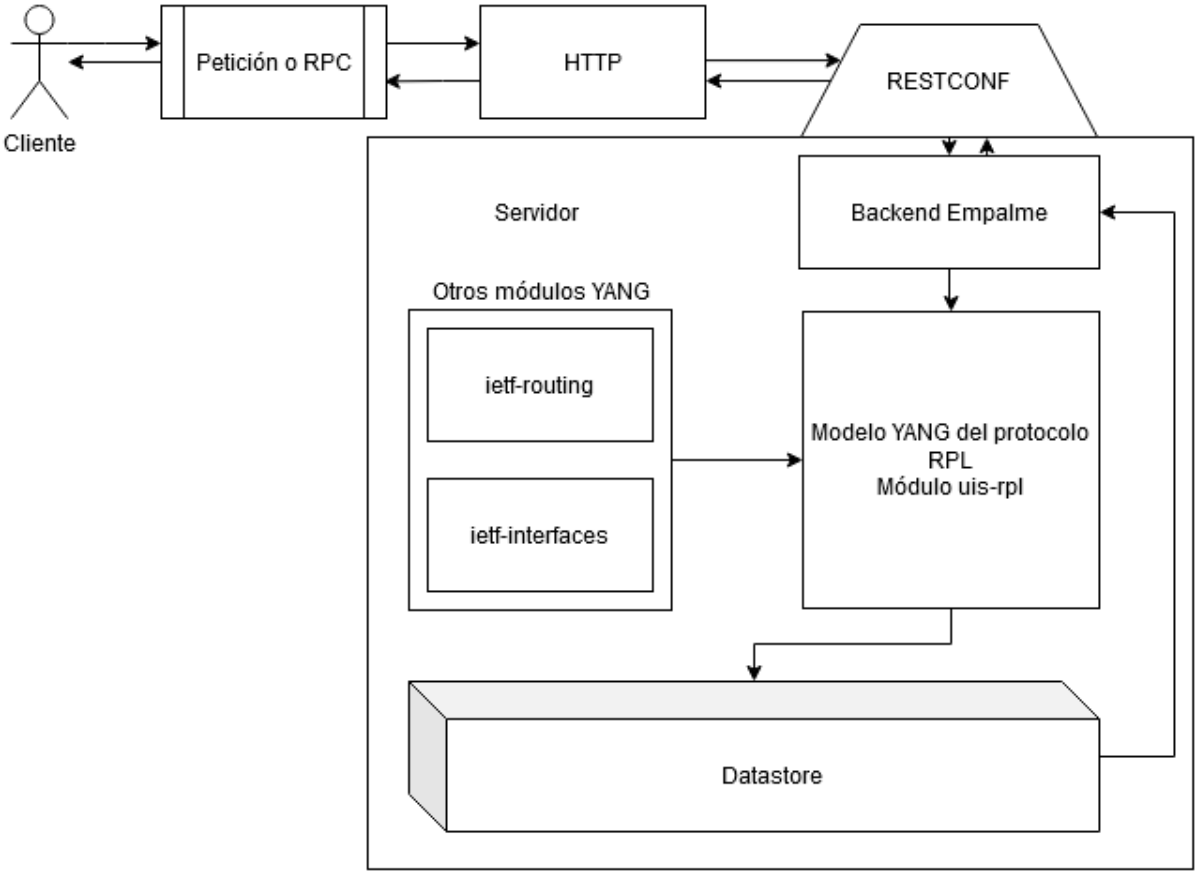


Figura 30. Diagrama del sistema.

El funcionamiento del sistema, explicado de forma general, se basa en peticiones que realiza un cliente (estas pueden ser una petición hacia uno o varios recursos o una llamada a procedimiento remoto) a través de verbos HTTP, es decir, obedeciendo a una API REST. Estas peticiones son interpretadas por el protocolo RESTCONF, si la petición cumple con las validaciones propias de este protocolo, procede a ser manejada por el backend de Jetconf configurado, en este caso Empalme.

El backend Empalme, dependiendo de si es una petición o llamada a procedimiento remoto, solicitará ciertos datos basados en la estructura modelada con un módulo YANG, para este proyecto este modelo de datos es el proporcionado por el modelo YANG del protocolo RPL. Si la petición cumple con la estructura de datos correcta, se solicitarán, agregarán o modificarán los

datos presentes en el datastore. Finalmente, se retorna una respuesta por parte del servidor hacia el cliente.

4. Resultados

En este capítulo se muestran los resultados de las dos fases más importantes del desarrollo del proyecto: el modelo YANG del protocolo RPL y las pruebas al servidor RESTCONF usando el backend de Jetconf personalizado “Empalme”.

Todos los resultados están presentes con más detalle en los apéndices, en concreto, el código de los módulos YANG uis-rpl y uis-ofs-of0 en los apéndices D y E, respectivamente, y los resultados de las pruebas RESTCONF en el apéndice C.

4.1 Modelo YANG del protocolo de encaminamiento RPL

4.1.1 Módulo uis-rpl. El modelo YANG del protocolo de encaminamiento RPL, cuyo nombre de módulo es uis-rpl, cumple con el objetivo de adecuarse y funcionar correctamente con el estándar para la NMDA, véase la sección 3.2.4, además de tener como características principales la posibilidad de extender el modelo para agregarle funciones objetivo diferentes, siempre y cuando cumplan con la interfaz planteada siguiendo el patrón de diseño de software de fábrica abstracta.

Los segmentos estipulados en la sección 3.2.1 y 3.2.2 fueron completamente desarrollados y ensamblados. La siguiente figura, el árbol de datos generado con Pyang al módulo uis-rpl, presenta algunos segmentos individuales.

