UNIVERSIDAD NACIONAL DE LA PLATA

Facultad de INFORMÁTICA
UNIVERSIDAD NACIONAL DE LA PLATA

Secretaría de Posgrado

Hybrid Networking SDN y SD-WAN: Interoperabilidad de Arquitecturas de Redes Tradicionales y Redes definidas por Software en la era de la digitalización

Tesis Doctoral – Ciencias Informáticas

Autor

Gustavo D. Salazar Chacón

Director

Luis Marrone

La Plata, Argentina, 2021

“Labor omnia improba vincit.”

“El trabajo tenaz todo lo vence.”

Virgilio (70 a.C. – 19 a.C.)

Poeta Romano

Geórgicas. Libro I. Líneas 145-146.

“Difficile est tenere quae acceperis nis exerceas.”

“Es difícil retener lo que has aprendido a menos que debas practicarlo.”

Gaius Caecilius Cilo (61 d.C. – 113 d.C.)

Poeta Romano

National University of La Plata - UNLP

Abstract

Informatics Faculty

Doctorate in Computer and Informatics Science

Hybrid Networking SDN and SD-WAN: Traditional Network Architectures and Software-Defined Networks Interoperability in digitization era

by Gustavo D. Salazar Chacón

For a decade, business data networks have undergone a dizzying evolution given by the adoption of cloud-based models, containers and microservices, primarily because they provide greater flexibility, effectiveness, and cost reduction in Information Technology.

Data networks allow transmitted information to reach any place, at any time, however, the requirements of current users and modern companies increasingly demand a better response from the telecommunications infrastructure. These requirements were more evident during the time of COVID-19 pandemic, leading us to the "Age of Data and Wisdom", virtuality, and Digital Transformation, since most human and economic activities depend on an adequate and secure transfer of information supported by data networks. This is how data became the most important intangible asset of the 21st Century.

As more 4G/5G-WiFi6E mobile devices enter the network, as well as novel applications and services, an increased forwarding and processing load is added, creating difficulties in both management and monitoring, as well as configuration and troubleshooting in case of failures.

Software-Defined Networking (SDN), since its creation and adoption, has promised to be the solution to the aforementioned problems through the use of techniques such as programmability, Open-Hardware with P4-programmatic capabilities, extreme agility, separation of concerns, in addition to the use of Graphical Interfaces that provide total visibility.

This thesis takes a look at the evolution of data networks towards the SDN paradigm and its various adoptions (*SD-Access, SD-Data Center and SD-WAN*). In order to check its feasibility of implementation, and addressing the fundamentals of these technologies, the investigation starts from what decoupling of Control Plane from the Data Plane in network equipment implies, to the concept of a cultural and technological change called *NetDevOps*, essential for the SDN-agile ecosystem to function properly, going through the analysis of next-generation standardized protocols that allow the implementation of these environments in real world: *LISP, VXLAN, OMP* and *Segment-Routing*, while developing proofs of concepts (PoCs) in emulated and physical scenarios, thus closing the investigative process that validates the SDN integration based on *software* and *hardware programmability* with traditional networks, and that is the main contribution of the present doctoral thesis.

Keywords

PoCs (Proof-of-Concepts); Software-Defined Networks (SDN); SD-WAN; SD-Access; LISP; VXLAN; Segment-Routing; Viptela; Open Networking; NetDevOps; OpenFlow; SDN-Controllers (ODL, ONOS, RYU); Data Formats (JSON, XML, YAML); NETCONF; RESTCONF; Data Models (YANG); Napalm, Netmiko; Ansible; P4; REST-APIs, SDLC.

Copyright

Universidad Nacional de La Plata - UNLP

Resumen

Facultad de Informática

Doctorado en Ciencias Informáticas

Hybrid Networking SDN y SD-WAN: Interoperabilidad de Arquitecturas de Redes Tradicionales y Redes definidas por Software en la era de la digitalización

por Gustavo D. Salazar Chacón

Desde hace una década, las redes de datos empresariales han experimentado una evolución vertiginosa dada por la adopción de modelos basados en la nube, contenedores y microservicios, primordialmente por que entregan mayor flexibilidad, efectividad y reducción de costos en cuanto a Tecnologías de la Información se refiere.

Las redes de datos permiten que la información transmitida pueda llegar en cualquier momento a cualquier lugar, sin embargo, los requerimientos de usuarios y empresas actuales exigen cada vez una mejor respuesta de la infraestructura de telecomunicaciones. Esos requerimientos fueron más evidentes durante la época de pandemia COVID-19, llevándonos a una "Era de Datos y Sabiduría", virtualidad y Transformación Digital, ya que la mayor parte de las actividades humanas y económicas dependen de una adecuada y segura transferencia de información sustentada por las redes de datos. Es así como los datos se convirtieron en el bien intangible más importante del siglo XXI.

A medida que más dispositivos móviles 4G/5G-WiFi6E ingresan a la red, así como aplicaciones y servicios novedosos entran en funcionamiento, una mayor carga de envío y procesamiento se añade a la red, generando dificultades tanto en la administración y monitoreo, como en la configuración y restablecimiento en caso de fallas.

Software-Defined Networking (o SDN por sus siglas en inglés), desde su creación y posterior adopción, ha prometido ser la solución a los problemas mencionados a través del uso de técnicas como programabilidad, *Hardware* Abierto con capacidades programáticas con P4, agilidad extrema, además del empleo de Interfaces Gráficas seguras que otorgan visibilidad total de la infraestructura.

La presente tesis da un vistazo a la evolución de las redes de datos hacia el paradigma SDN y sus diversas adopciones (*SD-Access, SD-Data Center* y *SD-WAN*) con el fin de comprobar su facilidad de implementación, para lo cual se aborda los fundamentos de dichas tecnologías, partiendo de lo que implica el desacoplamiento del Plano de Control del Plano de Datos en equipos de red, hasta el concepto de cambio cultural y tecnológico denominado *NetDevOps,* imprescindible para que el ecosistema ágil SDN funcione de forma adecuada, atravesando por el análisis de protocolos estandarizados de próxima generación que permiten la implementación de estos entornos en redes reales: LISP, VXLAN, OMP y *Segment-Routing*, desarrollando a la par pruebas de concepto (PoCs) en ambientes de emulación y con equipos físicos, cerrando de esa manera el proceso investigativo que da validez a la integración de SDN basadas en programabilidad con redes tradicionales, siendo esa justamente la mayor contribución entregada por esta tesis.

Palabras Clave

Pruebas de Concepto (PoCs); Redes Definidas por Software (SDN); SD-WAN; SD-Access; LISP; VXLAN; Segment-Routing; Viptela; Open Networking; NetDevOps; OpenFlow; Controladores SDN (ODL, ONOS, RYU); Formato de Datos (JSON, XML, YAML); NETCONF; RESTCONF; Modelo de Datos (YANG); Napalm, Netmiko; Ansible; P4; REST-APIs, SDLC.

Derechos de Autor

Dedicatoria

A mi amada esposa Cristinita, a mis amados padres, Sorayita y Gustavo Ramiro, a mis abuelitos, a mi querida familia y amigos, este logro es dedicado a todos y cada uno de ustedes.

Agradecimiento

Al concluir el objetivo más importante en mi vida académica, quiero agradecer a Dios y a la Virgen María por todo, en especial por cuidar de mi familia, seres queridos y de mí en momentos de incertidumbre tan difíciles que está atravesando el mundo y mi país.

Gracias a mis ángeles en el cielo que han intercedido en nuestras súplicas y oraciones ante Dios Padre: Isabelita, Pedrito y Pepito y a mi ángel en la Tierra Ginita, mis abuelitos.

A mi esposa Cristinita, te agradezco el apoyo incondicional que me has brindado durante estos años de compartir el camino juntos, gracias por darme ese aliento adicional que necesité en momentos de angustia, los besos y abrazos en momentos de alegría y por dedicarme palabras de amor para que siga hasta cumplir todos los objetivos que me proponga, te amo.

Agradezco profundamente a mis amados padres, Sorayita y Gustavo Ramiro, no hubiera conseguido absolutamente nada sin su guía, consejos, amor y comprensión, gracias por su tiempo, esfuerzo, apertura y ayudarme a ser una mejor persona y un mejor docente universitario que enseña basado en valores, valores que los aprendí por ustedes, mis mejores amigos y ejemplo a seguir, los amo papitos, ¡esto es por ustedes!

A mi familia y amigos, les agradezco sus buenos pensamientos y deseos, siempre han llegado en el momento justo, gracias por estar junto a mí.

Quiero agradecer a la Universidad Nacional de La Plata, en especial a mi director, Luis Marrone, quien confió en mí para junto a él dictar la cátedra de Redes Definidas por *Software* en la Universidad, compartiendo los conocimientos adquiridos durante mis estudios doctorales a estudiantes de Argentina y Ecuador, además de haberme guiado para la publicación de artículos científicos de vanguardia que sustentan esta investigación, siendo reconocidos con uno de ellos como Mejor artículo en la Universidad de Columbia, logro realizado con esfuerzo y dedicación.
Quiero agradecer también a toda la planta de profesores, al personal administrativo y todos quienes conforman la UNLP, sus conocimientos y experiencias compartidas han caído en terreno fértil, ténganlo por seguro que aplicaré todos lo aprendido durante el doctorado en mis actividades docentes y profesionales.

Finalmente, un agradecimiento a la SENESCYT, Secretaría de Educación Superior, Ciencia, Tecnología e Innovación de Ecuador, por el apoyo brindando para la consecución de este éxito académico.

¡Gracias!

Gustavo David

Índice de Contenidos

Lista de Figuras

Lista de Figuras (Anexos)

Lista de Tablas

Lista de *Scripts*

Página intencionalmente en blanco

Capítulo 1

1. Introducción

Las Tecnologías de la Información han evolucionado en las últimas décadas con una rapidez impresionante gracias a que los requerimientos de usuarios de redes modernas así lo han demandado, no en vano Platón mencionó que *"*La necesidad es la madre de todas las invenciones" (Platón, 369 ac).

Por nombrar algunas tendencias que modifican paradigmas dentro de las redes están:
BYOX, siglas de *Bring Your Own Everything*[1], el cual cambia la forma de desarrollar los negocios al permitir el uso de dispositivos electrónicos personales y acceder con ellos a la red empresarial en cualquier momento y cualquier lugar, mejorando la productividad notablemente más aún con el advenimiento de las redes inalámbricas y celulares de nueva generación (WiFi6E, 4G-LTE AdvancedPro y 5G/6G), las cuales se han repotenciado con la computación en la nube y virtualización tan necesaria hoy en día. Algo similar ocurrió en las industrias con redes de sensores (*All-IP Industrial Networks*) y automatización de sus sistemas mediante AI (inteligencia artificial), cambio conocido como Industria 4.0, impulsado por la conectividad de lo no conectado o IoT (*Internet of Things*), llevando ese concepto a lo más cercano al usuario final, el hogar, tema abordado por el *World Economic Forum* tanto en el año 2016 como en el 2017 (World Economic Forum, 2017).

Todos esos avances han hecho que sea difícil imaginar el momento cuando el universo digital no se encontraba a nuestra disposición. El ser humano ahora requiere sentirse conectado y tener toda la información al alcance de la mano y de forma instantánea, desarrollando un eslabón más en la conectividad extrema, donde las cosas, procesos, datos y personas se relacionan entre sí. Este concepto se amplió a lo que se definió como la Trasformación Digital, siendo ahora no solo un tópico tecnológico, sino un asunto que impacta e impactará a toda la sociedad en su conjunto.

La digitalización, corazón de la Transformación Digital antes mencionada, modificará la forma en que vivimos, trabajamos, aprendemos y jugamos, posibilitando traer todo lo imaginable a la realidad, justamente este tema fue tomando en cuenta en *Cisco Live* Barcelona 2019 (Cisco Systems - Senior VicePresident Liz Centoni, 2019), en el *Cisco Academy Conference LATAM* 2017 desarrollado en la Ciudad de México y el *Cisco Academy Conference LATAM* 2019 en Guayaquil, Ecuador (Cisco Systems, 2019).

Por nombrar unos ejemplos de esta digitalización están la VR (Realidad Virtual) y AR (Realidad Aumentada), tópicos en auge para el comercio y el entretenimiento, pero también lo será para revolucionar la forma de estudiar en escuelas y universidades, así como generar nuevos modelos de negocio y de trabajo, convirtiéndose en una oportunidad enorme para la humanidad de los próximos treinta o cuarenta años que tendrá la responsabilidad de no sólo pensar en la tecnología, sino en generar una sinergia entre innovación y medioambiente, por ello, las áreas

[1] También denominado Consumerización de las tecnologías de la información

que más énfasis tienen gracias a la digitalización y acorde a *Habitat III* (HabitatIII, 2016) dictado en Ecuador en el año 2016 son:

- Ciudades Inteligentes: Conjunto de servicios novedosos que mejorarán los estándares de vida de los ciudadanos. Tráfico automovilístico inteligente, manejo de sitios de estacionamiento de autos, vigilancia y seguridad ciudadana, comprar lo que se desea sin necesidad de hacer colas, trámites consolidados en ventanillas únicas y conectividad total a Internet inalámbrica segura y de altísimo ancho de banda, por nombrar algunos servicios a implementar en las ciudades modernas.
- Medioambiente Inteligente: Programas como manejo adecuado de desechos, uso de energía limpia, control de polución serán temas que mejorarán con sistemas IoT.
- Salud Inteligente: La Telemedicina servirá para llegar con cuidados médicos e incluso cirugías a sitios remotos usando redes convergentes y telepresencia, así como para monitorear a pacientes y recibir cuidados de médicos extranjeros en caso de ser necesario. IoT está muy involucrado en mejorar los tratamientos de salud.
- Agricultura Inteligente: Agricultura usando sensores para controlar la temperatura y humedad con el fin de maximizar la producción de productos del campo. Mejorar la cadena de distribución y manejo de los alimentos también forma parte de este sistema inteligente.
- Industria, Producción y Logística Inteligente: Sistemas inteligentes de control y monitoreo de las plantas industriales, empleando análisis y automatización de procesos mediante una red segura de datos permitirá controlar la cadena de valor de un producto: desde su invención hasta la etapa de producción y post-producción (cadena de valor inteligente).
- Monedas Inteligentes: El uso de sistemas digitales como modelos de monetización no es solo una moda, sino una necesidad donde las transacciones económicas son inmediatas, así como deben ser seguras empleando sistemas de autenticación, encriptación y control de integridad como *BlockChain*. Bitcoin (Bitcoin, 2020), según algunos entendidos del tema, será la moneda del futuro. Hay que mencionar también que nuevas monedas electrónicas están surgiendo, por ejemplo, Libra, la criptomoneda creada por Facebook (Coppola, 2019), la cual junto con Calibra, su sistema monetario o "billetera digital", intentaron revolucionar las finanzas, por ello, los gobiernos muestran ya su preocupación por la falta de estandarización y legislación adecuada en este campo (Infobae, 2019).

Desde un punto de vista más técnico, la digitalización implica conectar personas y cosas, pero dando importancia y valor a los datos que surgen de esa conectividad. El fundamento de esa conexión extrema es la red; sin embargo, las empresas y organizaciones involucradas en mantener y transportar los datos, el bien intangible más importante hoy en día (Salazar & Chafla, 2015), han incluido mecanismos para adaptarse a estas tecnologías emergentes no con la misma velocidad y desarrollo que las demandas de tráfico actual, por ese motivo la disrupción tecnológica desplazará a aquellos negocios que no se adapten a estos cambios, incluso a proveedores de servicio de Internet (ISPs) que no migren sus infraestructuras a unas más modernas.

Un hecho que cambió el mundo tal como lo concebimos fue la pandemia de COVID-19[2], hecho que permitió visualizar las dificultades que ciertos modelos de negocio, educación y salud han tenido y tienen para adoptar la transformación digital, mostrando la necesidad de cambios culturales para dar continuidad y estabilidad económica, no solo en esas áreas, sino también en el resto de las actividades humanas dependientes de tecnología.

Las industrias y empresas han pagado millones de dólares debido a la revolución digital en la última década, es más, los gerentes generales (CEOs) tienen muy presente este cambio y consideran una inversión obligatoria el mejorar sus infraestructuras de telecomunicaciones, pues si no se adaptan, sucumbirán ante la competencia que utilice la información generada en sus redes y cree nuevo conocimiento para liderar sus negocios, pero, ¿Es realmente necesaria una inversión tan alta en las infraestructuras? La respuesta a esa pregunta dependerá de la aproximación empresarial tecnológica a los objetivos planteados por la compañía. Gartner ha planteado un conjunto de buenas prácticas y guías a tomar en cuenta por el BoD (*Board of Directors*) (Gomolski, 2019) y llegar a la tan anhela transformación digital, sin embargo, bajo el contexto e impacto de COVID-19, trajo consigo desafíos socioeconómicos nunca pensados, impactando directamente en el grado de preparación que tienen las infraestructuras tecnológicas para afrontar estos inconvenientes. Un estudio propuesto por el Observatorio CAF (Banco de Desarrollo de América Latina) del Ecosistema Digital, presentó publicaciones con base de evidencia empírica sobre la posición y oportunidades de Latinoamérica en términos de digitalización, dando resultados alarmantes:

> En particular, se identifica una disminución de velocidad de banda ancha fija en Chile (-3%) y Ecuador (-19,6%), combinando esto con un incremento de la latencia en la misma tecnología en Brasil (11,7%), Chile (19,0%), Ecuador (11,8%) y México (7,4%) … El propósito de este trabajo es evaluar cómo está América Latina posicionada y cómo la digitalización puede jugar un papel fundamental en mitigar los efectos de la pandemia. En efecto, el análisis del impacto econométrico del primer virus SARS-Cov, demostró que los países con una infraestructura de conectividad desarrollada pudieron mitigar en un 75% las pérdidas económicas asociadas con la pandemia (CAF - Banco de Desarrollo de América Latina, 2020)

Es importante mencionar que, por más de treinta años, las redes y las infraestructuras de TI (Tecnologías de la Información) han evolucionado, comenzado por una conexión de un par de computadoras, hasta hoy en día, donde varios sitios remotos pueden conectarse entre sí, uniendo compañías, ciudades y países enteros. Esto trajo consigo una demanda de tráfico de más de un Zettabyte para el 2017, con una proyección de 4.8ZB para el 2022 (Cisco Systems, 2019), mientras que para el año 2021 se espera existan cerca de 30 billones de dispositivos interconectados (Hanes et al., 2017), con millones de conexiones M2M o "*Machine-to-Machine*". La digitalización está en su etapa de despliegue y sin lugar a duda las infraestructuras de red tendrán un rol importantísimo para que eso ocurra, rompiendo paradigmas y así posibilitar la hiper-conectividad.

[2] COVID-19: El Director General de la Organización Mundial de la Salud (OMS), el Doctor Tedros Adhanom Ghebreyesus, anunció el 11 de marzo de 2020 que la nueva enfermedad por el coronavirus 2019 (COVID-19) puede caracterizarse como una pandemia. La caracterización de pandemia significa que la epidemia se ha extendido por varios países, continentes o todo el mundo, y que afecta a un gran número de personas.: Link: https://www.paho.org/es/tag/enfermedad-por-coronavirus-covid-19

Diseños estructurados de redes que contemplen características como jerarquía, modularidad, resiliencia y flexibilidad se requerirán para dar soporte a los negocios. Es justamente ahí donde el concepto de redes inteligentes y programables nace (Salazar & Chafla, 2015).

Una red inteligente y programable es aquella que posee un conjunto de controles de tráfico y características de flexibilidad que involucran a la LAN, WAN y *Data Center* (DC) para un adecuado envío de datos mediante la orquestación y toma de decisiones empleando marcaje dinámico de paquetes, técnica usada en modelos de Calidad de Servicio (QoS) de próxima generación.