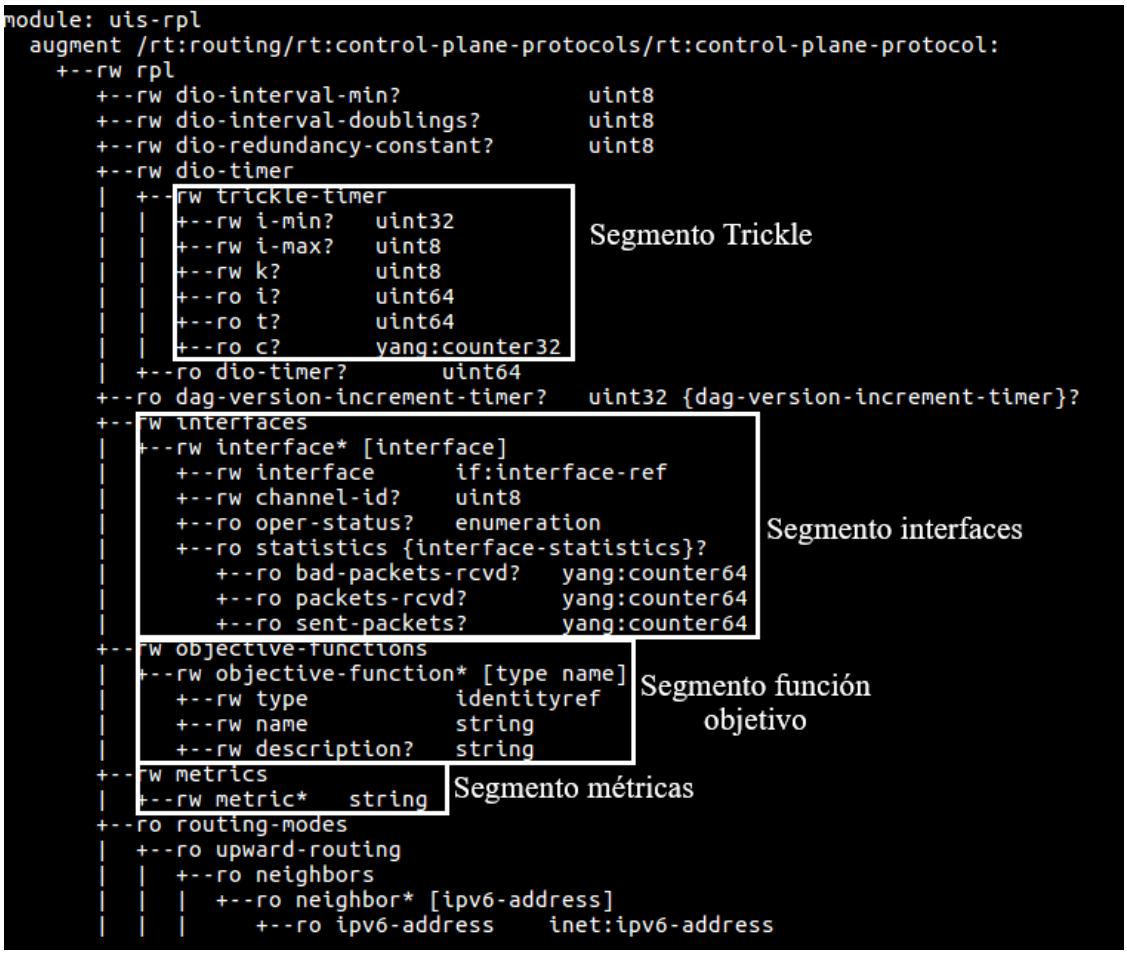


Figura 31. Todos los segmentos independientes, exceptuando el segmento Estadísticas.

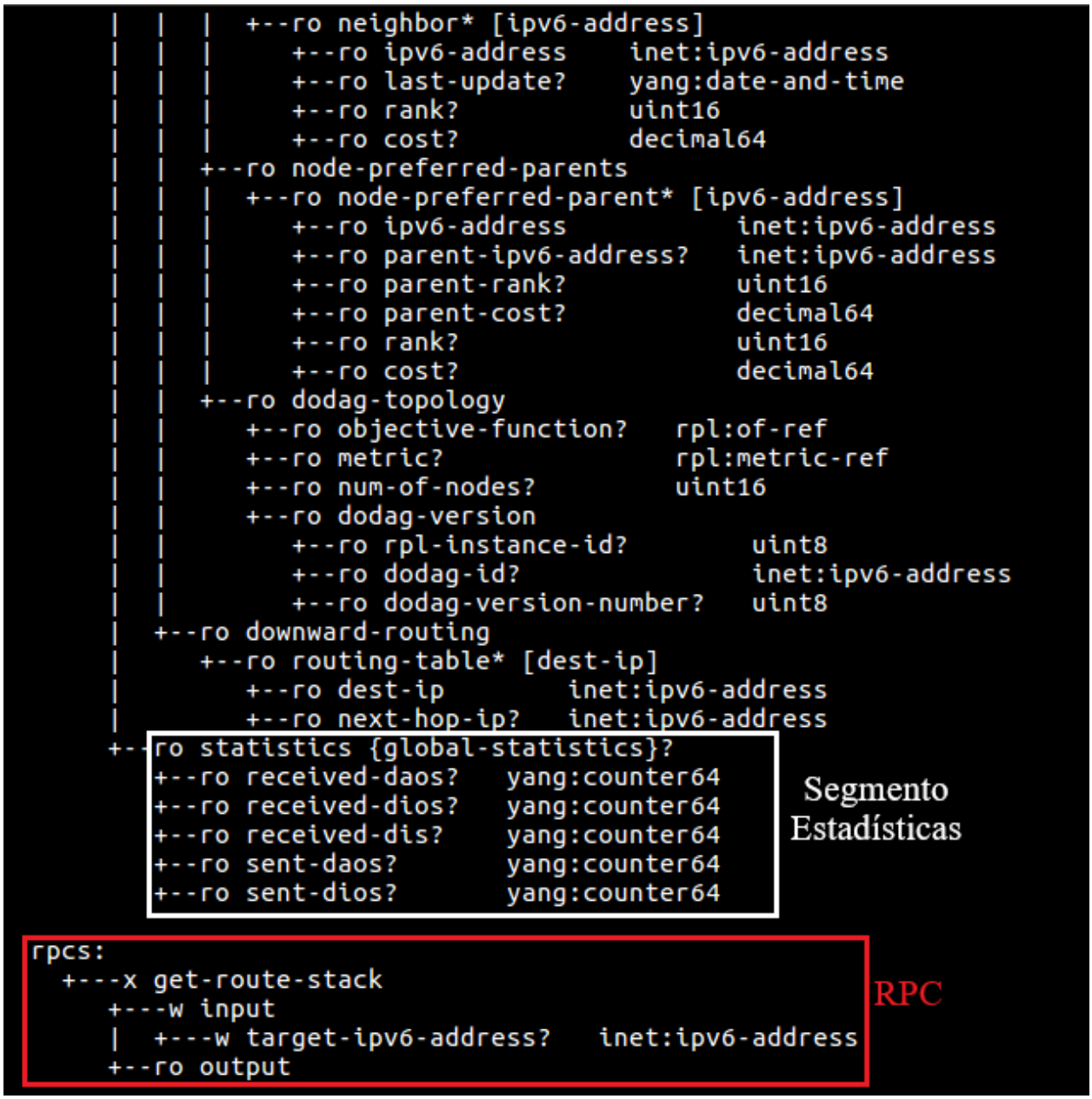


Figura 32. Segmento estadísticas y RPC.

El código completo del módulo uis-rpl se encuentra en el apéndice D.

4.1.2 Módulo uis-ofs-of0. Adicionalmente al modelo YANG del protocolo de encaminamiento RPL, se diseñó un pequeño modelo para la función objetivo que trae por defecto este protocolo, la función objetivo cero, presentada en el RFC 6552 (Thubert, 2012). El nombre del módulo del modelo es uis-ofs-of0, donde se plantea la nomenclatura que siga el espacio de nombres uis-ofs.

Este pequeño modelo YANG sirve de ejemplo de cómo implementar nuevas funciones objetivo, extendiendo al módulo uis-rpl, como se ve en la figura siguiente:

```
identity of-of0 {
  base rpl:objective-function;
}

augment "/rt:routing/rt:control-plane-protocols/"
  + "rt:control-plane-protocol" +
  "/rpl:rpl/rpl:objective-functions/rpl:objective-function" {
  when "derived-from-or-self(rpl:type, 'of0:of-of0')";

  container of0 {
```

Figura 33. Instrucción de extensión al módulo uis-rpl

El contenido del módulo uis-ofs-of0 se presenta, en forma de árbol de datos, a continuación:

```
module: uis-ofs-of0
  augment /rt:routing/rt:control-plane-protocols/rt:control-plane-protocol/rpl:r
pl/rpl:objective-functions/rpl:objective-function:
    +--rw of0
       +--ro variables
       |  +--ro step-of-rank?    uint16
       |  +--ro rank-increase?   uint16
       +--rw parameters
          +--rw stretch-of-rank?   uint16
          +--rw rank-factor?       uint16
```

Figura 34. Módulo uis-ofs-of0.

El código completo de este módulo se encuentra en el apéndice E.

4.2 Ejecución del servidor Jetconf

Al momento de ejecutar el servidor Jetconf, este realiza las tres validaciones estipuladas en la sección 3.3 en el orden planteado, una vez superadas estas validaciones, el servidor procede a ejecutar las instrucciones de inicio y posteriormente abre el servicio del protocolo RESTCONF, siguiendo los lineamientos impuestos por el fichero de configuración.

En este caso, el servidor Jetconf está alojado dentro de una máquina virtual, cuya NIC está vinculada a través de un puente lógico de software a la NIC física de la máquina anfitriona, permitiendo obtener una dirección IP (la versión depende de la configuración DHCP de la LAN a la cual la máquina anfitriona esté conectada) de la misma red local, en el puerto 8443.

La siguiente figura muestra los mensajes informativos presentados por Jetconf cuando el servicio de servidor RESTCONF está en línea.

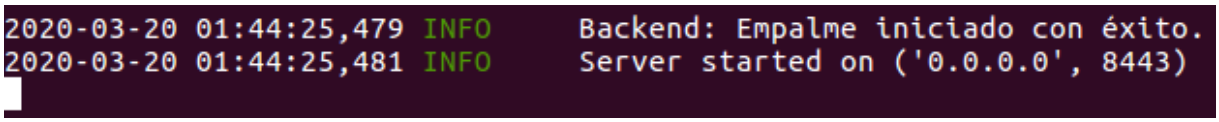


Figura 35. Mensajes informativos de Jetconf al terminar la inicialización.

4.2.1 Datos de simulación de enrutador en ejecución. Para las pruebas que se realizan al modelo YANG del protocolo RPL, al inicializar el servidor Jetconf, se carga un datastore con datos de configuración suficientes y pertinentes para instanciar a todos los módulos YANG necesarios para la correcta ejecución del módulo a testear, en este caso, el módulo uis-rpl. Adicionalmente, el paquete Empalme contiene datos, destinados a simulación, correspondientes a segmentos cuyo tipo de dato es de estado, es decir, que se generan sobre la marcha, debido a que, en un entorno de ejecución real, estos los proporcionaría la implementación del protocolo RPL.

Estos datos propios para simulación se basan en la siguiente topología:

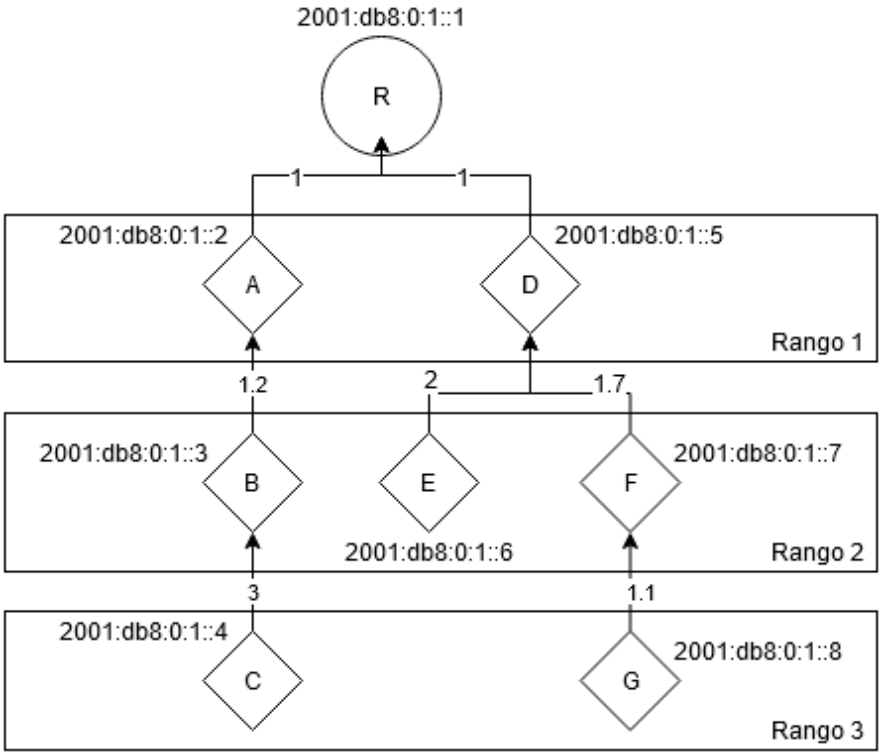


Figura 36. Topología planteada.

La métrica planteada en la topología corresponde a ETX.

4.3 Pruebas RESTCONF

Las pruebas realizadas al modelo YANG del protocolo RPL sobre RESTCONF están divididas en 3 categorías, las de acceso o visualización de datos, las de adición de datos al datastore y las de actualización de datos existentes del datastore. Por separado está la prueba al RPC.

Todos los scripts desarrollados para realizar las pruebas están nombrados y disponibles en los apéndices.

4.3.1 Pruebas de acceso y visualización de datos. Debido a que el protocolo RESTCONF funciona principalmente brindando una API REST, todas las pruebas de acceso a datos se realizan a través del verbo HTTP GET, el cual se encarga de realizar este tipo de solicitudes.

Adicionalmente, las pruebas de adición y de actualización de datos siempre realizan una petición de visualización de datos posterior a la operación hecha, a modo de confirmación.

4.3.1.1 Datastore completo

A continuación, se realiza la petición para ver todo el datastore disponible en el servidor. Esta petición se realiza a través del script **test_get_root.sh,** cuyo código es el siguiente:

```
curl --http2-prior-knowledge -X GET "http://$1:8443/rpl/data"
```

Figura 37. Código del script test_get_root.sh

El resultado de la petición es el siguiente:

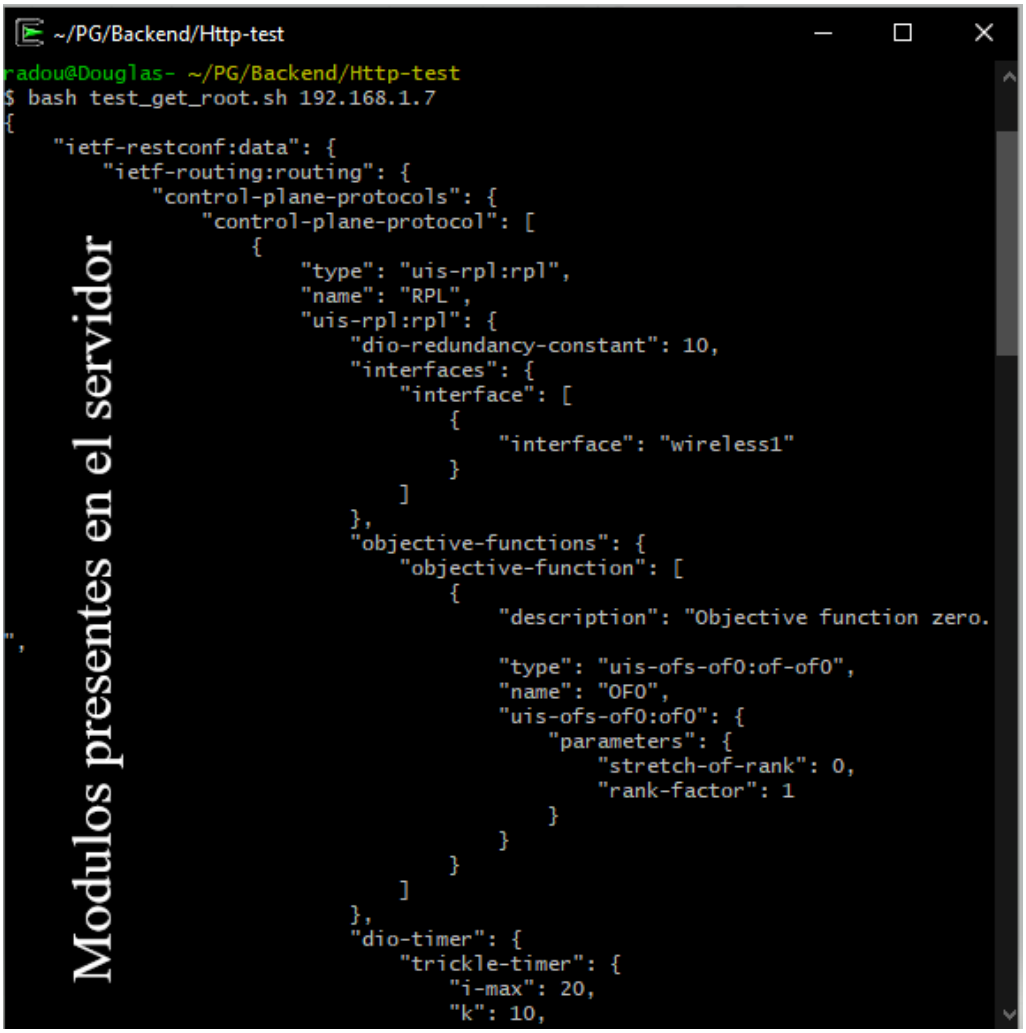


Figura 38. Respuesta del servidor con datastore completo.