La siguiente evolución del *networking* es el concepto de red centralizada *open source/*interoperable en lugar de red distribuida tradicional, donde el controlador es quien tomará las decisiones de enrutamiento y conmutación definidas en *scripts* a través de lenguajes de programación como Python, Java, P4, Bash, C++ o cualquier lenguaje que el administrador de la red desee. Estos *scripts*, que en palabras más simples generan aplicaciones de *software*, se muestran bajo APIs o *Application Programming Interfaces*, diseñadas para enviar señales al controlador según los requerimientos planteados por el usuario a través de un tipo de formato de intercambio de datos como JSON, YAML o XML. Una API muy usada y que perfectamente se adapta a esos formatos son las REST-APIs (*Representational State Transfer APIs*). Este cambio se da bajo el concepto de SoC (*Separation of Concerns*) traído del mundo del *software* y de IoT, concepto que facilita el diseño, interacción y resultado de un proyecto/programa/infraestructura. Cabe decir que en la actualidad existe una tendencia evolutiva de cambio de interfaces de configuración entre el usuario y la infraestructura de igual manera. Tradicionalmente se usa CLIs (*Command Line Interfaces*) que emplean formato estructurado de configuración, pero, para administrar los cambios y necesidades modernas de una forma más adecuada y efectiva, es necesario el uso de datos estructurados con formatos como los mencionados anteriormente, junto con modelos como YANG para su procesamiento.

Ciertas implementaciones y desarrollos tecnológicos, por otro lado, traen el concepto de redes *Underlay* y *Overlay* a través de DMPVN, BGP, VXLAN, MPLS-VPN, EVPN y mecanismos modernos de encapsulación y etiquetado como MPLS *Segment-Routing* y LISP, permitiendo un crecimiento óptimo y escalable de la infraestructura ya sea usando el Internet, redes celulares de última generación o enlaces de bajo costo sin comprometer el desempeño, la confiabilidad y seguridad, manteniendo una disponibilidad de "cinco nueves"[3] para dar lugar a las redes inteligentes y programables; para estas redes, desde el lado del proveedor se acuñó el nombre de *intelligent WAN,* SD-WAN o WAN definido por *Software*.

Una *iNetwork (Intelligent Network)* dinámicamente enrutará el tráfico priorizándolo según el uso de las diversas aplicaciones, dispositivos finales y condiciones de red para tener la mejor experiencia de usuario posible sin desfinanciar a una empresa que cuenta con una red ya implementada, generando un pronto retorno de inversión (ROI) en caso de que se tenga que realizar alguna actualización obligatoria.

Las Redes Definidas por Software o SDN por sus siglas en inglés (*Software-Defined Networks*) cambiarán el presente y futuro de las redes pues, rompe paradigmas tradicionales, pero con el

[3] 99.999% de disponibilidad en la red – 5.26 minutos de caída anual máximo.

objetivo de mejorar el desempeño y la experiencia del usuario adaptándose a los cambios tecnológicos demandados.

SDN y el concepto de *iNetwork* son muy compatibles entre sí, quizá el segundo es paso previo a la implementación completa del primero en una red de producción, pues dentro de las características de redes inteligentes y programables se conceptualiza a un Controlador Máster que centralizará las operaciones de configuración y/o gestión.

Es necesario emplear estándares definidos y especificar unos nuevos para que realmente SDN sea viable como mecanismo de comunicación en las redes modernas y ahí surgen nuevos desafíos para los profesionales en Tecnologías de la Información, desafíos que serán abordados en la presente investigación, así como trabajos futuros en temas de seguridad, escalabilidad y flexibilidad en este tipo de entornos, acuñando el concepto que Cisco Systems trajo en el 2020, *User Defined Networking* o Redes definidas por el usuario (Cisco Systems - User Defined Network, 2020), redes que serán ágiles bajo el paraguas de *NetDevOps*.

1.1 Publicaciones Científicas

A continuación, se muestran diversos artículos científicos y Conferencias (publicados e indexados en revistas de renombre, la mayoría en inglés) escritos y dictados por el autor de la tesis, los cuales dan soporte a la presente investigación doctoral:

- G. D. Salazar-Chacón and A. R. Reinoso García, "*Segment-Routing Analysis: Proof-of-Concept Emulation in IPv4 and IPv6 Service Provider Infrastructures*," 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), 2021, pp. 1-7, doi: 10.1109/IEMTRONICS52119.2021.9422559.
- A. Gordón, and G. Salazar-Chacón, "*DRP Analysis: Service Outage in Data Center due to Power Failures*," 2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2020, pp. 0182-0187, doi: 10.1109/IEMCON51383.2020.9284920.
- G. D. Salazar-Chacón and L. Marrone, "*OpenSDN Southbound Traffic Characterization: Proof-of-Concept Virtualized SDN-Infrastructure*," 2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2020, pp. 0282-0287, doi: 10.1109/IEMCON51383.2020.9284938.
- J. E. Vaca P. and G. D. Salazar-Chacón., "*VXLAN-IPSec Dual-Overlay as a Security Technique in Virtualized Datacenter Environments*" 2020 IEEE ANDESCON, 2020, pp. 1-6, doi: 10.1109/ANDESCON50619.2020.9272160.
- G. Salazar-Chacón, E. Naranjo and L. Marrone. (2020). "*Open networking programmability for VXLAN Data Centre infrastructures: Ansible and Cumulus Linux feasibility study*". Revista Ibérica de Sistemas e Tecnologias de Informação, (E32), 469-482.
- G. Salazar, "*Ansible y SDN en acción: Los pilares de la Era de la Programabilidad*", Cisco Community Support, 26 de noviembre, 2019 [online]. Disponible: https://youtu.be/m0bSM8Xv10g

- G. Andrade-Salinas, G. Salazar-Chacon and L. M. Vintimilla. (2019). "*Integration of IoT Equipment as Transactional Endorsing Peers over a Hyperledger-Fabric Blockchain Network: Feasibility Study*". In International Conference on Applied Technologies (pp. 95-109). Springer, Cham.
- G. D. Salazar Ch., C. Hervas, E. Estevez and L. Marrone, "*High-Level IoT Governance Model Proposal for Digitized Ecosystems*," 2019 International Conference on Information Systems and Software Technologies (ICI2ST), 2019, pp. 79-84, doi: 10.1109/ICI2ST.2019.00018.
- G. D. Salazar Ch, C. Venegas, and L. Marrone, "*MQTT-Based Prototype Rover with Vision-As-A-Service (VAAS) in an IoT Dual-Stack Scenario*," 2019 Sixth International Conference on eDemocracy & eGovernment (ICEDEG), 2019, pp. 344-349, doi: 10.1109/ICEDEG.2019.8734341.
- G. D. Salazar Ch., E. F. Naranjo and L. Marrone, "*SDN-Ready WAN networks: Segment Routing in MPLS-Based Environments*," 2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), 2018, pp. 173-178, doi: 10.1109/UEMCON.2018.8796613. – *Best Paper – Columbia University.*
- G. D. Salazar Ch., C. Venegas, M. Baca, I. Rodríguez, and L. Marrone, "*Open Middleware proposal for IoT focused on Industry 4.0*," 2018 IEEE 2nd Colombian Conference on Robotics and Automation (CCRA), 2018, pp. 1-6, doi: 10.1109/CCRA.2018.8588117.
- E. F. Naranjo and G. D. Salazar Ch, "*Underlay and overlay networks: The approach to solve addressing and segmentation problems in the new networking era: VXLAN encapsulation with Cisco and open source networks*," 2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM), 2017, pp. 1-6, doi: 10.1109/ETCM.2017.8247505.
- G. Salazar, "*Direccionamiento IPv6 - Bases y Fundamentos*" Cisco, 02 Febrero 2016. [online]. Disponible: https://supportforums.cisco.com/blog/12914981/direccionamiento-ipv6-bases-y-fundamentos
- G. Salazar, "*Fundamentos de QoS-Calidad de Servicio en Capa 2 y Capa 3*", 2016, [online] Disponible: https://community.cisco.com/t5/blogsrouting-y-switching/fundamentos-de-qos-calidad-de-servicio-en-capa-2-ycapa-3lba-p/3103715
- G. Salazar Chacón y G. Chafla Altamirano. (2015). "*Empleo de Path-control Tools en una red empresarial moderna mediante Políticas de Enrutamiento*". 3C Tecnología. Glosas De Innovación Aplicadas a La Pyme, 4(1), 1-18. Recuperado a partir de http://ojs.3ciencias.com/index.php/3c-tecnologia/article/view/233
- G. Salazar, "*DMVPN Fase1 y 2 en IPv4 Fundamentos y Configuración básica enfocado al CCIE RS*", 2017, [online] Disponible: https://community.cisco.com/t5/videos-routing-y-switching/dmvpn-fase1-y-2-en-ipv4-fundamentos-y-configuraci%C3%B3n-b%C3%A1sica/ba-p/3104173
- G. Salazar, "*Fundamentos de IP Multicast Routing y sus Modos de Operación: Demo en Vivo*", 2017, [online] Disponible: https://www.youtube.com/watch?v=xKD_Vppf8co

1.2 Objetivos y Aporte de la Investigación

Objetivo General:
Comprobar la factibilidad de implementación de una red definida por software (SDN) en ambientes híbridos IP-SDN donde la automatización de flujos de datos en base a las aplicaciones, visibilidad total, así como la programabilidad y rapidez son necesarios, más aún cuando la digitalización y Cloud Computing marcarán el presente y futuro de los negocios, la salud, la industria y las telecomunicaciones en general.

Objetivos Específicos:

- Identificar las necesidades y requerimientos de usuarios y empresas actuales, evaluando su impacto en las infraestructuras tradicionales.
- Identificar los avances tecnológicos, procedimientos y protocolos para adaptar las redes tanto a nivel LAN y WAN a la era del Internet de las cosas, *Cloud-Fog Computing* y la digitalización.
- Definir la Programabilidad, Automatización y Orquestación en las Redes y su relación con la Inteligencia Artificial, Cloud, IoT y SDN.
- Definir y analizar el concepto de SDN y SD-WAN como propuestas de continuidad a los negocios conformados por redes de datos masivos.
- Establecer los beneficios de las Redes Definidas por Software para permitir comunicaciones empresariales consistentes.
- Entender la arquitectura, topologías SDN y su operación mediante *OpenFlow, OpenDaylight* y sus diversos estándares.
- Comparar a SDN con otras formas de virtualización como NFV y Redes *Underlay-Overlay* (VXLAN, LISP y Segment-Routing).
- Determinar la posibilidad de integración e interoperabilidad de redes tradicionales IP con Redes SDN mediante automatización/programabilidad y su impacto en las redes tanto empresariales como de proveedores de servicio (SD-WAN).
- Diseñar y Simular una red prototipo SDN y SD-WAN mediante un software de emulación de redes avalado por la industria y la comunidad educativa.
- Implementar una red prototipo SDN, así como implementar la programabilidad en Redes Tradicionales y confirmar la factibilidad de su integración en equipos reales que soporten esta tecnología.

Aporte de la Investigación
Realizar pruebas de concepto sobre SDN, automatización y redes programables, así como de telemetría, *OpenNetworking* y nuevos protocolos de transporte, planteando además modelos innovadores de interoperabilidad entre redes tradicionales y redes definidas por *software* son los aportes principales de esta investigación, pues permitieron llegar a las conclusiones finales luego de un análisis profundo de los resultados obtenidos en la fase de simulación/emulación y pruebas en equipos reales, dando lugar a una red híbrida IP-SDN donde no necesariamente se tienen los equipos robustos o flujos de tráfico tan altos como en las redes de un Centro de Datos SDN (SD-DC).

Durante el desarrollo de la tesis, se emplea *software* de emulación de redes avalado por la comunidad universitaria, la academia y por empresas fabricantes de tecnología en redes, empresas que basan sus procesos en ciclos de vida como *DevOps*, Lean-Agile, ITIL, FCAPS y

PPDIOO, donde el prototipado de las tecnologías es fundamental para el éxito de las industrias de telecomunicaciones contemporáneas.

Una vez verificada la factibilidad de los conceptos y protocolos planteados en la investigación mediante *software* emulador, la fase final del desarrollo de esta tesis doctoral comprueba el comportamiento de SDN en equipos físicos, aterrizando los conceptos de programabilidad y SDN a la realidad.

Sin lugar a duda, la experimentación es clave para cualquier investigación científica, ya que traslada los conceptos teóricos a la práctica, dando así un impacto transcendental a la presente tesis doctoral, totalmente adecuada al método científico experimental y a la metodología heurística aplicada.

1.3 Contexto y Estado del Arte de las Redes de Información y Datos: Transformación o Mutación Digital

Las redes de datos han pasado de simplemente unir dispositivos electrónicos como computadoras, laptops, teléfonos IP, a conectar elementos biológicos y lo que tradicionalmente no es conectable, con el fin de digitalizar el entorno, maximizando de esa manera las ventajas que una infraestructura de datos puede dar más allá de sólo romper las barreras geográficas de comunicaciones; pero con ello también aumentaron los riesgos, problemas y nuevos requerimientos que deben ser solucionados con tecnologías emergentes.

Llegar al punto donde nos encontramos en términos de conectividad ha sido un camino evolutivo que se ha convertido en una revolución innovadora en los últimos años.

Los sistemas de comunicaciones, desde las primeras civilizaciones con las señales de humo, mensajeros a pie y a caballo, señales acústicas, visuales y el telégrafo hidráulico en el siglo IV a.C. han tratado de comunicar a los seres humanos a larga distancia. Tuvieron que pasar muchos años para lograr avances significativos en las comunicaciones, por ejemplo la instalación del telégrafo óptico en la Edad Contemporánea en 1684, implementación de la red telegráfica eléctrica en 1866, desarrollo del TDM (*Time Division Multiplexing*) gracias a los estudios de Emile Baudot y Tomás Edison a finales del siglo XIX, patente del teléfono en 1876 por Alexander Graham Bell, bases de la Teoría de la Información por parte de Claude Shannon, Harry Nyquist y Ralph Hartley a inicios del siglo XX dieron los cimientos para el origen de los sistemas de trasmisión de datos contemporáneos, nacimiento que se confirmó con el desarrollo del primer organismo de estandarización en las telecomunicaciones denominado ITU (*International Telecommunication Union*) adscrita a la ONU en 1947. El siguiente eslabón evolutivo fue la creación del módem en la década de los 60s en los laboratorios Bell, dando lugar a la era de la Conmutación de Paquetes definida por Leonard Kleinrock en el MIT y modelado bajo el *Stack* TCP/IP de Robert Kahn y Vinton Cerf en 1975; paso a paso dieron cabida al nacimiento de las telecomunicaciones tal como las conocemos hoy, desarrollando ARPANET, el embrión de Internet en los 70s, construido gracias a los avances e investigaciones del Departamento de Defensa de Estados Unidos (DoD) y ARPA (*Advanced Research Projects Agency*).

La evolución en los medios de transmisión también ha favorecido esta revolución, por ejemplo, los cables submarinos, en un inicio para comunicar países europeos (Gran Bretaña con el resto

de Europa) mediante el telégrafo y teléfono, hoy ya con fibra óptica que recorre la Tierra (TeleGeography, 2020), están otorgando la capacidad requerida de grandes anchos de banda en cualquier lugar del mundo, siendo además un cableado que sigue su expansión.

ARPANET se convierte en Internet y se da un boom comercial a partir de ahí, muchos de los requerimientos empresariales dependen del ciberespacio y se da comienzo a la hiper-conectividad del ser humano, modificando la forma en que vive, aprende, trabaja, cuida de su salud, se divierte y se comunica. Por todo ello y en vista de la necesidad de más direcciones para identificar todos los dispositivos interconectados, así como para mejorar errores de despliegue y seguridad de IPv4, en junio del 2011, el *Internet Society* (ISOC) lleva a cabo el *World IPv6 Day* (Internet Society, 2020) y en junio del 2012 oficialmente se lanza IPv6.

El Internet se ha convertido en una tecnología WAN ideal, debido a su bajo costo, escalabilidad, flexibilidad y adaptabilidad, lo que ha impactado la manera en que las empresas conectan sus sedes, eliminado sus principales debilidades: la seguridad y bajo ancho de banda.

Las fases evolutivas del Internet las podemos apreciar en el siguiente cuadro:

Tabla 1-1 Fases evolutivas del Internet

Fase	Descripción
Fase 1: Conectividad (Acceso Digital)	Esta fase conecta las personas a servicios como e-mail, la web, así como a búsquedas de información.
Fase 2: Economía conectada (Negocio Digital)	Esta fase habilita el comercio electrónico y mejoras en la cadena de distribución de productos, junto con la colaboración en los negocios.
Fase 3: Experiencias Inmersivas (Interacciones Digitales)	Esta fase extiende la experiencia del Internet junto con la inclusión de video, redes sociales, movilidad. Nacimiento del Cloud.
Fase 4: Internet de las Cosas (Mundo Digital)	Esta fase añade conectividad a los objetos y máquinas en el mundo que nos rodea, así como habilita nuevos servicios y experiencias. Conectar lo no conectado de forma segura.

Fuente: (Hanes et al., 2017)

Actualmente nos encontramos entre la fase 3 y fase 4 del Internet.

En general, "las tecnologías de la Información y las comunicaciones están evolucionando hacia un modelo de consumo en la nube" (Salazar Ch & Naranjo, 2017), lo que también implica un rediseño de las arquitecturas en los DCs, donde la flexibilidad y movilidad, resiliencia, servicios bajo demanda, analítica de datos, capacidad *multitenant*, microsegmentación, así como excelente desempeño y escalabilidad son aspectos que deben ser tomados en cuenta en diseños de redes modernos.

Como un resultado claro de ello, los DCs están evolucionando de su diseño jerárquico tradicional tipo *Fat-Tree* a uno denominado *Spine-Leaf* (ver *Figura 1-1*), dando así una aproximación mucho más simple, ágil, con soporte de tráfico *east-weast* y *north-south* de manera más eficiente (Salazar-Chacón & Vaca, 2020).

Según (Salazar Ch & Naranjo, 2017), estas son las razones primordiales para desarrollar soluciones con SDN, redes híbridas o al menos redes tradicionales que tengan características de *Open-Networking,* automatización y programabilidad (*NetDevOps*).

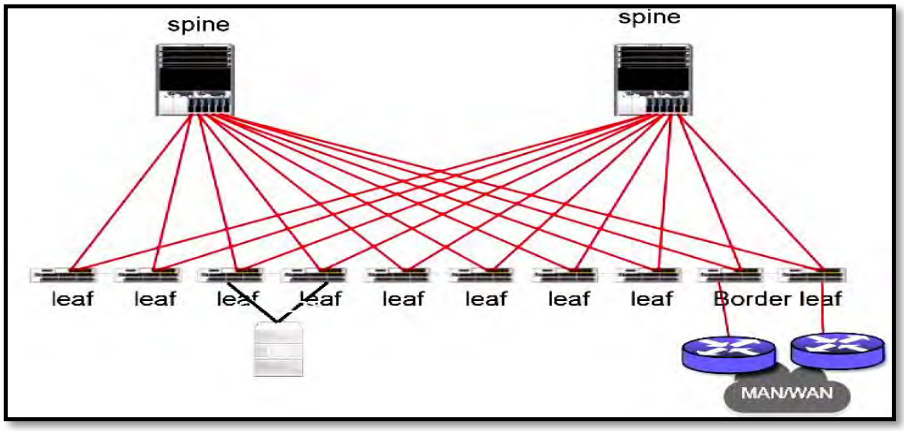


Figura 1-1 Arquitecturas Spine-Leaf en DCs modernos.
Recuperado de (Salazar Ch & Naranjo, 2017)

Los centros de datos modernos se encuentran en la tercera generación, tal como se aprecia en la *Figura 1-2*. Las generaciones evolutivas en los DCs comienzan con los *mainframes*, siendo éstos los de 1ra. Generación, el siguiente eslabón es el modelo Cliente-Servidor y computación distribuida como 2da. Generación, y la 3ra. Generación, según (Bruno & Jordan, 2011) es aquella que está definida por los siguientes componentes:

- Virtualización: Abstracción de los Sistemas Operativos del *Hardware*. Computación en la nube depende de la Virtualización, donde la abstracción separa el *hardware* de las aplicaciones (SaaS, PaaS e IaaS[4] y demás entonos como servicios).
- *Unified Fabric:* Medios Físicos y recursos de *networking* que dan soporte al altísimo ancho de banda. Ejemplos son *Fibre-Channel over Ethernet* (FCoE).
- *Unified Computing:* Plataformas computacionales convergentes de próxima generación, tanto a nivel de procesamiento como memoria.
- Hiper-convergencia: Almacenamiento, *Backups* y flexibilidad transparente para el usuario.