4.3.1.2 Datos de estado: Tabla de rutas y topología mediante tabla de padres elegidos

La prueba para visualizar los datos de estado de los segmentos de Tabla de rutas y Padres elegidos se realiza a través de la ejecución del script **test_get_state_data.sh.**

```
echo "====TOPOLOGÍA:===="
curl --http2-prior-knowledge -X GET "http://$1:8443/rpl/data/\
ietf-routing:routing/control-plane-protocols/\
control-plane-protocol=uis-rpl:rpl,RPL/uis-rpl:rpl/\
routing-modes/upward-routing/node-preferred-parents/node-preferred-parent"

echo "====TABLA DE RUTAS===="
curl --http2-prior-knowledge -X GET "http://$1:8443/rpl/data/\
ietf-routing:routing/control-plane-protocols/\
control-plane-protocol=uis-rpl:rpl,RPL/uis-rpl:rpl/\
routing-modes/downward-routing/routing-table"
```

Figura 39. Código del script test_get_state_data.sh.

Los resultados de la ejecución del script son los siguientes:

Figura 40. Tabla de padres elegidos como respuesta a la petición hecha al servidor.

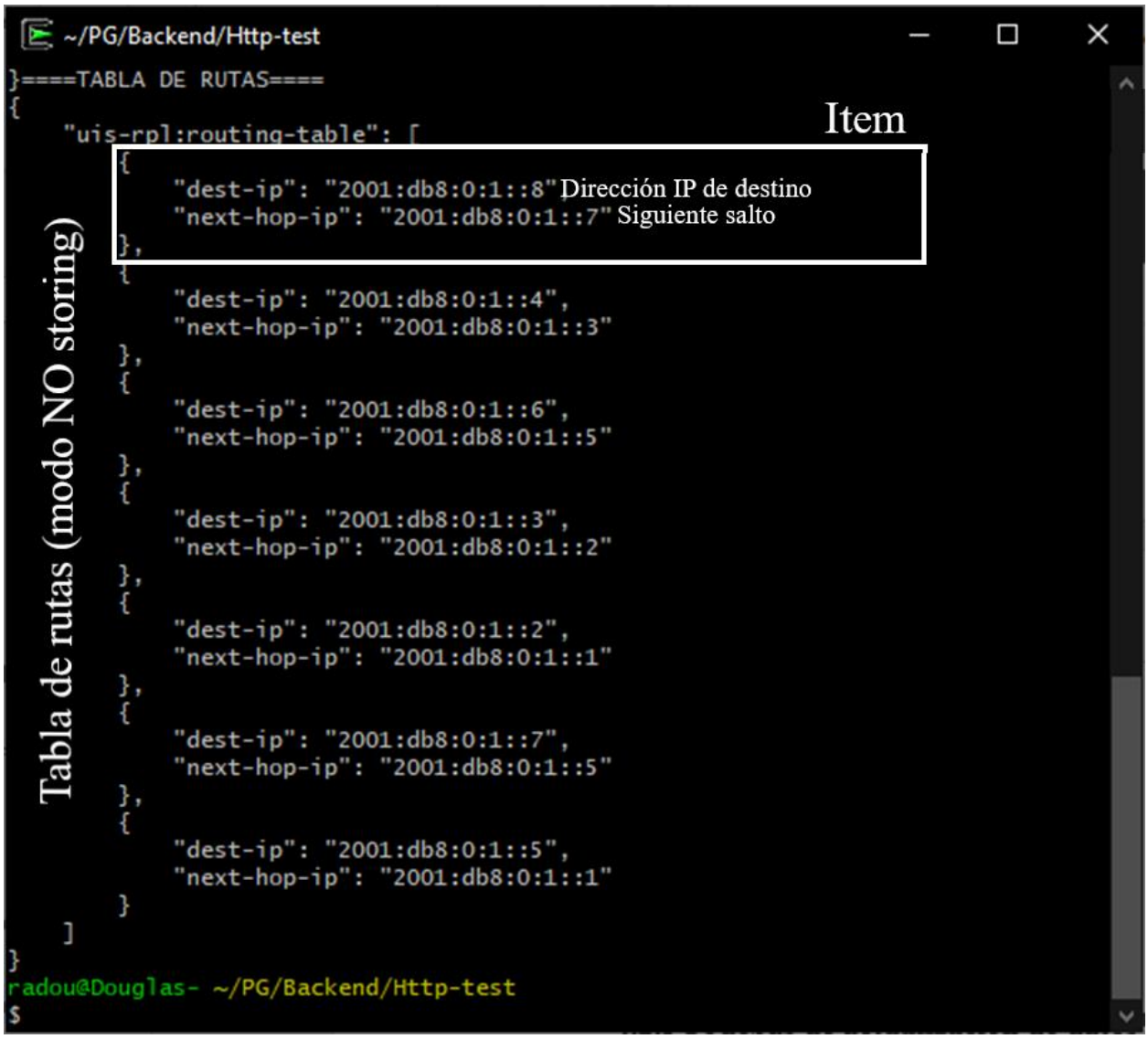


Figura 41. Tabla de rutas como respuesta del servidor.

4.3.2 Pruebas de adición de datos. Las pruebas de adición de datos se realizan enviando peticiones al servidor con el verbo HTTP POST. El script desarrollado para hacer las peticiones necesarias para probar esta funcionalidad, al modelo YANG del protocolo RPL, agrega entradas a la lista de métricas, perteneciente al segmento de Métricas. En un entorno real, esto se haría dependiendo de los algoritmos de métricas que el dispositivo que ejecuta el protocolo RPL tenga implementadas.

A continuación, se muestra la lista de métricas disponible de forma inicial (debido a que esto se encuentra indicado en el datastore), a través de una petición de visualización.

Figura 42. Lista de métricas definidas inicialmente en el datastore.

Esta petición se hace con el fin de mostrar las métricas disponibles antes de añadir las siguientes, a modo de demostrar la adición exitosa.

A través del script **test_add_metric.sh** se pueden añadir métricas con un nombre definido por el usuario, el código es el siguiente:

```
echo 'AÑADIENDO...'
curl --http2-prior-knowledge -X POST -d '{"uis-rpl:metric":"'$2'"}' \
"http://$1:8443/rpl/data/\
ietf-routing:routing/control-plane-protocols/\
control-plane-protocol=uis-rpl:rpl,RPL/uis-rpl:rpl/metrics"

echo 'RESULTADO...'
curl --http2-prior-knowledge -X GET "http://$1:8443/rpl/data/\
ietf-routing:routing/control-plane-protocols/\
control-plane-protocol=uis-rpl:rpl,RPL/uis-rpl:rpl/metrics"
```

Figura 43. Código del script test_add_metric.sh.

En la siguiente figura se muestra cómo se añade la métrica ETT (Expected Transmission Time):

```
~/PG/Backend/Http-test

radou@Douglas- ~/PG/Backend/Http-test
$ bash test_add_metric.sh 192.168.1.7 ETT
AÑADIENDO...
RESULTADO...
{
    "uis-rpl:metrics": {
        "metric": [
            "ETX",
            "ETT"
        ]
    }
}
radou@Douglas- ~/PG/Backend/Http-test
$
```

Figura 44. Resultado devuelto por el servidor a la petición de adición de una métrica.

Como se puede apreciar, hay un nuevo ítem dentro de la lista de métricas, cuyo nombre (y llave, debido a que corresponde a una lista de hojas) es ETT.

Para realizar la prueba siguiente, actualizar un dato, se añade otra entrada a la lista de métricas, que represente un dato válido para el modelo YANG del protocolo RPL, pero inválido para el entorno de ejecución del dispositivo, por tanto, habría que realizar una corrección:

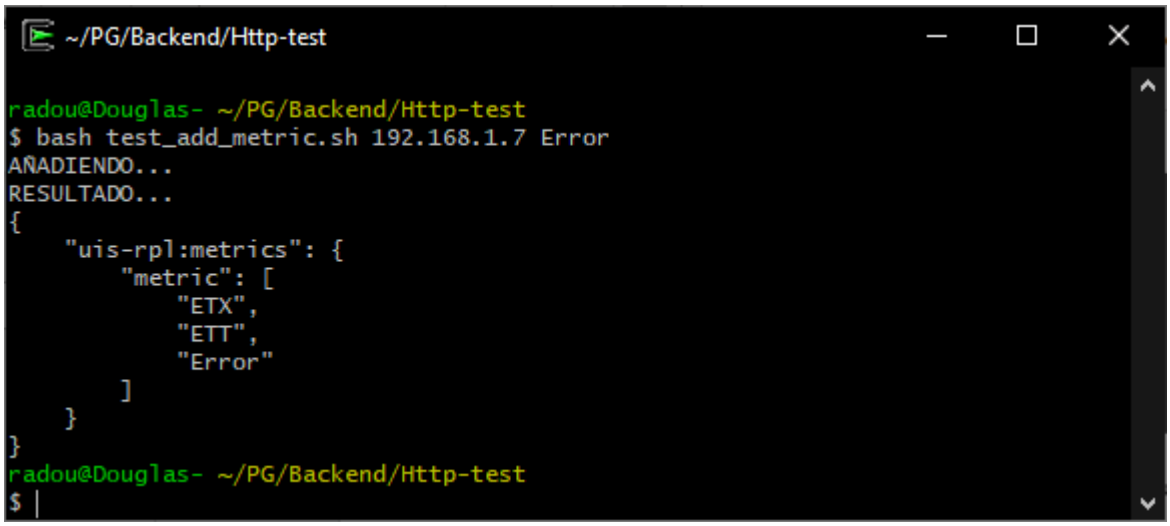


Figura 45. Respuesta del servidor al añadir una entrada cuyo nombre es Error.

4.3.3 Pruebas de actualización de datos. La arquitectura REST, a través de la API planteada, permite la modificación de datos mediante dos posibles verbos HTTP: PUT y PATCH. Ambos verbos tienen la misma finalidad, actualizar un recurso. No obstante, se diferencian en la forma en la que se realiza el cambio sobre el recurso en cuestión.

El verbo PUT realiza un reemplazo total del recurso, es decir, sobrescribe por completo el recurso estipulado. Esto hace obligatorio que se indiquen todos los datos necesarios, incluido el dato a corregir, del recurso a reemplazar. En palabras más simplificadas, el verbo PUT elimina el recurso a actualizar y crea uno nuevo en su lugar, con los datos enviados por el mensaje HTTP.

El verbo PATCH, a diferencia del verbo PUT, permite modificar secciones específicas del recurso, sin tener que especificar todos los datos de este.

A continuación, se presenta el código del script **test_update_metric.sh.** Este script es el usado para realizar la corrección mediante el verbo PUT, de la entrada errónea en la lista de métricas, añadido adrede en la sección 4.2.2.

```
echo 'ACTUALIZANDO...'
curl --http2-prior-knowledge -X PUT -d '{"uis-rpl:metric":"'$3'"}' \
"http://$1:8443/rpl/data/\
ietf-routing:routing/control-plane-protocols/\
control-plane-protocol=uis-rpl:rpl,RPL/uis-rpl:rpl/metrics/metric=$2"

echo 'RESULTADO...'
curl --http2-prior-knowledge -X GET "http://$1:8443/rpl/data/\
ietf-routing:routing/control-plane-protocols/\
control-plane-protocol=uis-rpl:rpl,RPL/uis-rpl:rpl/metrics"
```

Figura 46. Código del script test_update_metric.sh.

Este script recibe tres (3) parámetros (los encerrados con un pequeño cuadrado blanco en la figura) en lugar de dos (2) como el script usado en la sección anterior. Esto es debido a que el tercer parámetro, para esta prueba, recibe el nombre por el cual se quiere reemplazar (o corregir) del recurso.

En la siguiente figura se muestra el resultado de reemplazar la entrada errónea de la lista de métricas por la métrica Hops Count (HC).

```
~/PG/Backend/Http-test

radou@Douglas- ~/PG/Backend/Http-test
$ bash test_update_metric.sh 192.168.1.7 Error HC
ACTUALIZANDO...
RESULTADO...
{
    "uis-rpl:metrics": {
        "metric": [
            "ETX",
            "ETT",
            "HC"
        ]
    }
}
radou@Douglas- ~/PG/Backend/Http-test
$
```

Figura 47. Respuesta del servidor a la petición de reemplazo de la entrada Error por HC.