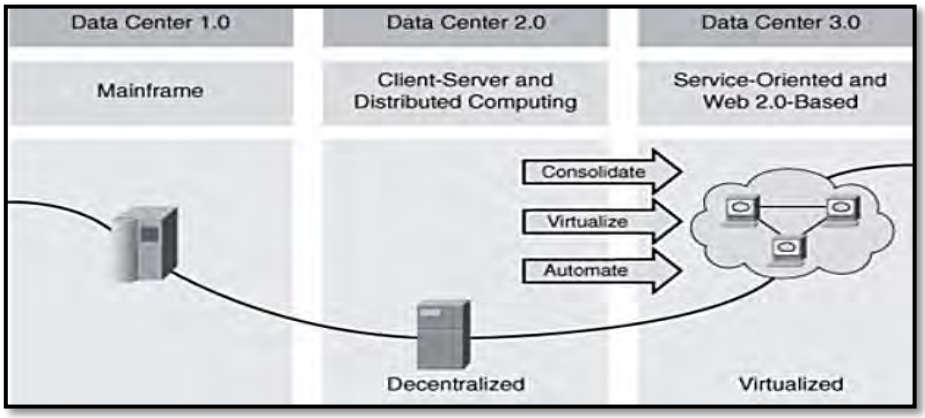


Figura 1-2 Evolución DCs.
Recuperado de (Bruno & Jordan, 2011)

[4] SaaS – Software as a Service; PaaS – Platform as a Service; IaaS – Infrastructure as a Service

De acuerdo con (Bruno & Jordan, 2011), el DC 3.0 fomenta el pronto retorno de inversión (ROI) y disminuye el costo total de propiedad (TCO).

Varios fabricantes de *hardware* y *software* como Microsoft, Schneider Electric y Cisco Systems nombran un eslabón adicional basado en la centralización de las aplicaciones, rapidez de configuración, visibilidad y seguridad total, así como programabilidad, orquestación y optimización automática, eslabón totalmente relacionando con la Industria 4.0 y nueva generación de redes de datos, tal como se aprecia en la *Figura 1-3*.

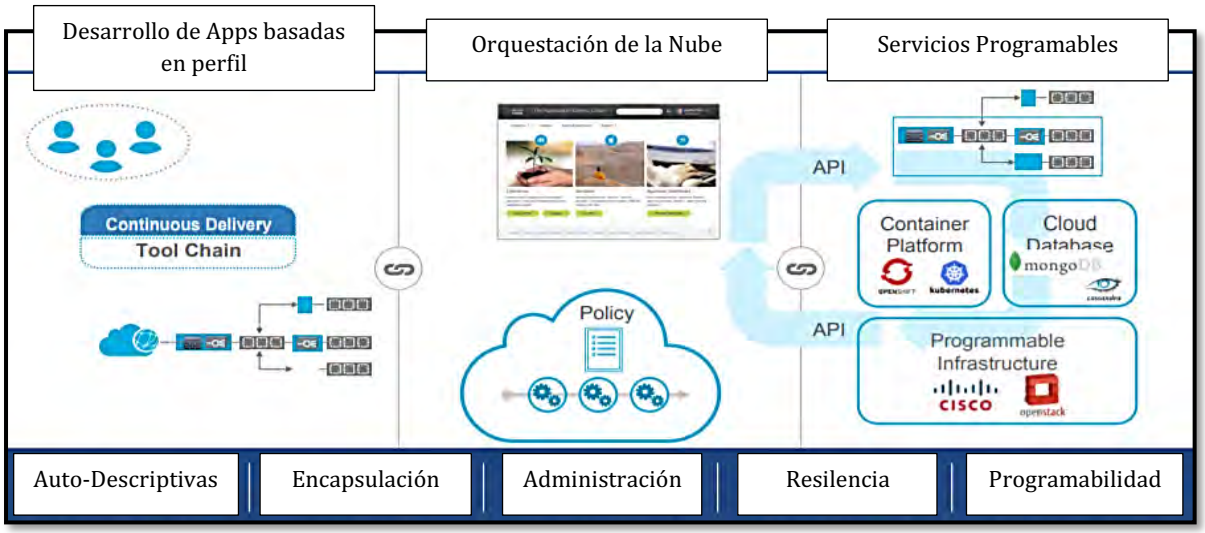


Figura 1-3 Data Center de Nueva Generación.
Recuperado de (Manville, Woolwine, & Benny, 2019)

La infraestructura de redes de comunicaciones que da cabida a estos cambios tecnológicos también debe adaptarse a la hiperconectividad y DCs de nueva generación, por ello, muchos protocolos y tecnologías de transporte surgieron a la par de las necesidades del usuario final.

La *Figura 1-4* muestra esa evolución, marcando a MPLS (*Multiprotocol Label Switching*) como punto de inflexión en la evolución de las redes de transporte de datos WAN, mientras la *Fig. 1-5* muestra una topología de SD-WAN, presente y futuro de las Redes de Transporte.

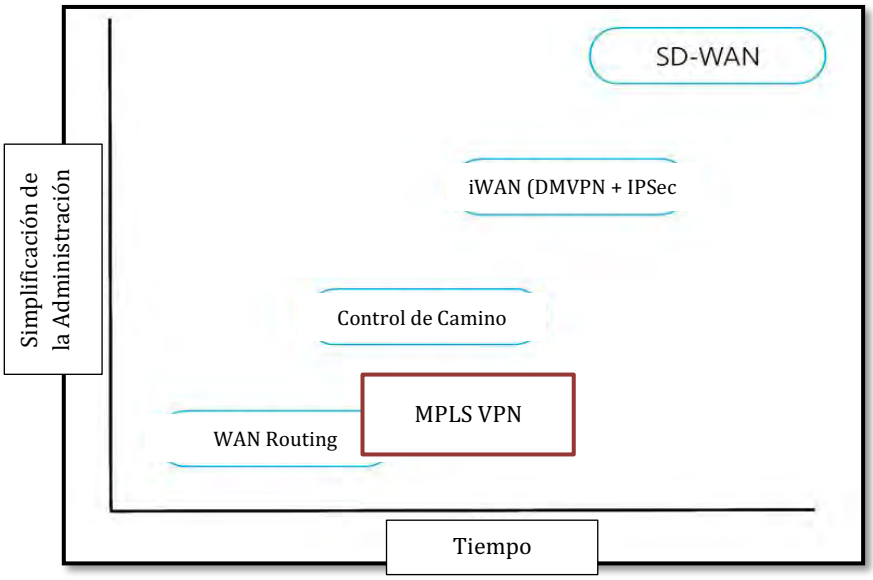


Figura 1-4 Evolución de las Redes WAN
Fuente: *Autor*

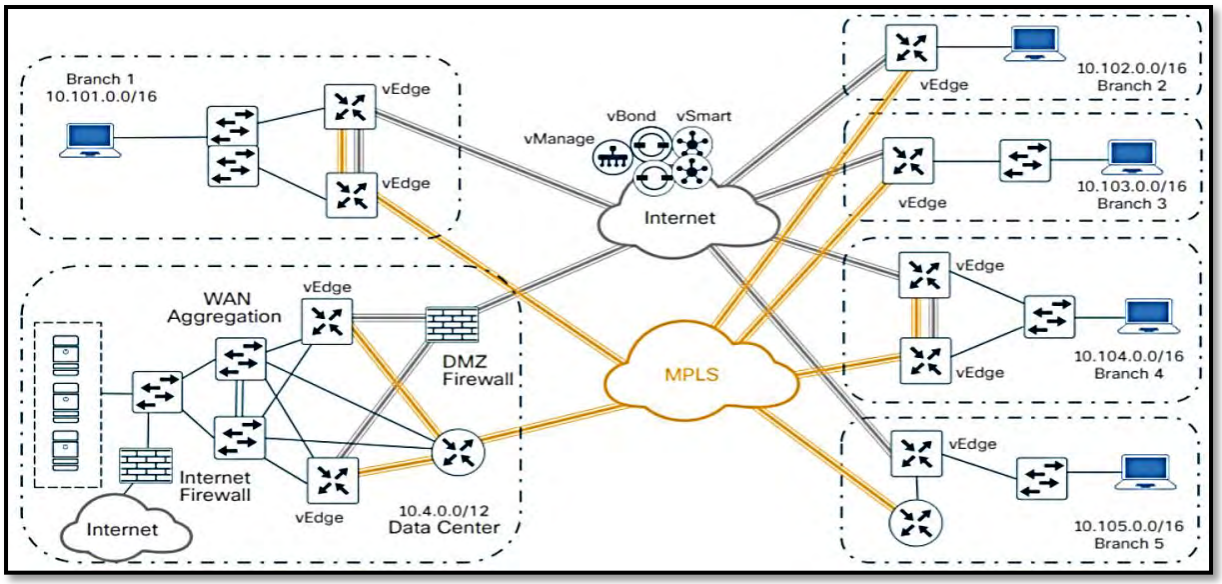


Figura 1-5 Infraestructura SD-WAN.
Recuperado de *(Cisco Systems, 2019)*

Las redes modernas deben tener ciertas características que complementen el criterio tradicional de escalabilidad, tolerancia a fallas, QoS y seguridad, pues los requerimientos de los usuarios y de las empresas así lo demandan.

Además de esas características básicas, deben tener:

- Resolución **automática** de fallas y errores.
- Defensa **automática** ante ataques.
- Optimización **automática.**
- Análisis **automático** de datos para una toma de decisiones adecuada.
- Visibilidad completa de la red mediante ***Insights***.

El concepto de automático significa dotar de inteligencia y facilidad de manejo a las redes para que puedan adaptarse al entorno y de esa manera cubrir los requerimientos de IoT, *BigData*, BYOX, Conectividad 5G/6G, teletrabajo, *streaming* de video masivo, realidad virtual, entre otros.

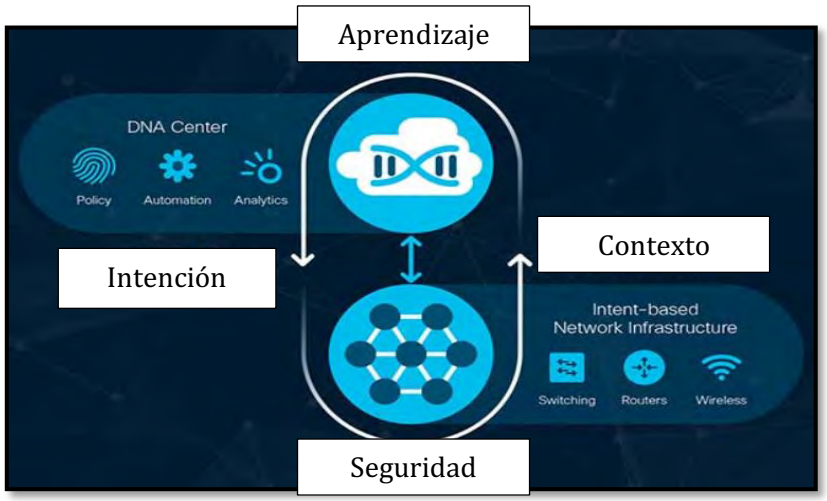


Figura 1-6 Redes Basadas en la Intención.
Recuperado de *(Cisco Systems, 2019)*

Debido a que las redes requieren de un mejor entendimiento del contexto para generar visibilidad total y demás funciones importantes en las redes actuales, la importancia de los datos toma un nivel superlativo y con ello, el avance en el proceso de digitalización va de la mano de esa evolución.

La digitalización da una importancia relevante a los datos, es más, todo elemento que forma parte de una red podrá generar algún tipo de información.

Según Juniper Networks, el tráfico de datos se incrementará en 9.6 veces para el año 2025, conectando una cantidad de más de 30 billones de dispositivos para el 2020-2021 y cerca de 80 billones para el 2025 (Wexler, 2016). Más que seguro, esa expectativa se quedará corta debido a los requerimientos de conectividad generados por la pandemia COVID-19.

Por otro lado, la transformación digital va más allá de la digitalización, pues no solo es una nueva tendencia tecnológica, es en realidad una estrategia empresarial y cambio cultural muy rentable que involucra a la innovación de los mercados y giros de negocio, pensando en el beneficio de los usuarios finales y de la empresa.

Bob Parker, vicepresidente de *IDC Research*, indicó que al menos el 55% de las organizaciones usarán plataformas tecnológicas digitales, las mismas que habilitarán nuevos productos y servicios digitales, impactando enormemente en las Tecnologías de la Información mundial. (Parker & Shawn, 2018). En dicho evento de IDC, se llegó a la conclusión que, soportado por plataformas tecnológicas móviles-sociales, junto con analítica y *cloud*, la transformación digital representa una oportunidad de redefinición de la experiencia de clientes, logrando nuevos niveles de productividad empresarial.

Gracias a la digitalización, los datos pueden ser fácilmente accesibles a través de diversas plataformas, equipos e interfaces. Ahora es momento de dar valor e importancia a cada uno de esos datos generados, provocando así una gran rentabilidad y nuevas plazas de trabajo.

Ejemplos claros de que la digitalización es el nuevo "oro negro" de la economía, son las grandes empresas que basan su rentabilidad en el uso de los datos de sus clientes con el fin de entregar un servicio, tal como se aprecia en la *Figura 1-7*.

Figura 1-7 Transformación Digital.
Recuperado de *(Boorsma, 2016)*

Para llegar a la digitalización y transformación digital, IoT tiene un sitial especial, pues está modificando la vida del ser humano, así como del mundo empresarial también, razón por la cual es mencionado y analizado en esta introducción.

Si bien existen muchas definiciones para el Internet de las Cosas, la más adecuada es la que se acuñó por Kevin Ashton en 1999. Ashton mencionó que IoT se refiere a la interconexión digital de objetos simples que se encuentran en la cotidianidad. Ashton realizó las primeras pruebas e investigaciones empleando etiquetas de radio RFID, dando así el primer paso para redes D2D (*Device-to-Device*) y M2M para el sector industrial.

Se tiene la idea de que cada persona está rodeado de entre 1000 a 5000 objetos que podrían conectarse y según una investigación de *ABI Research*, para el 2020 habrá más de 30 billones de cosas conectadas a Internet (ABI Research, 2013), tal como se mencionó anteriormente.

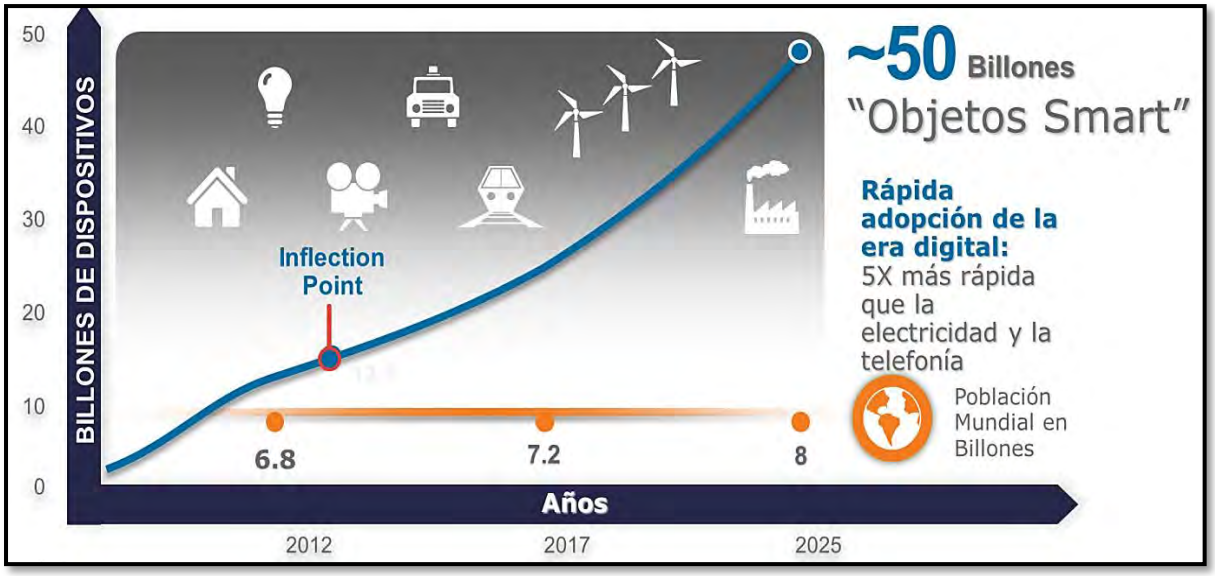


Figura 1-8 Adopción de Internet de las Cosas (IoT)
Fuente: *Autor*

A pesar de ser clara esa definición, IoT ha tenido aplicaciones que van más allá de M2M o D2D, en realidad, hoy se incursiona en la programabilidad y en el valor de los datos, por lo que el IoT moderno trata de la interconexión de Personas, Objetos, Datos y Procesos automatizados.

Según (Salazar, Venegas, Baca, & et-al, 2018), la Industria 4.0 no es solo alta tecnología, comunicaciones M2M y digitalización, pues además representa una manera de diseñar e implementar procesos dentro de la compañía, procesos que deben estar acompañados de una adecuada analítica y capacitación del talento humano y es ahí donde *BigData* o MacroDatos toma importancia, cambiando el contexto a "Mutación Digital".

BigData es un término muy escuchado y se refiere a la adquisición y procesamiento de un gran volumen de datos, ya sean estos estructurados o no estructurados, los cuales se generan tanto de nuevas tendencias tecnológicas como la Digitalización, IoT, o de la automatización de procesos convirtiéndolos en inteligentes, así como de procesos digitalizados tradicionales también.

La importancia de *BigData* radica en brindar valor a estos datos recolectados más que únicamente a la masividad de información que procesa.