En caso de que se intente corregir o reemplazar un elemento que no se encuentre en la lista, el servidor responde con el siguiente mensaje, indicando el error:

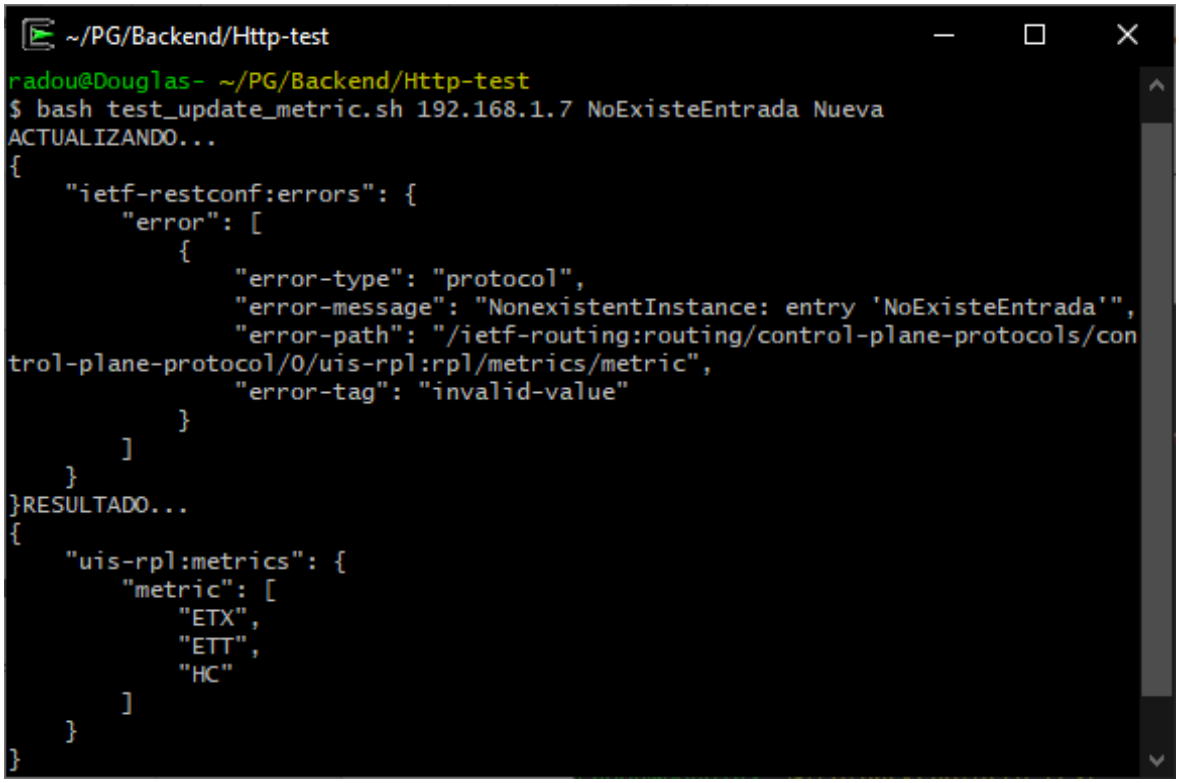


Figura 48. Respuesta del servidor a una petición inválida.

La lista no se ve afectada, debido a que la petición enviada fue rechazada por ser inválida, en este caso, por intentar reemplazar una entrada que no existe en la tabla.

4.3.4 Pruebas al RPC. Para realizar la prueba a la llamada a procedimiento remoto, se utiliza el script llamado **test_rpc_routes.sh**, este script a su vez requiere de un fichero llamado **get_route_stack.json**, el cual contiene los datos de entrada estipulados en el modelo YANG del protocolo RPL.

```
{
  "uis-rpl:input":{
    "target-ipv6-address":"2001:db8:0:1::8"
  }
}
```

Figura 49. Contenido del fichero get_route_stack.json

El código del script **test_rpc_routes.sh** se encuentra en la siguiente figura:

```
curl --http2-prior-knowledge -X POST -d "@get_route_stack.json" \
 "http://$1:8443/rpl/operations/uis-rpl:get-route-stack"
```

Figura 50. Código del script test_rpc_routes.sh.

El objetivo de esta llamada a procedimiento remoto, como se explica en mayor detalle en la sección 3.2.3, es devolver la pila de direcciones IPv6 a recorrer para enviar un paquete a un nodo destino a partir del nodo raíz.

A continuación, se presenta el resultado de la llamada a procedimiento remoto, buscando como destino al nodo con dirección IPv6 2001:db8:0:1::8.

```
~/PG/Backend/Http-test

radou@Douglas- ~/PG/Backend/Http-test
$ bash test_rpc_routes.sh 192.168.1.7
[
    "2001:db8:0:1::5",
    "2001:db8:0:1::7",
    "2001:db8:0:1::8"
]
radou@Douglas- ~/PG/Backend/Http-test
$
```

Figura 51. Respuesta del servidor al RPC get_route_stack.

5. Conclusiones

El desarrollo de los segmentos estipulados para el modelo YANG del protocolo RPL, se realizó por completo, ofreciendo una base sólida para el desarrollo posterior de un modelo YANG del protocolo RPL más extenso, de acuerdo con las necesidades que podrían ser expuestas en una RFC por parte de la IETF.

El modelo YANG resultante cumple con los requerimientos necesarios del estándar propuesto para la NMDA, como se puede ver en la sección 3.2.4, extendiendo el módulo ietf-routing y combinando de forma lógica y efectiva los datos de estado y configuración en los segmentos respectivos. Además, se plantea, mediante el patrón de diseño de fábrica abstracta, una arquitectura eficaz para la implementación o posterior desarrollo de funciones objetivo más complejas, de acuerdo con las necesidades presentadas o casos posibles que podrían llegar a darse en el momento de acoplar o adoptar el protocolo de encaminamiento RPL en diferentes entornos.

Mediante las pruebas pertinentes realizadas a cada utilidad brindada por RESTCONF, el modelo YANG del protocolo RPL realizado demuestra solidez y robustez, conformando una base confiable para la implementación de este modelo en servidores de gestión de redes cuyas funciones se realicen a través de RESTCONF (o planeen migrar de NETCONF).