Si bien es cierto que el almacenamiento y procesamiento de datos es un término que proviene de algunas décadas atrás, *BigData* es un concepto relativamente nuevo, pues se lo acuñó gracias a los trabajos e investigaciones realizadas por Doug Laney en los primeros años de la década de los 2000s, dándole características básicas como las denominadas y bien conocidas tres Vs:

- **Volumen:** La masividad de datos es quizá la característica que hace reconocido a *BigData*, pues en efecto es así. Empresas, industrias, procesos y las actividades cotidianas del ser humano generan una gran variedad de volúmenes de datos de diversas fuentes, tanto de eventos transaccionales como información de uso de plataformas en línea, tendencias que ocurren en redes sociales, transacciones bancarias o también datos provenientes de comunicación D2D, P2M y M2M en procesos industriales o soluciones IoT. En el pasado, almacenar y procesar esta cantidad de datos hubiera sido un problema, pero hoy en día, con la creación de aplicaciones y el avance tecnológico para tratar este volumen de información tan grande han disminuido estas dificultades, un caso muy relevante es el del *Framework Hadoop*, cuya creación está íntimamente relacionada con la necesidad que tuvo Google de procesar e indexar la inmensa cantidad de información que posee años atrás y lo bueno de esta aproximación, es el hecho que nació en un entorno de software libre para *BigData*.
- **Velocidad:** Los flujos de datos que se adquieren en este tipo de implementaciones llegan con una velocidad muy grande, esto gracias a que muchos (la mayoría) de procesos son entornos reactivos en tiempo real, lo que implica que los procesos, por ejemplo, industriales o IoT, una vez que se capta el flujo, se debe tomar una decisión o realizar una acción mediante un actuador. Sistemas considerados *Real-Time*, son recurrentes en el mundo *BigData*.
- **Variedad:** En cada uno de estos flujos, los datos vendrán en diversos formatos, desde no estructurados, estructurados provenientes de una base de datos tradicional, datos numéricos, textuales hasta datos transacciones de servicios TCP o UDP.

Las empresas del futuro usarán todas estas herramientas para sostener sus negocios. Se mencionó la existencia de empresas que lo entendieron desde muchos años atrás y hoy dominan el mercado (*Figura 1-7*), pero ¿En realidad estas empresas son producto de una evolución progresiva o son mutaciones instantáneas que vieron su oportunidad en el mundo digital? La respuesta a esa interrogante planteada en (Gómez, 2017) la dio Aileen Lee, quien acuñó a este tipo de negocios denominados "Unicornio", como aquellas que lograban superar el valor de USD$1000 millones de dólares en sus etapas de levantamiento de capital, naciendo no como una evolución natural de empresas tradicionales, sino producto de una disrupción. Ejemplos existen en Netflix, MercadoLibre o Airbnb, quienes no nacieron de la evolución de Blockbuster, del comercio y de las cadenas hoteleras respectivamente, sino como un nuevo negocio basado en tecnologías disruptivas, que, en época de pandemia, incluso deben seguir innovando si no quieren desaparecer.

Según (Gómez, 2017), el secreto para que las empresas tradicionales triunfen, radica en desarrollar modelos de negocio innovadores que les permita competir con empresas que nacieron en el mundo digital.

El fundamento para la transformación digital en las empresas está en la evolución de su infraestructura de comunicaciones, una red digital enfocada a las programabilidad, simplicidad e inteligencia, red *Open-Source* que rompa los paradigmas tradicionales y justamente esas son, los pilares de las Redes Definidas por Software.

1.4 Estructura de la Tesis Doctoral

Una vez familiarizado con el contexto tecnológico y evolución de las redes de comunicaciones hacia las Redes programables y Definidas por Software (SDN), se plantea la estructura de la presente tesis doctoral, cuyo contenido se describe de la siguiente manera:

Capítulo I: Introducción

En este capítulo se define el estado del arte de las redes de información y datos, así como su impacto en la transformación digital, dando soporte a los nuevos requerimientos de los usuarios y empresas de siglo XXI. De la misma manera, se plantean los objetivos y aporte de la investigación de la presente tesis, junto con las publicaciones científicas realizadas por el autor en el marco del tema propuesto.

Capítulo II: Fundamentos de las Redes Definidas por Software

Se expone la conceptualización, beneficios, impactos en la industria, arquitectura, componentes básicos y tipos de controladores de una Red Definida por Software comparándolas mediante KPIs (indicadores de desempeño), además de entender la importancia de las APIs en el contexto amplio SDN, partiendo de la evolución en Eras del *Networking*, hasta el concepto de *OpenSDN, NG-SDN* y programabilidad en *hardware abierto* con Lenguaje P4.

Se define el concepto de *NetDevOps* y se plantea la necesidad de la programabilidad en las Redes Definidas por Software, así como el uso de *Frameworks* abiertos y de herramientas desarrolladas en *Python* que permitan que las redes tradicionales puedan convivir con las redes de nueva generación.

Finalmente, se plantea la relación de SDN con IoT, *Cloud Computing* y la Digitalización, para con ello entender el ecosistema SDN.

Capítulo III: Redes Híbridas: Infraestructura tradicional y SDN

SDN vista desde un entorno de túneles inteligentes y programables se plantea en este capítulo, solución aceptada en el mercado y desplegada en implementaciones de redes en la actualidad. Se define a *SD-Access, SD-DC y SD-WAN* junto con su integración a las redes tradicionales, teniendo a MPLS-LDP como punto de evolución en las redes de transporte de nueva generación y a VXLAN, LISP y *MPLS-Segment Routing* como tecnologías a implementar en el presente y futuro en los ISPs y DCs.

Se conceptualiza a SD-WAN Viptela como una solución práctica y efectiva. Se determina mediante PoCs su factibilidad de uso e implementación.

Capítulo IV: Simulación/Emulación de Redes SDN Híbridas

En este capítulo se emularán varios entornos en forma de PoCs (Pruebas de Concepto), entre ellos: *Segment Routing,* LISP como parte del entorno *SD-Access*, VXLAN bajo un ambiente programable con *Ansible*, así como una infraestructura SDN mediante Mininet, *Open Virtual Switches* (OVS) y dos tipos de controladores externos (*OpenDayLight* y ONOS). Previamente se dará una breve explicación de las herramientas de emulación usadas, así como de los controladores, máquinas virtuales y aplicaciones de programabilidad utilizados.

Para finalizar el capítulo, se desarrollan dos PoCs de ambientes modernos y comercialmente viables: SD-WAN Viptela y NG-SDN basado en ONOS, *switches Stratum* y controlador ONOS bajo programación del plano de datos en P4.

El análisis de factibilidad, capturas de tráfico y estudio de protocolos e intercambio de mensajes se generó dentro de cada PoC.

Capítulo V: Implementación de una Red Prototipo SDN

El capítulo V plantea una topología física real SDN empleando elementos como *Switches OpenFlow* y *OpenHardware* (*Northbound Zodiac FX y Zodiac WX)* para redes cableadas e inalámbricas, así como controladores virtualizados (RYU-*FlowManager* y *Aruba VAN SDN Controller*), planteando de esa manera infraestructuras SDN mediante dispositivos que se pueden encontrar en una red SMB (*Small-Medium Business*).

Los resultados de capturas de tráfico y pruebas de factibilidad se exponen al final del capítulo.

Conclusiones de la Investigación

Se exponen las conclusiones del presente trabajo investigativo una vez se ha pasado por planteamientos teóricos e investigación científica del tema, contrastando la teoría con la práctica mediante distintos métodos de experimentación, tanto en entornos emulados como físicos.

Estudios Futuros

Con el fin de cerrar el estudio de Infraestructuras SDN y su interacción con las redes tradicionales, se propuso los siguientes tópicos relevantes: Empleo de aplicaciones de Inteligencia Artificial para el monitoreo y visibilidad de una red SD-WAN masiva tipo *ThousandEyes*; comprensión del entorno SASE para controlar niveles de riesgos en ciberseguridad; y finalmente solucionar el problema de la Colocación de sedes distribuidas en ambientes *multicloud* (*CoLocation*) en entornos SD-WAN con tecnologías símiles a *Cloud-OnRamp*.

Acrónimos

Se enlistan los acrónimos utilizados en la presente tesis doctoral.

Bibliografía

Se enlistan los libros, artículos y revistas científicas, así como memorias, páginas web y blogs de relevancia en el ámbito tecnológico empleados en el presente trabajo doctoral.

Anexos

Este trabajo investigativo cuenta además con ocho (8) Anexos teórico-prácticos, mostrando documentos, métodos, programación e instrumentos investigativos en general, los cuales se usaron y/o crearon para sustentar el desarrollo de tesis doctoral.

Entre los anexos están:

- **Anexo A: Formatos y Estructuras de Datos para *NetDevOps* (JSON, YAML y XML)**
- **Anexo B: Guía de Instalación GNS3-VM, EVE-ng y Mininet**
- **Anexo C: NETCONF en la Práctica**
- **Anexo D: Ansible para entornos *NetDevOps* en infraestructuras de Red: VXLAN**
- **Anexo E: Pruebas de Concepto de Netmiko y Napalm – Telemetría**
- **Anexo F: Scripts para la Implementación de PoC NG-SDN**
- **Anexo G: Proceso de Instalación de SDWAN Viptela en EVE-ng**
- **Anexo H: Entrada en Funcionamiento de *Zodiac FX* y *Zodiac WX* – SDN en infraestructura física con *Aruba VAN SDN Controller y RYU-FlowManager***

Cabe decir que para llegar a las conclusiones de la investigación doctoral, se siguió el flujograma planteado a continuación, junto con el método científico y heurístico para comprobar la factibilidad de uso de las tecnologías mencionadas a través de pruebas de concepto con emuladores de grandes prestaciones e implementación de un prototipo con equipos físicos.

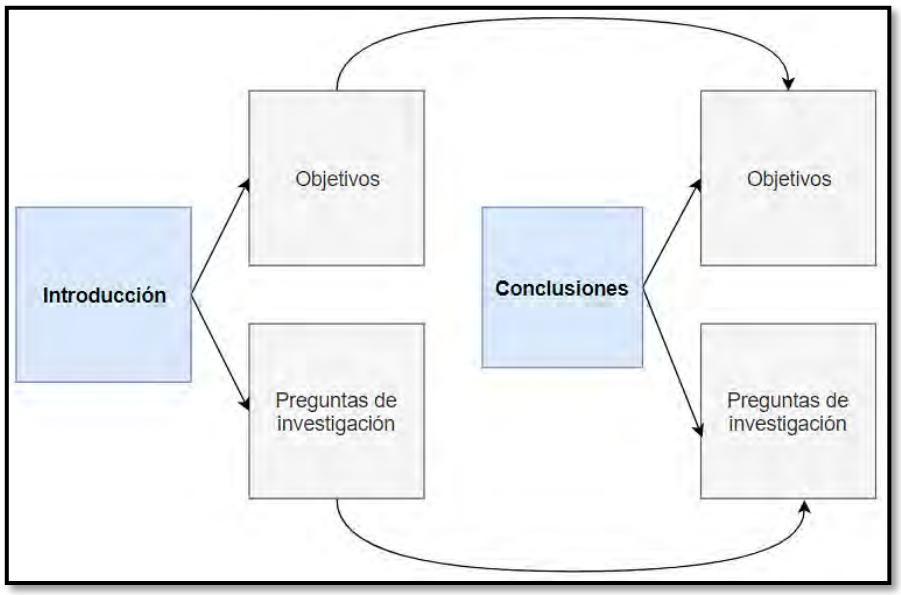


Figura 1-9 Flujograma conclusiones de tesis doctoral – Método Científico Circular y heurístico
Fuente: *Autor*

Capítulo 2

2 Fundamentos de las Redes Definidas por Software

Hoy en día se considera a SDN un tópico tecnológico de moda, ya que prácticamente todas las implementaciones de redes de estos últimos años abarcan alguna variante del concepto *Software-Defined*.

Lo más interesante de esta evolución en las redes es que si bien ya tiene algunos años tratando de darse un lugar en las infraestructuras y en muchos casos, logrando posicionarse de una manera muy adecuada en el mercado del *networking* empresarial, no existe en sí una definición única para SDN pues aún depende del fabricante que lo despliegue, sin embargo, en todas las definiciones está la búsqueda de mejorar patrones de rendimiento, seguridad, enfocado a la nube y visibilidad usando el *Internet-Fabric* como medio de transporte, reinventando de esa manera las redes de datos y comunicaciones.

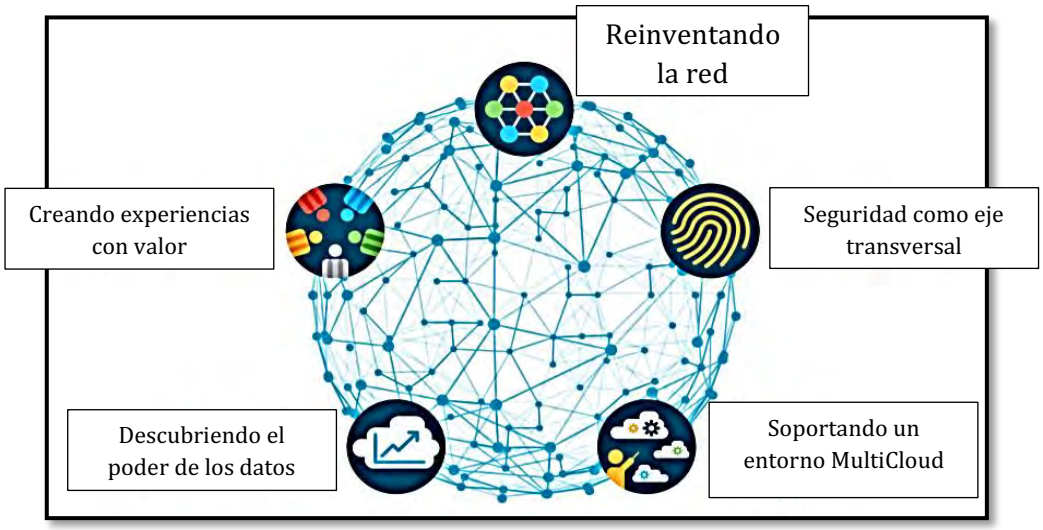


Figura 2-1 Reinventado la red con SDN, AI y ML (Machine Learning).
Recuperado de *(Cisco Systems, 2018)*

2.1 Definición de SDN: Conceptualización, beneficios y posibles impactos en la Industria

Los negocios necesitan de una renovación en sus infraestructuras de comunicaciones con el fin de:

- Tener redes inteligentes y con ello tomar mejores decisiones que beneficien al negocio.
- Mejorar la postura de seguridad y brindar visibilidad y control total de la red. La Seguridad es una parte fundamental de toda estrategia de digitalización. A medida que la conectividad extrema avanza, la exposición a riesgos y amenazas aumenta también. En base a un estudio de (Oracle, 2017), el costo promedio global de la violación a la seguridad de los datos llegó a $USD3.62 Millones de dólares en el 2017 y en el 2020

llegó a $USD3.86 Millones de dólares según IBM (IBM, 2020), lo que implica un crecimiento sostenido.

Figura 2-2 Impacto económico en la Seguridad de los Datos Recuperado de *(Cisco Systems, 2018)*

- Enriquecer la experiencia del usuario de las redes, dándoles comunicaciones transparentes, bajo demanda y con niveles de QoS claros.
- Transformar procesos y modelos de negocio, aumentando con ello la productividad y efectividad, por ejemplo, al permitir que aplicaciones soliciten dinámicamente recursos y servicios a la red.
- Contratar, retener y empoderar talento humano mejorando el comprometimiento, desarrollo en innovación y creatividad (*DevOps* y *NetDevOps*).
- Otorgar a las infraestructuras de red un enfoque holístico y clave para el crecimiento empresarial, reduciendo OPEX y simplificando su despliegue e implementación.

La evolución de las redes y del cómputo, tal como lo vimos antes, ha progresado rápidamente en los últimos años. Según (Forbes, 2017), el año 2017 fue el inicio de un ascenso meteórico de la tecnología, tanto en campo de SDN, AI y ML, jugando un rol primordial en la transformación digital, donde entornos abiertos, *open-source* y *open-hardware*, así como el crecimiento y avances en el uso del *Cloud* han transformado el entorno de producción, almacenaje y procesamiento de datos, abaratando costos, mejorando el *performance* y dando lugar a *BigData*.

En el pasado, sistemas inteligentes requerían mucho poder computacional que en el momento era extremadamente costoso conseguirlo, pero con el avance de la ciencia y tecnología, evolución de la Informática, mejora en los CPUs, GPUs y *Cloud/Fog Computing* han reducido el precio y dificultad de contar con ese poderío informático requerido (*Figura 2-3*).

Entornos colaborativos tanto del sector público como privado, universidades, compañías y gobiernos están fomentando estos cambios revolucionarios en las comunicaciones y campos de las Tecnologías de la Información.

Debido a la ubicuidad de las redes de comunicaciones, (Ransbotham, Kiron, & Gerbert, 2017) indican que el 84% de los ejecutivos aceptan estos cambios y deben prepararse para tomar una ventaja competitiva.

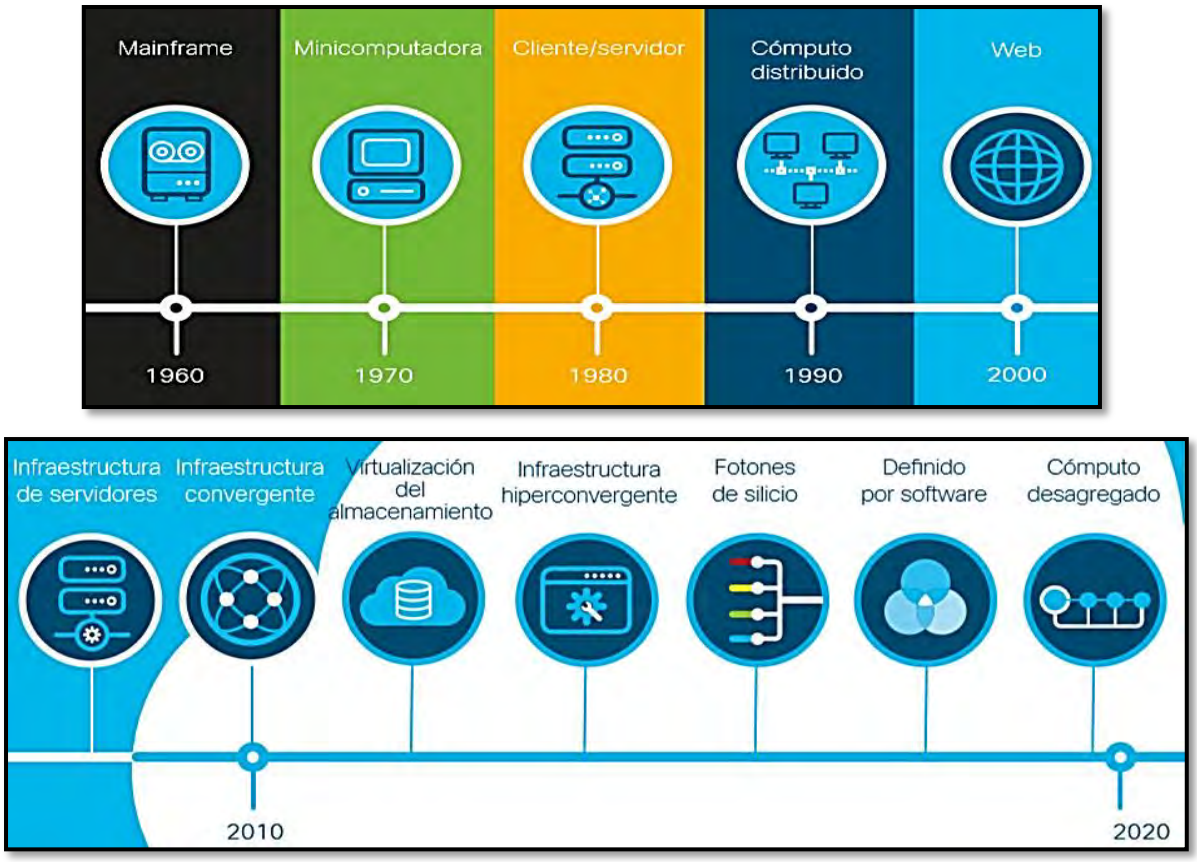


Figura 2-3 Evolución de la Infraestructura de Cómputo
Recuperado de *(Cisco Systems, 2019)*

Contar con una red que sea ágil, programable, rentable, segura e inteligente es un requerimiento y entender a SDN, siendo este cambio tecnológico el puntal de esta migración, toma una relevancia muy grande. Ya desde inicios del 2017, se realizaron predicciones indicando el impacto económico en las industrias que no realicen transformaciones digitales en sus negocios. Para el 2020, los negocios e industrias que sean innovadoras tomarán $USD1.2 Trillones de dólares de aquellas que no lo sean (McCormick, 2016). Con la pandemia esa cifra sería aún mayor.

Figura 2-4 Impacto económico en la falta de Innovación
Recuperado de *(Cisco Systems, 2018)*

SDN se encuentra en una etapa de estandarización, proceso importante para su desarrollo y así cubrir los requerimientos en la era de la digitalización.

Para Cisco Systems, SDN se implementa mediante el modelo de *Intent-Based Networking*, donde la escalabilidad, automatización a través de programabilidad, visibilidad centralizada y optimización de las redes tanto de infraestructura o *Enterprise*, DC y *Cloud* público/privado son elementos infaltables (Cisco Systems, 2020).

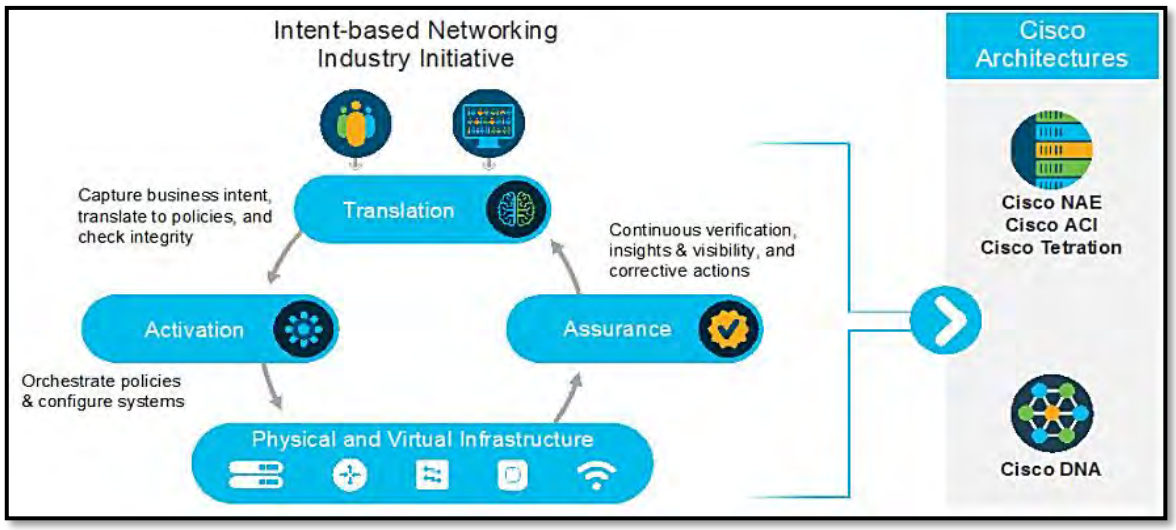


Figura 2-5 Intent-Based Networking
Recuperado de *(Apostolopoulos, 2018)*

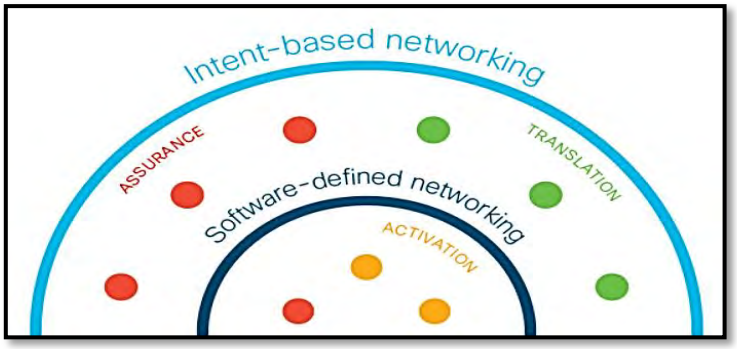


Figura 2-6 Intent-Based Networking soportada por SDN
Recuperado de *(Apostolopoulos, 2018)*

Otros *vendors* muy relevantes en el desarrollo de SDN como VMWare, se han preparado durante los últimos años para adoptar esta nueva ola de innovación mediante virtualizaciones especiales de la red o NVs (*Network Virtualizations*) ofreciendo maneras de implementar SDN de forma flexible y sin utilizar demasiados recursos, tanto computacionales como económicos. VMWare adquirió Nicira en 2012, uno de los mayores impulsores de la creación de *vSwitches*, desarrollando VMWare NV en forma de contenedores denominándolo VMWare NSX, compañía dedicada a SD-DC (VMWare SDX-Central, 2018).

Por su parte, Juniper (Juniper Networks, 2019), Citrix (Citrix, 2019), Huawei (Huawei, 2019), Cisco Systems (Cisco Meraki SD-WAN, 2019), entre otros fabricantes adoptaron también el concepto de *Underlay-Overlay Networks*, motivando así el despliegue de SD-WAN, concepto que será analizado más adelante.

Debido a esta diversidad de conceptos, la ONF (*Open Networking Foundation*[5]), consorcio sin fines de lucro creado para delinear el camino de los nuevo modelos de negocio e infraestructura, trabajando colaborativamente como una comunidad abierta para consolidar proyectos de desagregación de redes, *White-Box Networks*, así como para establecer estándares que revolucionarán la industria del *networking*, definió a SDN como la *separación física del plano de control del plano de datos de un equipo*, con el fin de centralizadamente controlar, monitorear y programar varios dispositivos a la vez a través de peticiones/respuestas desde una API.

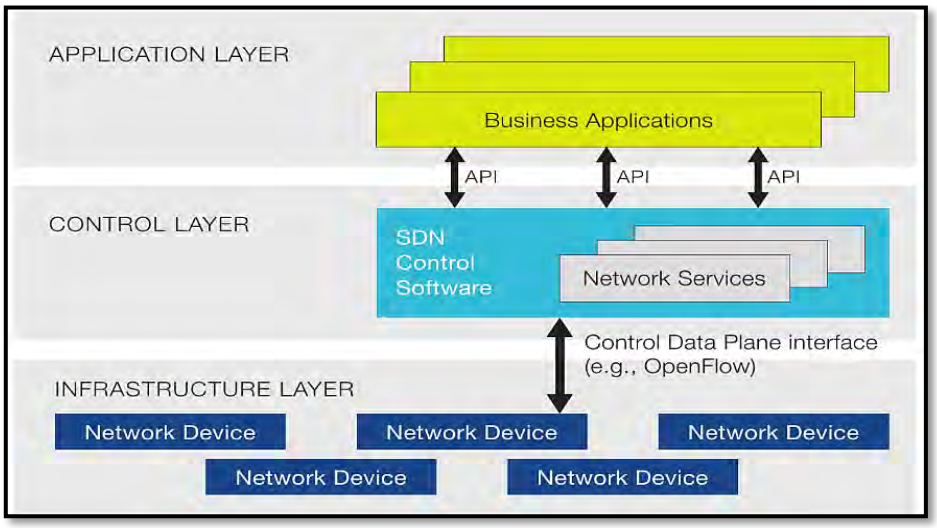


Figura 2-7 SDN según la ONF
Recuperado de *(SDX Central, 2019)*

SDN también se relaciona con el uso de equipos abiertos u *Open Hardware*. Un proyecto abierto originalmente lanzado por el equipo de *Open* vSwitch (OVS) en Nicira (ahora parte de VMWare) nombrado *Open Virtual Network* (OVN) se encarga de desarrollar estándares y protocolos agnósticos en esta área, usando APIs transparentes e interoperables.

Empresas como Facebook han desarrollado sus propios equipos y *hardware* (*Figura 2-8*), basando la inteligencia de dichos equipos en la programabilidad abierta y centralizada que entrega el administrador de la red. Facebook denominó esta solución como *Hybrid SDN*.

Figura 2-8 Open-Hardware de Facebook – 6-Pack /Backpack device
Recuperado de *(Bachar, 2015)*

[5] ONF: https://www.opennetworking.org/

Por lo general, equipos de red abiertos o *White-Boxes* corren NOS (*Network Operating Systems*) basados en Linux, un ejemplo sería Cumulus Linux[6], dando una mejor adaptación a entornos virtualizados en *DataCenters* que tengan VXLAN, *Segment Routing*, etc. Una de las ventajas de esta aproximación, es que, al ser protocolos abiertos, no dependen de un fabricante para implementar una infraestructura (Solución *Open Source* u *Open Networking*).

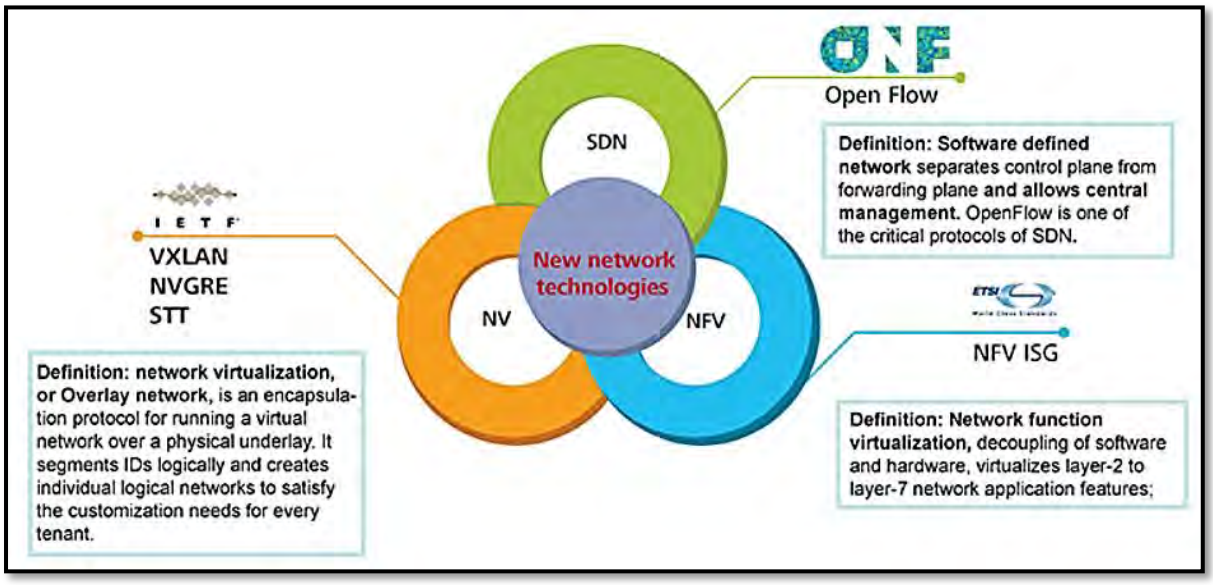


Figura 2-9 SDN: Aproximaciones y Conceptos
Recuperado de *(H3C, 2019)*

A pesar de que existan muchos conceptos y aproximaciones de SDN, esta tecnología es una realidad, es así como empresas alrededor del mundo están adoptando SDN y sus variaciones.

Las redes definidas por *Software* seguirán evolucionando durante los próximos años, por tal motivo los profesionales en IT deben entender a SDN, desarrollar mejoras y nuevas aplicaciones.

Según (Citrix, 2019), se estima que el mercado de SDN (en especial en el ámbito de infraestructura, Acceso y DC) crezca en más de $USD12 billones de dólares para el 2022, además, acorde con el análisis de (Dell'Oro Group, 2020), ya para el 2019 el *SDN Market* creció en un 64% comparado con el 2018, siendo Cisco Systems, Silver Peak, Versa, VMWare y Fortinet los cinco mejores SDN-SDWAN *vendors* en el mundo.

2.2 Arquitectura y Componentes básicos de SDN

Con el fin de comprender la arquitectura y componentes básicos de SDN, en primer lugar, se deben definir los planos estructurales de un equipo de red.

Bajo el contexto de las Tecnologías de la Información, un plano estructural de un equipo es el área de operación usada para diferenciar los distintos flujos de tráfico y el uso que se les da.

[6] Cumulus Linux: https://cumulusnetworks.com/products/cumulus-linux/

Las arquitecturas de red suelen diferenciar el flujo de datos de usuario de aquellos flujos que sirven para tomar la decisión de reenvío (mensajes de los protocolos de enrutamiento) y aquellos usados para monitorearlos o administrarlos remotamente.

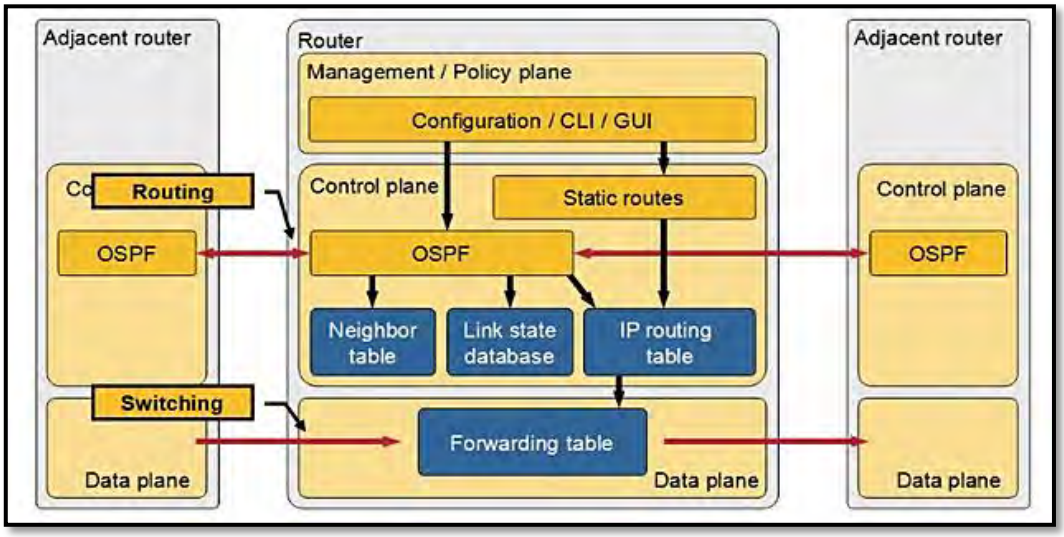


Figura 2-10 Planos estructurales de un equipo de red
Recuperado de *(Pepelnjak, 2013)*

Plano de Datos

También conocido como plano de reenvío o plano de usuario (*Forwarding Plane*), es el plano de un equipo de red implementado en *hardware* donde el flujo de datos llega y sale del dispositivo en dirección del destino a través de una interfaz de entrada y salida respectivamente.

El plano de datos se considera el "músculo operativo" del dispositivo de red.

Plano de Control

A diferencia del plano de reenvío, el plano de control se implementa en *software* (en el CPU del equipo), por lo tanto, no es tan veloz para procesar los flujos de información como lo es el *hardware*.

La función principal de este plano estructural es la de recopilar información de control, por ejemplo, la proveniente de los protocolos de enrutamiento, con el fin de construir la denominada Tabla de Enrutamiento (RIB - *Routing Information Base*), tabla que indica al equipo cuál es la interfaz de salida que debe tomar el flujo de datos proveniente del plano de datos.

De igual manera, el plano de control, además de generar la tabla de enrutamiento, la cual consta de las direcciones de red de destino junto con la interfaz por la que el tráfico debe salir para alcanzarla, la lógica de procesamiento de este plano permite definir ciertos paquetes que serán descartados, filtrados, marcados o priorizados.

En equipos convencionales y en *networking* tradicional, tanto el plano de control y el plano de datos, se encuentran en el mismo dispositivo, muchas veces implementados en el *firmware* de *routers* y *switches*.

El plano de control se considera el "cerebro" del dispositivo de red.

Plano de Administración

El plano de administración, parte del plano de control, es el encargado de permitir que el administrador pueda configurar los equipos, así como monitorearlos y visualizarlos tanto de forma local como remota. Este plano permite que se pueda ingresar a un equipo de red por sus distintas líneas de acceso (Consola/Auxiliar, Líneas VTY) o empleando protocolos especializados de monitoreo de red (SSH, SNMP).

En el caso de las líneas VTY o líneas de acceso virtual, esta forma de ingreso se denomina en banda (*In-Band*) y se da gracias al uso de protocolos como SSH; mientras los puertos de consola y auxiliar, requieren de un cable especial para configurar los equipos, por lo que el administrador debe estar cerca para configurarlo. Esta forma de acceso se denomina Fuera de Banda (*Out-Of-Band* u OOB).

El plano de administración permite tener una interfaz entendible entre los equipos de red y quien debe configurarlos o monitorearlos.

SDN: Una tendencia Moderna

La tendencia moderna es desacoplar el plano de control del plano de reenvío con el fin de centralizar la inteligencia, entregar programabilidad y acceso abierto a los protocolos, dando como resultado, una administración flexible y un envío de datos efectivo, este es uno de los conceptos que tiene SDN, pero llegar a ese punto no fue repentino, los diversos fabricantes de equipos de *networking* han ido adaptando sus sistemas operativos para de alguna manera lograr una separación de estos planos estructurales.

El concepto de la ONF para SDN nació de las redes *wireless*. En un inicio, los APs (*Access Points*) poseían toda su inteligencia de forma distribuida en cada uno de ellos, sin embargo, al creer el tamaño de la red y la necesidad de instalar más APs con el fin de generar mayor área de cobertura, su integración, monitoreo y administración era más complicada, por tal motivo, se diseñó un entorno con APs ligeros denominados LAPs (*LigthWeight Access Points*) pues no tenían un plano de control, encargándose solo de recibir y enviar datos y con un Controlador Wireless o WLC (*Wireless LAN Controller*), encargado de integrar, configurar y generar visibilidad total de la red inalámbrica, es decir, ser el "Cerebro de la Red Wireless". La comunicación entre los APs ligeros y el controlador *Wireless* se da gracias a CAPWAP (*Control and Provisioning of Wireless Access Points*), protocolo estandarizado en el RFC 5415 en el año 2009, el cual se basó en el protocolo LWAPP (*Light Weight Access Point Protocol*).

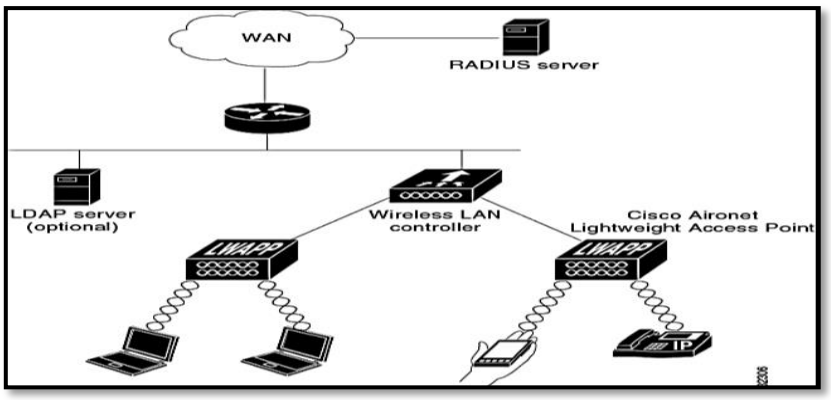


Figura 2-11 Esquema de Red Wireless con controladora – Split MAC Architecture
Recuperado de *(Cisco Systems, 2018)*

Muchos fabricantes de equipos de infraestructura tanto de *networking* de Campus como de DCs, dieron sus primeros pasos tratando de separar el plano de control del de datos modificando sus sistemas operativos monolíticos. Cisco Systems lo realizó con un tipo de conmutación de paquetes denominado CEF (*Cisco Express Forwarding*), duplicando su RIB (*Routing Information Base*) ubicada en el CPU, en una tabla llamada FIB (*Forwarding Information Base*) almacenada en un caché del plano de datos junto con una tabla de adyacencia similar a la tabla ARP, logrando así una conmutación más rápida al ejecutarse los procesos de decisión de envío a nivel de *hardware.* Este mecanismo fue replicado en otras versiones de Sistemas operativos, tanto de Cisco como de otras marcas.