En la índole personal, este proyecto ha sido una gran fuente de aprendizaje sobre el área del Network DevOps, además de abrir las posibilidades y brindar el conocimiento sobre la programación de redes manejadas con modelos YANG.

6. Recomendaciones

En este capítulo se presentan varias direcciones de trabajos futuros que se pueden derivar de la investigación realizada en este documento.

- Las necesidades para el IoT están en constante aumento debido a la evolución, invención e implementación de nuevas tecnologías. Por lo cual, se proporciona la sugerencia de investigar, desarrollar y ensamblar segmentos adicionales al modelo YANG desarrollado en este proyecto, por ejemplo, reparación local del DODAG y reparación global del DODAG.
- Debido a la naturaleza del protocolo RESTCONF en el estado del arte, podría ser una buena oportunidad desarrollar un prototipo de implementación del protocolo RPL en Python, el cual se podría vincular en una máquina que use Jetconf y el modelo YANG desarrollado en este proyecto. En otras palabras, reemplazar la parte de simulación de este proyecto con una implementación real del protocolo RPL escrita en Python.

Las herramientas recomendadas para proyectos derivados de este, aparte de las nombradas en el presente documento, son Atom, Git, Jupyter Notebook y validadores de código Json.

Referencias Bibliográficas

Alexander, R., Brandt, A., JP, V., Hui, J., Pister, K., Thubert, P., … Winter, T. (2012). RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. *Tools.Ietf.Org*, p. 157. https://doi.org/10.17487/RFC6550

Bierman, A., Bjorklund, M., & Watsen, K. (2017). *RESTCONF Protocol*. https://doi.org/10.17487/RFC8040

Bierman, Andy. (2018, October). *Guidelines for Authors and Reviewers of Documents Containing YANG Data Models*. https://doi.org/10.17487/RFC8407

Bierman, Andy, Björklund, M., Schönwälder, J., Watsen, K., & Wilton, R. (2019, March). *YANG Library*. https://doi.org/10.17487/RFC8525

Birrell, A. D., & Nelson, B. J. (1984). Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems (TOCS)*. https://doi.org/10.1145/2080.357392

Björklund (Ed.), M. (2016). *The YANG 1.1 Data Modeling Language* (M. Bjorklund, Ed.). https://doi.org/10.17487/RFC7950

Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., & Wilton, R. (2019). *RESTCONF Extensions to Support the Network Management Datastore Architecture*. https://doi.org/10.17487/RFC8527

Björklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., & Wilton, R. (2018). *Network Management Datastore Architecture (NMDA)*. https://doi.org/10.17487/RFC8342

Björklund, Martin. (n.d.). Pyang: An extensible YANG validator and converter in python. Retrieved June 10, 2019, from https://github.com/mbj4668/pyang

Björklund, Martin. (2010). YANG - A Data Modeling Language for the Network Configuration

Protocol (NETCONF). *RFC Editor*, 173. Retrieved from https://rfc-editor.org/rfc/rfc6020.txt

Byrne, B. (2018, January). Hands On Kicking the Tires of RESTCONF. *Hands On Kicking the Tires of RESTCONF*, 17. Retrieved from https://www.ciscolive.com/c/dam/r/ciscolive/emea/docs/2018/pdf/DEVNET-2585.pdf

Dunkels, A., & Vasseur, J.-P. (2010). RPL Routing in Smart Object Networks. In *Interconnecting Smart Objects with IP*. Morgan Kaufmann.

Enns, Rob, Bjorklund, M., Schoenwaelder, J., & Bierman, A. (2011). *Network Configuration Protocol (NETCONF)* (R. Enns, M. Bjorklund, J. Schoenwaelder, & A. Bierman, Eds.). https://doi.org/10.17487/rfc6241

Ladislav, L., Lindem, A., & Qu, Y. (2017). ietf-routing. Retrieved from http://www.claise.be/YANG-modules/ietf-routing@2018-01-07.yang

Lhotka, L., Lindem, A., & Qu, Y. (2018, March). *A YANG Data Model for Routing Management (NMDA Version)*. https://doi.org/10.17487/RFC8349

Liu, X., Sarda, P., & Choudhary, V. (2018). ietf-rip. Retrieved from https://github.com/YangModels/yang/blob/master/experimental/ietf-extracted-YANG-modules/ietf-rip%402018-02-03.yang

NIC.CZ. (2019). Jetconf documentation. Retrieved March 1, 2020, from https://jetconf.readthedocs.io/en/latest/

Noer, G. J. (1998). Cygwin32: A Free Win32 Porting Layer for UNIX® Applications. *2nd USENIX Windows NT Symposium*. Retrieved from https://www.usenix.org/legacy/publications/library/proceedings/usenix-nt98/full_papers/noer/noer_html/noer.html

Palacios, C., & Rodriguez, W. (2019). Arquitectura Cliente Servidor - EcuRed. Retrieved April 1,

2020, from https://www.ecured.cu/Arquitectura_Cliente_Servidor#Partes_que_componen_el_sistema

Shakir, R. (n.d.). Pyangbind - A plugin for Pyang that generates a Python class hierarchy from a YANG data model. Retrieved July 14, 2019, from http://pynms.io/pyangbind/getting_started/

Stenberg, D. (2017). *Everything curl*. Retrieved from https://web.archive.org/web/20170414151225/https://www.gitbook.com/download/pdf/book/bagder/everything-curl

Sukho, O., DongYeop, H., Kim, K., & Ki-Hyung, K. (2018). A hybrid mode to enhance the downward route performance in routing protocol for low power and lossy networks. *International Journal of Distributed Sensor Networks*, *14*, 13. https://doi.org/10.1177/1550147718772533

Thubert, P. (Ed.). (2012). *Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)*. https://doi.org/10.17487/rfc6552

Wikipedia. (2020). *VirtualBox --- Wikipedia, La enciclopedia libre*. Retrieved from https://es.wikipedia.org/w/index.php?title=VirtualBox&oldid=123735836