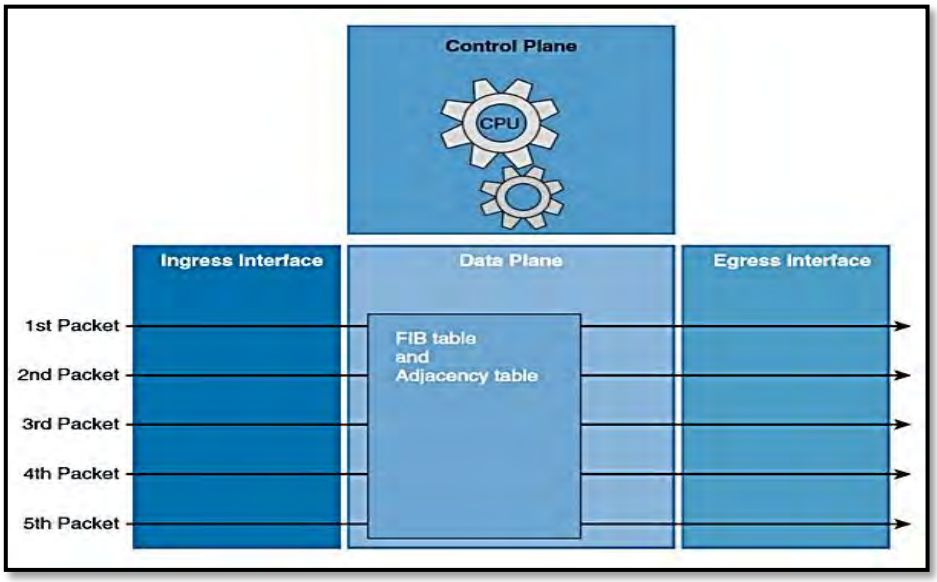


Figura 2-12 CEF – Cisco Express Forwarding
Recuperado de *(Teare, Vachon, & Graziani, 2015)*

Posteriormente surgió el nuevo paradigma de SDN, donde existe un plano de inteligencia y toma de decisiones centralizado en un Controlador, ante el cual, diversos equipos de plano de datos se relacionan, cumpliendo con ello todo lo que requiere una infraestructura de redes moderna.

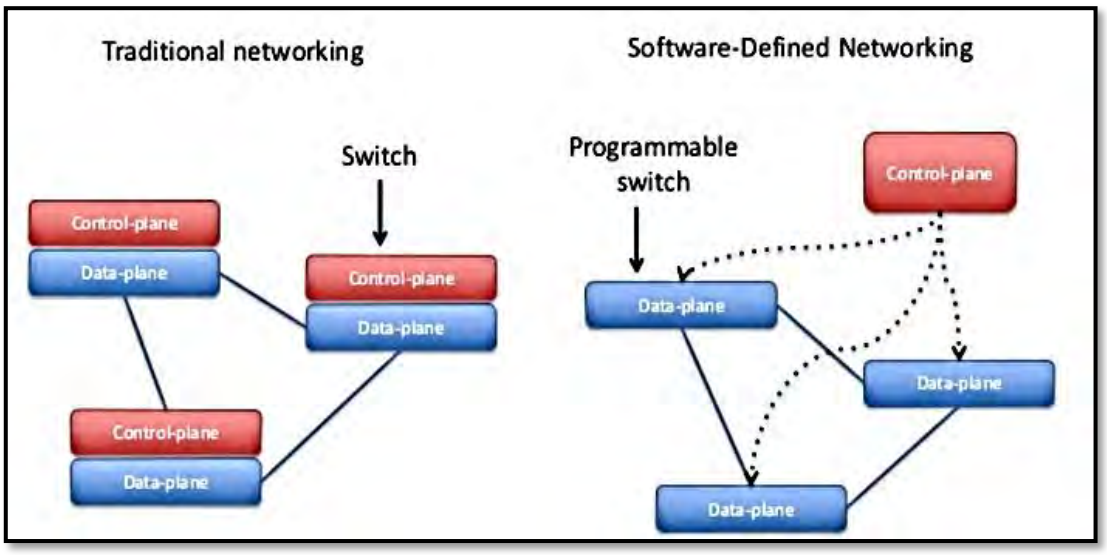


Figura 2-13 SDN: Nuevo paradigma de las redes
Recuperado de *(Capone & Cascone, 2015)*

Componentes de SDN

Los componentes básicos de una infraestructura de este tipo son:

- **Equipos de Red:** Son los dispositivos que conforman el plano de datos o plano de *forwarding* de la red. En este caso, estos dispositivos carecen de inteligencia y solo poseen rapidez de envío. Cabe mencionar que pueden ser equipos físicos o virtuales.
- **Controlador SDN:** Equipo encargado de dotar de inteligencia a la red. Es el lugar donde se toman las decisiones de envío, se establecen políticas de enrutamiento, conmutación, seguridad, así como políticas empresariales. Conforma el Plano de Control centralizado de la red, es decir, su cerebro. Es posible que el controlador se comunique hacia las aplicaciones, así como hacia los equipos de red usando APIs especializados.
- ***Northbound/Southbound Interfaces*:** APIs que permiten la comunicación desde el controlador hacia las Aplicaciones y el Plano de datos respectivamente. Emplean protocolos de comunicaciones estandarizados o propietarios.
- **NOS (*Network Operating System*)**: Sistema Operativo que emplean tanto los Controladores como los Equipos de Red. Pueden ser sistemas operativos abiertos o propietarios.
- **Servicios y Aplicaciones**: Constituyen la Capa de Aplicación del modelo ONF. Pueden ser aplicaciones que emplea el negocio (BSS - *Business Support System*) o aplicaciones que sirven para la operación de la red (OSS - *Operations Support System*) encargadas de dar visibilidad y comunicación con el administrador de la red.

El controlador (Plano de Control), es quien divide los dos tipos de protocolos/APIs que conforman una arquitectura SDN: *Northbound y Southbound.*

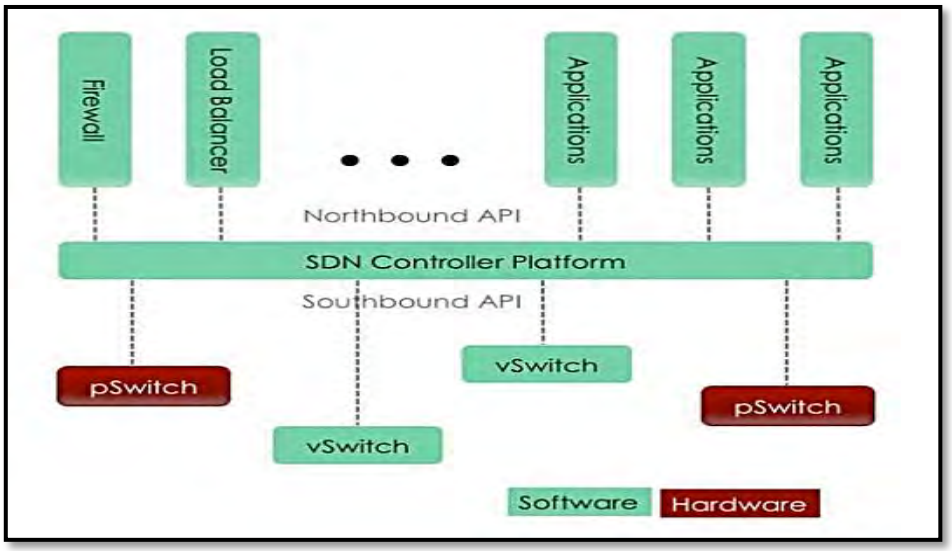


Figura 2-14 SDN: Esquematización de SDN: Northbound y Southbound APIs
Recuperado de *(Guis, 2012)*

Tabla 2-1 Interfaces SDN: Northbound y Southbound

Interfaces	Descripción
Northbound (API) Interface	Conjunto de protocolos que permiten la comunicación entre las aplicaciones del negocio y Apps para controlar los equipos de la red y el controlador. Esta API permite que la información de control de la red sea válida para instancias de abstracción mayores, tales como aplicaciones gráficas tipo *Front-ends* y dar *insights* a la red. Existen varios protocolos *Northbound* dependiendo de la infraestructura, por ejemplo, en entornos Cisco Systems se suele usar protocolos tipo REST (RESTCONF); en otros entornos con diferentes controladores como *OpenDayLight* (ODL) se usan además de RESTCONF, NETCONF o AMQP (*Advanced Message Queing Protocol*). El principal problema de estos protocolos es la falta de estandarización actualmente, pero entornos *Restful* son los más usados.
Southbound Interface	Conjunto de protocolos que permiten comunicar el controlador con los equipos que conforman el plano de datos, abstrayéndolos completamente de la Capa de Aplicación. Estos protocolos han tenido un mayor desarrollo en cuanto a la estandarización e interoperabilidad. Un gran ejemplo es *OpenFlow* (ONF), pero no es el único, existen muchos más como BGP-LS (*Border Gateway Protocol – Link State)*, NETCONF, PCEP (*Path Computation Element Protocol*), MQTT (*Message Queue Telemetry* Transport), CoAP (*Constrained Application Protocol*), entre otros.

Fuente: Elaboración Propia

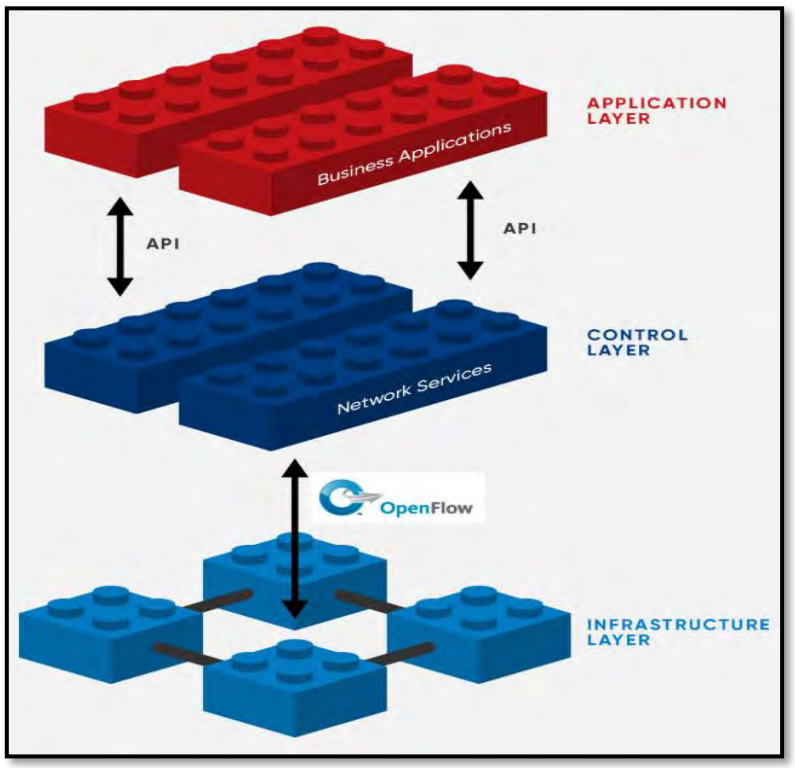


Figura 2-15 SDN: Capas de Aplicación, Control e Infraestructura
Recuperado de *(Open Networking Foundation - ONF, 2020)*

2.3 Automatización, Orquestación y Programabilidad de las Redes: DevOps y Redes Basadas en Contexto

SDN tiene el objetivo de crear infraestructuras mucho más ágiles, escalables, seguras y flexibles que las redes tradicionales. Uno de los mecanismos para llegar a ello es mediante la "Automatización" y así crear redes programables.

"Si el *software* va a comerse el mundo, entonces SDN se comerá al *networking"* fue dicho ya en el 2016 por (Rivenes, 2016). Las redes se han vuelto complejas y una forma de administrarlas adecuadamente es a través de SDN, es más, tal como lo hemos analizado, los usuarios demandan de cambios mucho más rápidos, velocidad que las redes tradicionales monolíticas no pueden satisfacer.

Según (Rivenes, 2016), los principios de SDN llevan las redes a lo que se acuñó como *Infrastructure-as-a-Code* (IaaC o IaC), lo cual permite que la automatización y orquestación sea una realidad, no obstante, IaaC también puede implementarse como una innovación en las redes tradicionales sin necesariamente llegar al punto de convertirla en SDN mediante mecanismos de automatización y programabilidad en redes y así exista una convivencia de redes modernas y de legado.

La Tabla 2-2, según (SDX Central, 2019), se establecen tres casos de uso donde se requiere y se necesita programación en SDN. La mayoría de los casos donde es posible aplicar la automatización y programabilidad es en la comunicación entre el controlador SDN con las aplicaciones y el plano de datos mediante APIs *Northbound y Southbound* respectivamente, sin embargo, sin importar donde se aplique, programabilidad adicional en la red llevará a un mejor uso del ancho de banda, mejora en el desempeño, así como a maximizar la eficiencia operacional y control de errores.

Tabla 2-2 Casos de Uso de Programabilidad SDN

Casos de Uso	Descripción/Características
Ajustar flujos de Red	Este caso de uso se enfoca en los protocolos, como *OpenFlow*, protocolo que será estudiado más adelante, con el fin de permitir la interacción de los controladores SDN con los *routers* y *switches* del plano de Datos. Para ajustar y manipular el tráfico en base a alguna política, se debe manipular los flujos en la red SDN, permitiendo así responder bajo demanda y automáticamente a los cambios.
Habilitar *DevOps* para automáticamente programar la red (*NetDevOps*)	El segundo caso de uso planteado va de la mano de la coordinación, automatización y filtraje en una red, con el fin de alinear de mejor manera las necesidades con las aplicaciones que corren en SDN. Con esta característica lo que se busca es extender las capacidades de automatización de los equipos del plano de datos de una forma escalable con el fin de soportar un mayor número de servicios, tal como se encuentra en estructuras de *Cloud Computing*, lo cual solo es posible con algún tipo de lenguaje utilizado en la creación de APIs y que se permita generar respuestas "*Cross-domain*". Claros ejemplos de un formato adecuado de datos son JSON (*Javascript Object Notation*), XMPP (*Extensible Messaging and Presence Protocol*) y YAML (*Yet Another Markup Language – Ain't Markup Language*). ***Ver Anexo A: Formatos y Estructuras de Datos para NetDevOps***

Programabilidad para generar automatización SDN	El último caso de uso muestra la intención de tener redes basadas en contexto, es decir, llevar la automatización al extremo, donde la red emplea AI y ML para tomar decisiones por sí misma.

Fuente: (SDX Central, 2019)

Tomando como referencia a la Tabla 2-2, SDN y *DevOps* no son para nada excluyentes el uno del otro, en realidad, *DevOps* compagina perfectamente en estos entornos.

En el primer caso de uso, se plantea SDN en su forma más simple, es decir, controlar *Hardware* con *Software* abierto, dando lugar a una red flexible y escalable, por ejemplo, mediante *OpenFlow* (u otro protocolo que permita controlar el plano de datos). Este hecho implica que las redes podrán alinearse sin problema con DevOps en una forma de IaaC, pero ¿Cuál es finalmente la definición de *DevOps*?

DevOps se refiere a la orquestación de procesos complejos, interdependientes y asociados con el desarrollo de *software* y las Operaciones en Tecnologías de la Información con el fin de acelerar los procesos de producción y desarrollo de servicios. *DevOps* se trató algún tiempo atrás y nació desde el lado del manejo puro de IT. La *Figura 2-16* muestra la evolución de *DevOps*, comenzando desde el primer código pensado como un desarrollador usando Fortran en 1957.

De acuerdo con (Gaurav, 2017), luego del crecimiento exponencial del Internet, en el 2003, Google contrató a Ben Treynor para liderar un proyecto donde se debía mantener estable y funcional una infraestructura por al menos el 99.97%, trabajando de la manera más transparente entre el equipo de desarrollo (*developers*) y el de operaciones (*IT Operations*) para lograr una satisfacción total del cliente. Con esa experiencia, John Allspaw y Paul Hammond, en la Conferencia *O'Reilly Velocity Conference*[7] del 2009, acuñan el nombre *DevOps* para referirse al equipo multidisciplinario, pero totalmente integrado que trabaja en una infraestructura automatizada.

A partir de ese momento, *DevOps* fue tendencia en la forma de implementar infraestructuras en IT, convirtiéndose en un movimiento cultural global que abarca temas de desarrollo, operación, automatización, seguridad, agilidad, pruebas de concepto y pruebas de desempeño, así como de cambio de estructura organizacional empresarial.

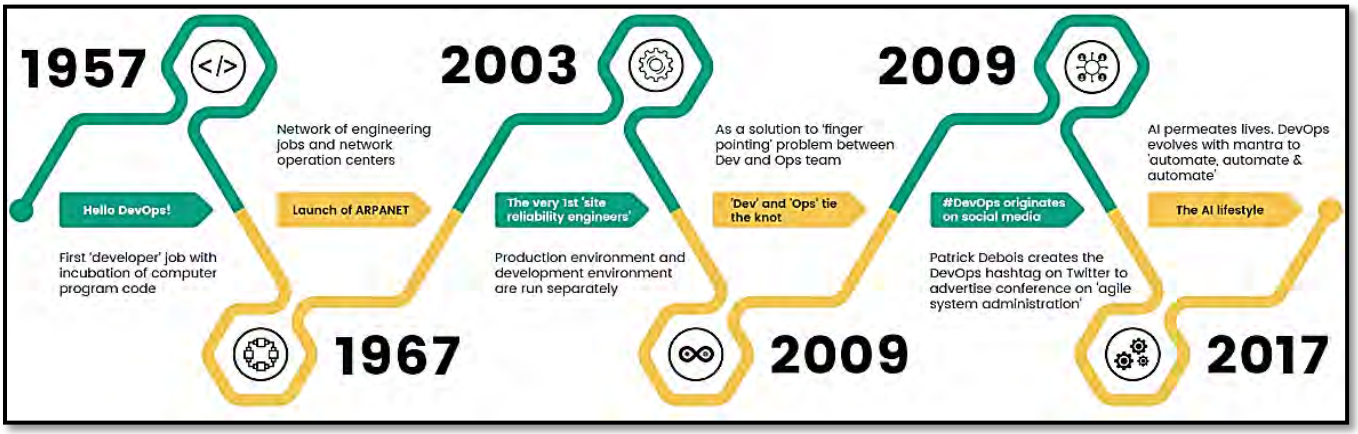


Figura 2-16 Evolución de DevOps
Recuperado de *(Gaurav, 2017)*

[7] O'Reilly Velocity Conference: https://conferences.oreilly.com/velocity/velocity2009/public/schedule/speaker/24342

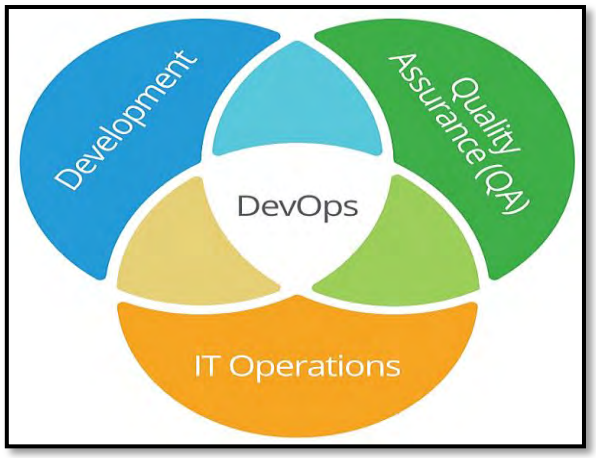


Figura 2-17 Definición de DevOps
Recuperado de *(MeritStep, 2019)*

El segundo caso de uso de la Tabla 2-2 está más relacionado al concepto de *NetDevOps* como tal, el cual se definirá a continuación y se establecerán sus diferencias y similitudes con la automatización y programabilidad de la red.

SDN, IaaC y *DevOps* promueven la creación y uso de un conjunto de herramientas para lograr una mayor y mejor adopción de la automatización en las redes.

2.3.1 El poder de la Automatización de Redes y NetDevOps

La automatización de las redes ha sido un tópico requerido y anhelado por mucho tiempo, pero debido a su difícil implementación en redes tradicionales, no ha impactado sino en los últimos años. Junto con SDN y un ambiente tecnológico adecuado, de forma natural todo conducirá hacia una automatización en todos los niveles.

Iniciativas tipo IaaC empezarán con los términos "*Dev*" y *"Ops"*, pero, a pesar de que los sistemas operativos estén automatizados y las aplicaciones automáticamente implementadas, la infraestructura de red, en muchos entornos, quizá la mayoría al momento de escribir la presente tesis, son estáticamente definidos y no automatizados, únicamente basados en modelos L2-L3 tradicionales con soluciones propietarias, dando problemas de interoperabilidad, escalabilidad y dificultad de monitoreo.

Partiendo de una investigación de la Revista *Network World* (Bednarz, 2016), se realizó una encuesta a 315 expertos en redes de empresas de tamaño mediano a grande, concluyendo que al rededor del 44% de los cambios que se generan en una infraestructura llevan a algún tipo de suspensión de servicios. La forma más habitual de encontrar y solucionar la red (mecanismo de *Tshoot*), según el 79% de los encuestados es usando CLI (*Command Line Interface*) e inspeccionar la configuración manualmente, quizá usar algún tipo de herramienta de monitoreo SNMP y finalmente *ping* o realizar una traza (*traceroute*). El último dato obtenido de esta investigación fue que el tiempo promedio para dar respuesta a un problema, según el 60% de la población encuestada es de 1 a 5 horas. La *Figura 2-18* resume esos datos estadísticos.

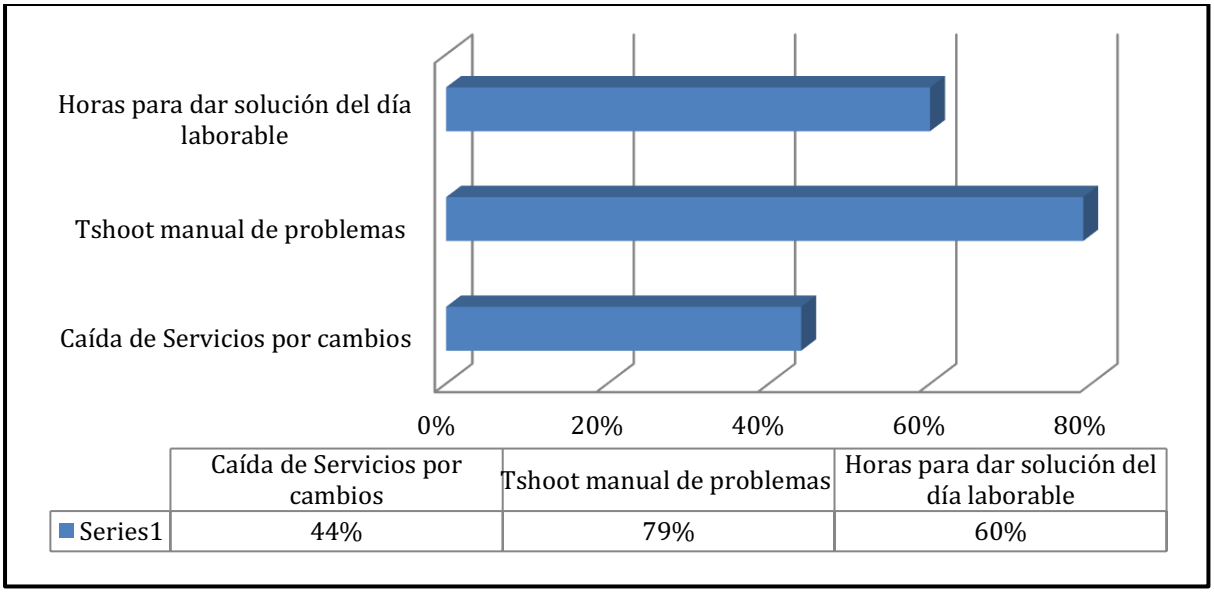


Figura 2-18 Datos Estadísticos – Cambios y Tshoot de red realizados de forma manual
Basado en *(Bednarz, 2016)*

Por todos estos motivos, es extremadamente necesario la programabilidad y automatización en la red.

Cabe mencionar que Programabilidad en la Red no es sinónimo de *NetDevOps*, ya que conocer *Python* y *Ansible*, así como representación de formatos de datos como YAML y *templates* tipo Jinja[8] son únicamente parte, fundamental eso sí, del conjunto que se denomina *NetDevOps.*

DevOps ha permitido el desarrollo de APIs que gestionan de mejor manera la automatización de IT, formando un *pipeline,* estableciendo además fases que constituyen el *development and operations*. Dichas fases, junto con las aplicaciones más importantes en el mundo del *DevOps* se aprecian en la *Figura 2-19*.

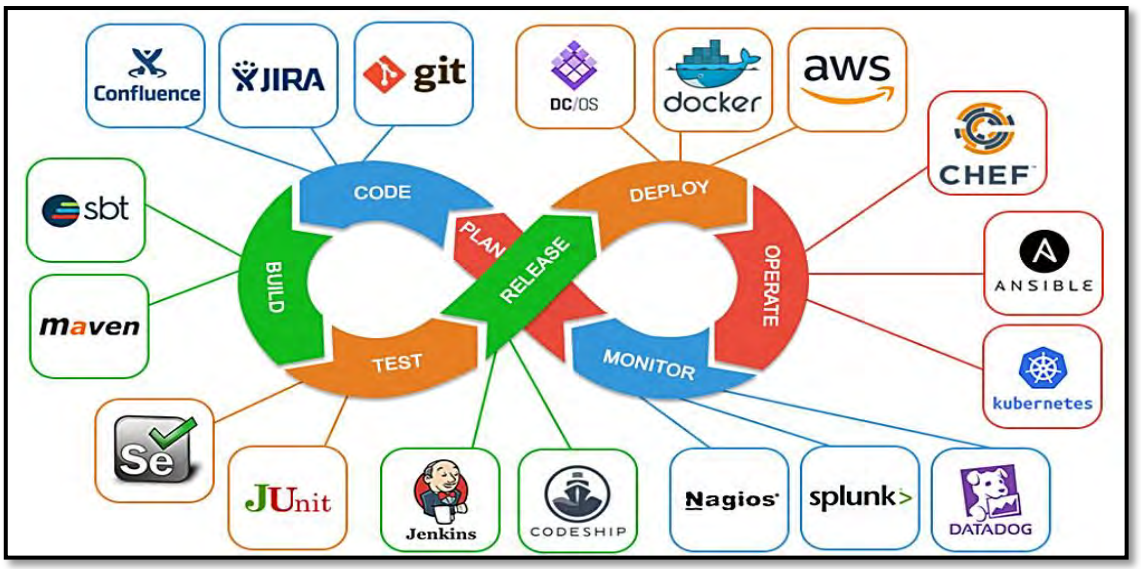


Figura 2-19 Ecosistema, Pipeline y ToolChain de DevOps
Recuperado de *(Erdem, 2019)*

[8] Jinja Template: http://jinja.pocoo.org/

NetDevOps es traer *DevOps* al *networking*; es así como un ingeniero en *DevOps* asegura la continua entrega del servicio IT, cambiando con ello el aspecto cultural-empresarial, buenas prácticas y herramientas que mejoran la efectividad. Bajo este modelo, tanto el equipo de desarrollo como el de operaciones no están separados, sino se unen en un solo equipo de trabajo, involucrándose en todas las etapas del ciclo de vida de una aplicación e infraestructura, por ello, las habilidades de los profesionales en este campo son variadas, algunas funcionando como eje transversal y llevando a la red a un concepto totalmente holístico.

En el modelo *DevOps*, existen tres prácticas habituales en el desarrollo, implementación y puesta en marcha de un *software,* una de las cuales se analizó en los párrafos anteriores:

- Integración Continua o CI por sus siglas en inglés de *Continuous Integration:* Práctica en la cual los desarrolladores guardan sus códigos en un repositorio central y colaborativo. En ese repositorio se pueden corren pruebas automáticas y gestión de versiones. Ejemplos muy importantes están en Git, Netbox, etc.
- Entrega Continua o CD por sus siglas en inglés de *Continuous Delivery:* Práctica donde los cambios en código son automáticamente enviados a un ambiente de pruebas después de la etapa de construcción (*Build Stage*).
  El término CI-CD de igual manera se emplea para establecer este *pipeline*.
- IaaC o *Infrastructure-as-Code*: Práctica donde la infraestructura es aprovisionada y administrada usando código en oposición a usar CLI, empleando técnicas como CI y CD.

Al tomar en cuenta en este *pipeline* a la infraestructura de red, nace el término *NetDevOps*, el cual no es más que la combinación de programabilidad en la red y más profundo, *Configuration as* Code e IaaC, con la compilación, prueba, entrega e implementación automática de extremo a extremo en la infraestructura empleando APIs adecuadas para el contexto de las redes de datos.

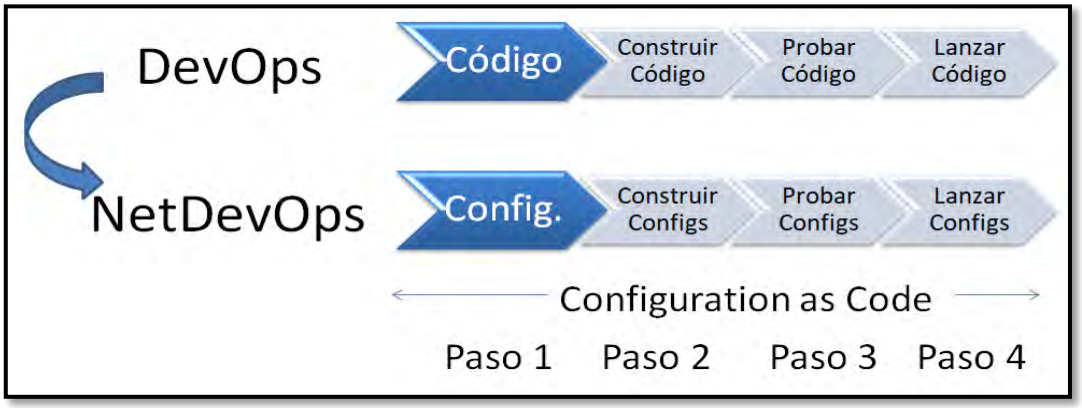


Figura 2-20 Cambios en una Red según NetDevOps Pipeline
Basado en *(Afaq, 2018)*

El *NetDevOps pipeline* puede dividirse en 4 pasos de ejecución (*Fig. 2-20*):

- Paso 1: Consiste en crear, verificar, ejecutar y enviar los cambios de configuración a un entorno de pruebas de concepto antes de implementarlo directamente, validando así su comportamiento. Este entorno se lo denomina *devBranch*. Herramientas basadas en Git como *GitHub* y *Ansible* para control de versiones o VCS (*Version Control System*), además de entornos de emulación de redes como Vagrant[9], Cisco CML - *Cisco Modeling Labs*[10],

[9] Vagrant: https://www.vagrantup.com/

EVE-ng[11], GNS3-VM[12], Mininet[13], entre otros, son muy útiles en este paso, considerado como un *white-box testing* o unidades de prueba de concepto remoto (*GitPush)*.

- Paso 2: Durante el paso 1, tan pronto como se desarrolló el *devBranch* y *GitPush* en los entornos emulados, se establece una estrategia de prueba en la infraestructura virtualizada, ejecutando todos los cambios requeridos a través de APIs de automatización. En este paso se generan los denominados *testing playbooks*, normalmente establecidos por los departamentos empresariales de QA (*Quality Assurance*).
- Paso 3: En este paso, una vez la prueba de concepto ha sido verificada y aceptada, se genera el *masterBranch* de ser posible, es decir, la etapa de la infraestructura demo o pruebas piloto.
- Paso 4: Una vez todos los códigos de automatización, programabilidad y monitoreo fueron probados y validados tanto en ambientes virtualizados como en pilotos, es momento de enviar las configuraciones al ambiente de producción, normalmente a través de un Ansible *Playbook* o cualquier API similar.

Los principios de operación detrás de *NetDevOps* se basan en medir el tiempo que le toma a la red en converger si se da un cambio o fallo, así como en el tiempo de implementar nuevamente otra infraestructura modificando los *scripts* definidos y validados en los pasos 1 y 2.

Estos pasos van de la mano con modelos y *frameworks* que han permitido que diversos programas se diseñen e implementen adecuadamente. Entre estos se puede mencionar al definido en la ISO 12207[14] o *Software Development Lifecycle* (SDLC) y sus modelos más importantes como *Waterfall,* Lean, Agile y *prototyping.*

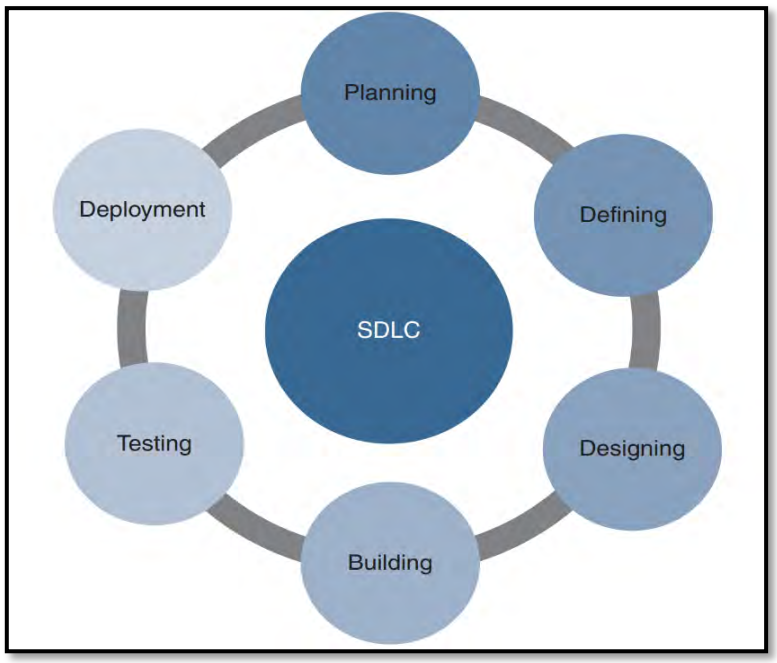


Figura 2-21 SDLC – Software Development Lifecycle
Recuperado de *(Jackson, Gooley, Iliesiu, & Malegaonkar, 2020)*

[10] Cisco Modeling Labs: https://www.cisco.com/c/en/us/products/cloud-systems-management/modeling-labs/index.html
[11] EVE-ng (Emulated Virtual Environment): https://www.eve-ng.net/
[12] GNS3-VM: https://www.gns3.com/
[13] Mininet: http://mininet.org/
[14] ISO/IEC 12207: ISO - ISO/IEC/IEEE 12207:2017 - Systems and software engineering — Software life cycle processes

Con el fin de comprender de mejor manera el mundo de la automatización y cómo es posible contar con un entorno programable en la red, el ***Anexo A: Formatos y Estructuras de Datos para NetDevOps***, da un resumen sobre los formatos de datos utilizados en en este ecosistema y sus características.

XML	JSON	YAML

<Servers>
<Server>
<name>Server1</name>
<owner>John</owner>
<created>123456</created>
<status>active</status>
</Server>
</Servers>
	{
	Servers: [
	{
	name: Server1,
	owner: John,
	created: 123456,
	status: active
	}
	]
	}
		Servers:
		- name: Server1
		owner: John
		created: 123456
		status: active

Figura 2-22 XML vs. JSON vs. YAML
Recuperado de *(IBM Developer , 2019)*

2.3.2 *Application Programming Interfaces* - APIs

Los formatos de Datos son utilizados por distintos dispositivos inteligentes para transferir información relevante y realizar alguna actividad automatizada. La interfaz que permite ese intercambio se denomina API por las siglas en inglés de Interfaz de Aplicación Programable, es así que una definición rápida de API sería un software que permite a otras aplicaciones acceder a sus datos e interactuar con ellos.

Una API permite definir un conjunto de reglas que describen una aplicación y así interactuar con otra. Estas APIs, en el mundo hiperconectado actual, las encontramos por doquier, por ejemplo, AWS (*Amazon Web Services*), Facebook, Google, servicios de aerolíneas para buscar vuelos, búsquedas en servicios de comida a domicilio o distintos entornos IoT utilizan algún tipo de API para interactuar con su entorno. La pandemia de COVID-19, dio un impulso adicional al empleo de APIs, pues también existen para obtener estadísticas en tiempo real de contagios, así como de información referente a ese tema.

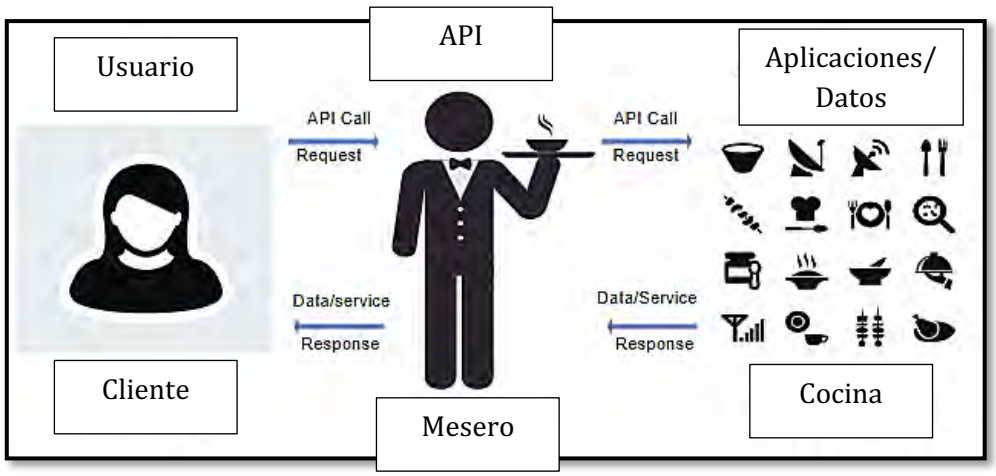


Figura 2-23 Esquema de un API
Recuperado de *(Cisco Netacad, 2020)*

En base a una explicación excelente sobre APIs dada por (Cisco Netacad, 2020), en la *Figura 2-23 Esquema de un API*, podemos apreciar el funcionamiento de una Interfaz de Aplicación

Programable, el cual sería como un intermediario entre el usuario y el pedido (*request*) de información o interacción con los datos. En dicha figura se hace un símil con un pedido de un cliente de comida en un restaurante a través de un mesero, mesero que funcionaría como API entre el comensal y la comida que está siendo preparada en la cocina.

En esta era de la programabilidad, es necesario visualizar el entorno, el cual, bajo mi criterio está conformado por:

- Formato de Datos que se pueden intercambiar en una infraestructura de red estructuradamente (XML, YAML, JSON)
- Modelos de Datos que permiten estandarizar este tipo de intercambio programático, describiendo al conjunto de datos. El más importante en la actualidad, YANG.
- Mecanismos de transporte seguro entre todos los actores que intercambian datos en el *networking* moderno. Muchas APIs suelen usar SSH, TLS o HTTPS para el intercambio de información, datos y órdenes.
- APIs que sirven como intermediario entre la administración, gestión y el *hardware/software* subyacente, por ejemplo, RESTful – RESTCONF y tipo NETCONF.

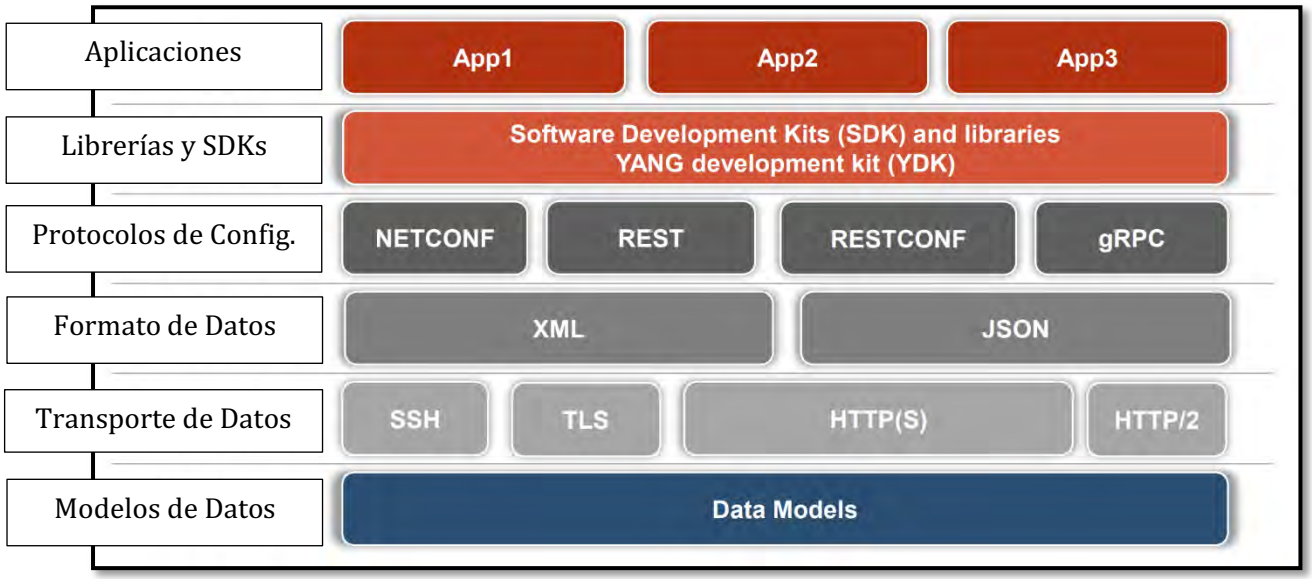


Figura 2-24 Stack para Programabilidad en las Redes
Recuperado de *(Roman & Bryan, 2018)*

Tipos de APIs y su Contexto
Una API puede clasificarse en tres categorías:

- API de Servicio: En este tipo de API, una aplicación puede llamar a otra para desarrollar alguna acción, normalmente de forma independiente entre ellas. Ejemplo de esta API sería el pago mediante tarjeta de crédito en una página Web de un comercio.
- API Informacional: API que permite que una aplicación solicite información a otra aplicación, por ejemplo, para funciones de telemetría e inventario de equipos en la infraestructura.
- API de *Hardware*: APIs que permiten obtener acceso a dispositivos tipo sensores. Son muy usado en entornos IoT.

Al momento de diseñar o querer utilizar una determinada API, es necesario tomar en cuenta el tipo de acceso también, así como flexibilidad y propiedad de uso, es así como las APIs pueden considerarse de:

- Acceso Público: También denominadas *Open-APIs,* son aquellas que están disponibles para el uso del público en general sin restricciones, de ahí que, muchos proveedores de servicios y de contenidos como *Google Maps* por ejemplo, requieren que los usuarios empleen un *token* abierto para monitorear el uso de las peticiones realizadas. En la cuenta de GitHub *Public APIs*[15], está una lista ordenada en categorías de APIs abiertas.
- Acceso Interno: También denominadas *Internal-APIs*, son aquellas usadas por una determinada organización con el fin de obtener datos para uso interno. Ejemplos típicos están en la recolección de información estadística de ventas en una empresa.
- *Partner API*: APIs creadas con el fin de mantener información entre una empresa y sus colaboradores y así facilitar el giro del negocio. Este tipo de APIs tuvo un crecimiento grande en época de pandemia de COVID-19 para el envío de comida a domicilio desde una central general de acceso.

Si bien existen diversos tipos de APIs, las APIs que están basados en estándares de servicios Web han tomado relevancia en *networking* principalmente por su ubicuidad y facilidad de aprendizaje.

Un servicio Web es el servicio que emplea la WWW (*World Wide Web*) sobre el Internet para su funcionamiento. Existen cuatro tipos de APIs en este contexto:

- SOAP – *Simple Object Access Protocol* (basado en XML)
- REST – *Representational State Transfer* (basado en mecanismos HTTP)
- XML-RPC – *eXtensible Markup Language – Remote Procedure Call*
- JSON-RPC – *JavaScript Object Notation – Remote Procedure Call*

Tabla 2-3 APIs tipo *Web Services*

	SOAP	**REST**	**XML-RPC**	**JSON-RPC**
Formato de Datos	XML	JSON, XML, YAML, etc.	XML	JSON
Primer Lanzamiento	1998	2000	1998	2005
Característica principal	Estable	Flexible y muy usado	Simple y muy usado	Simple

Fuente: (Cisco Netacad, 2020)

En vista de que SOAP, si bien emplea una estructura tipo XML para intercambiar información sobre HTTP o SMTP desde que fue creado por Microsoft en 1998, es considerado lento y complejo. De ahí nace la idea de desarrollar una API que no requiera del uso de XML, naciendo en ese momento un REST API, el cual usa HTTP, volviéndose una opción popular gracias a su

[15] Public-APIs: https://github.com/public-apis/public-apis

estructura. Según (Cisco Netacad, 2020), más del 80% de todas las APIs usan un variante tipo REST.

RESTful API

REST es una API diseñada para arquitecturas de tipo servicios web, pero eso no implica que es únicamente utilizado con ese fin, en realidad, hace referencia al estilo de arquitectura web cliente-servidor, el cual posee algunas características y comportamiento en particular que puede ser ampliamente utilizado en *NetDevOps*.

En general, una REST API corre sobre el protocolo HTTP, definiendo un conjunto de peticiones-respuestas comunes llamadas métodos, dos conocidos son GET y POST.

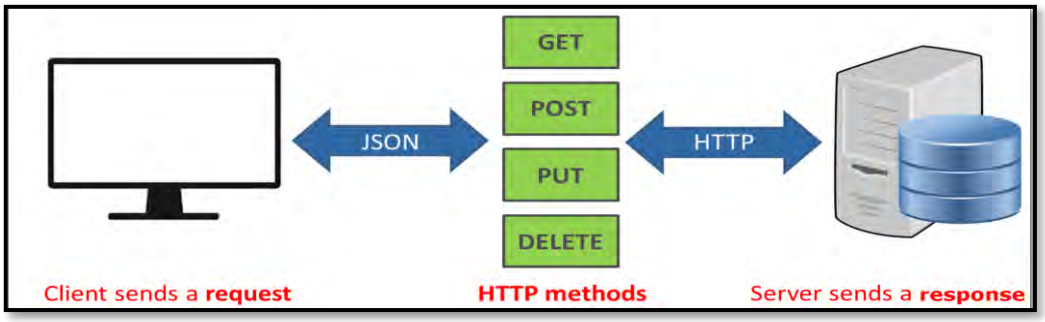


Figura 2-25 Esquema de funcionamiento REST API
Imagen bajo licencia Creative Commons *(Esta foto bajo CC BY-SA-NC)*

Para que una API sea considerada RESTful, debe tener las siguientes características:

- Desarrollado bajo un entorno cliente-servidor, donde el cliente maneja el *front-end*, mientras el servidor maneja el *back-end* de la aplicación.
- Debe contar con un entorno de tipo *stateless*, es decir, sin memoria de datos del lado del servidor al momento de realizar peticiones. El estado de la sesión se almacena en el cliente.
- Posibilidad de almacenaje de respuesta en *caché*.

Con el fin de implementar un RESTful API, es necesario tomar en consideración que, al implementarse mediante HTTP, se tienen cuatro (4) elementos básicos:

- URI (Identificador Uniforme de Recursos).
- Formato de datos admitido por la API (JSON, YAML, XML o cualquiera que sea un estándar de hipertexto válido).
- Métodos válidos para HTTP
- Control de API mediante hipertexto.

Los métodos u operaciones más comunes son *POST, GET, PUT, PATCH* y *DELETE*, mismas que son las más comunes para HTTP también (ver *Tabla 2-4*):

Tabla 2-4 Métodos de RESTful API

Método HTTP/RESTful	Descripción de la Operación
Post	Crear un dato/información
Get	Obtener/Leer un dato

Put/Patch	Actualizar un dato / Parcialmente almacenar un dato
Delete	Eliminar un dato
Head	Obtener el *header* de una petición *Get,* por lo general para comparar modificaciones
Trace	Solicitud para generar un *trace* de las acciones llevadas a cabo por el server

Fuente: (Jackson, Gooley, Iliesiu, & Malegaonkar, 2020)

Para identificar un recurso o servicio Web como las RESTful APIs se emplea un URI (*Uniform Resource Identifier*), el cual es una cadena de caracteres que se basa en dos especializaciones:

- URN (Nombre Uniforme de Recursos): Identifica la nomenclatura del recurso.
- URL (Localizador Uniforme de Recursos): Define la ubicación de red del recurso.

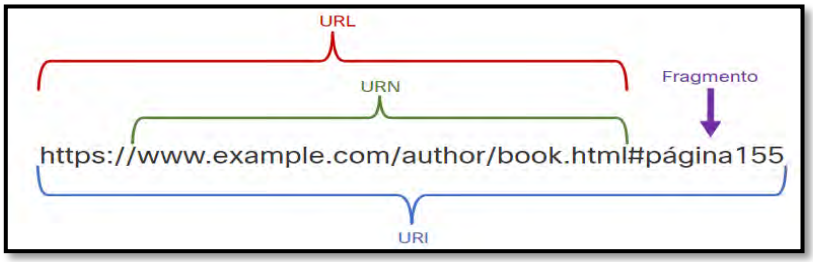


Figura 2-26 Partes de una URI
Recuperado de *(Cisco Netacad, 2020)*

La estructura de una petición REST, tiene tanto la identificación del Servidor API mediante una estructura URI, donde consta tanto la URL (URN + Protocolo), así como la localización de los recursos y la consulta o razón de la petición junto con el formato de datos.

En la porción de la consulta de la petición REST, se encuentra tanto el formato de datos a usar (JSON, XML, YAML, entre otros), la clave o patrones de seguridad, autorización y seguimiento y los parámetros propiamente dichos de la consulta, como el caso de la *Figura 2-27 Estructura petición REST,* donde se realiza una petición preguntando sobre los datos de contacto (Primer nombre, apellido, Identificación y número celular) de los usuarios registrados cuyo nombre sea Jean

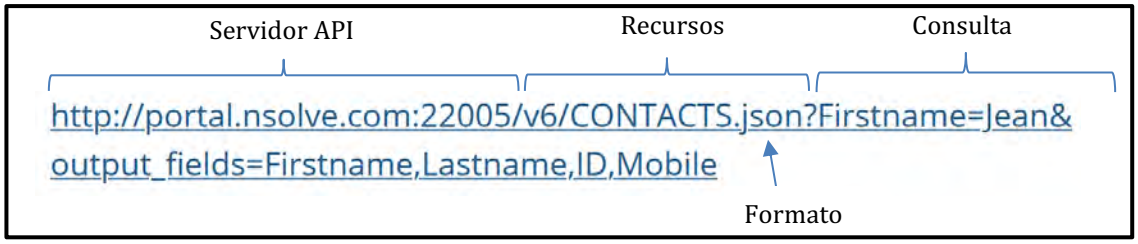


Figura 2-27 Estructura petición REST
Basado de *(nSolve - Excellence in software design, 2019)*

La respuesta de la petición REST anterior es la siguiente:


```
{
  "CONTACTS": {
    "Contact": [
      {
        "Firstname": "Jean",
        "Lastname": "Barnet",
        "Mobile": "07703 392745",
        "ID": 1763
      },
      {
        "Firstname": "Jean",
        "Lastname": "Parsons",
        "Mobile": "07703 355730",
        "ID": 1971
      },
      {
        "Firstname": "Jean",
        "Lastname": "Jones",
        "Mobile": "09843502893",
        "ID": 44057
      }
```


Figura 2-28 Respuesta parcial en JSON a una petición REST
Recuperado de *(nSolve - Excellence in software design, 2019)*

Podemos apreciar, según la figura anterior, que se obtuvo los datos solicitados.

Una aplicación muy útil para probar RESTful APIs es *Postman*[16], pues tiene todo lo necesario para construir y enviar peticiones. De la misma manera que al usar ese *software*, es posible interactuar con una REST API mediante *scripts* de Python, a través de cURL y con páginas Web para desarrolladores.

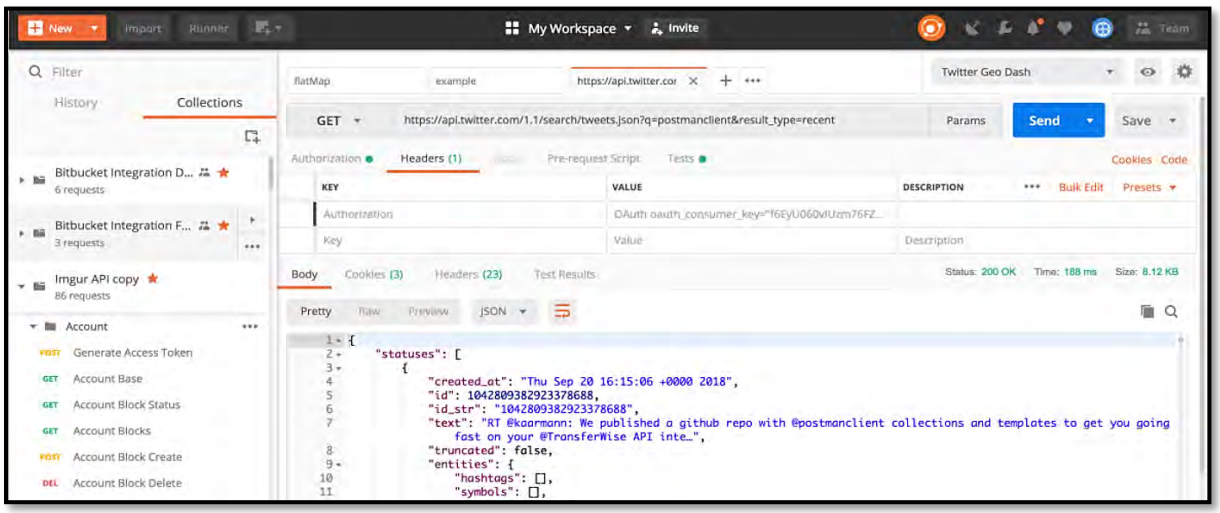


Figura 2-29 Petición RESTful mediante Postman a una API de Twitter
Recuperado de *(Logan, 2018)*

Debido a que el proceso RESTful maneja una estructura cliente-servidor, las peticiones HTTP generan códigos o estatus de respuesta para reconocer si la comunicación fue exitosa o no:

- **Código 1xx (Informacional):** La petición fue exitosamente recibida por el equipo de red/servidor, por lo que el proceso continúa.

[16] Postman: https://www.postman.com/

- **Código 2xx (Exitoso):** La petición fue exitosamente recibida, entendida, aceptada y servida por el equipo de red/servidor.
- **Código 3xx (Redirección):** Se requiere de acciones adicionales para completar la petición.
- **Código 4xx (Error del Cliente/Usuario):** La petición no fue entendida, es una petición no autorizada o es una petición a un recurso no encontrado en el equipo de red/*server*.
- **Código 5xx (Error del Equipo de Red/Servidor):** El equipo de red/servidor falló en el intento de procesar la petición.

Success (2xx)	Description
200	Request Succeeded
201	The request has been fulfilled; new resource created
204	The server fulfilled request but does not return a body

Server Error (5xx)	Description
500	Internal Server Error
501	Not implemented

Client Error (4xx)	Description
400	Bad Request. Malformed Syntax
401	Unauthorized
403	Server understood request, but refuses to fulfill it.
404	Resource not found given URI

Figura 2-30 Códigos de Estatus HTTP comunes en RESTFul APIs
Recuperado de *(Roman & Bryan, 2018)*

Partiendo de ese conocimiento, es posible emplear mecanismos de solicitud-respuesta para orquestar, aprovisionar, configurar y monitorear de forma automática y programática a equipos y sistemas operativos de red (NOS).

Protocolos como NETCONF (*NET CONFiguration*) y RESTCONF, permiten utilizar estas arquitecturas de intercambio de mensajes y dar mecanismos alternativos a la administración de una infraestructura de red moderna. En ese punto, el uso de un CLI (*Command Line Interface*) con protocolos de administración tradicional como SNMP es opcional, pues se podrá emplear *scripts* en Python o usar cualquier herramienta que realice peticiones tipo RESTful.

NETCONF fue creado en el año 2006 en el RFC 4741, a partir del cual evolucionó hacia el RFC 6241[17] en el 2011 con el fin de sobrellevar las dificultades de administrar una infraestructura de red de forma efectiva. Este protocolo establece un canal de comunicación entre un mánager y un agente o nodo monitoreado a través de SSH en el puerto 830 (NETCONF *port*) con el fin de entregar características o capacidades (*capabilities*) soportados en modelos de datos tipo XML en un inicio y actualmente YANG – *Yet Another Next Generation* (RFC 7950[18]) en un formato cliente-servidor tipo RPC (*Remote Procedure Call*).

YANG es un tipo de lenguaje para modelado de datos en infraestructuras de red y datos intercambiados en una red. Si bien, inicialmente fue creado para NETCONF, hoy es utilizado tanto con RESTCONF como con cualquier protocolo que se adapte a este modelo para establecer

[17] NETCONF – RFC 6241: https://tools.ietf.org/html/rfc6241
[18] YANG Model – RFC 7950: https://tools.ietf.org/html/rfc7950

configuraciones estandarizadas y estados operacionales de los datos, proveyendo de esa manera sintaxis y semántica clara.

```
module: ietf-interfaces
   +--rw interfaces
   |  +--rw interface* [name]
   |     +--rw name                          string
   |     +--rw description?                  string
   |     +--rw type                          identityref
   |     +--rw enabled?                      boolean
   |     +--rw link-up-down-trap-enable?     enumeration {if-mib}?
   +--ro interfaces-state
      +--ro interface* [name]
         +--ro name               string
         +--ro type               identityref
         +--ro admin-status       enumeration {if-mib}?
         +--ro oper-status        enumeration
[…]
```

Figura 2-31 Modelo Simple IETF tipo YANG
***Recuperado de** (Roman & Bryan, 2018)*

YANG tiene una definición totalmente estándar (IETF, ITU y *OpenConfig*), así como en entornos *vendor-specific* y para extensiones de protocolos como en BGP para equipos IOS-XE (*Model Driven Programmability*).

Por otro lado, RESTCONF es un protocolo tipo HTTP definido en el RFC 8040[19], en el que, en lugar de usar XML para codificar datos, emplea JSON también, pero no hay que pensar que RESTCONF es el reemplazo de NETCONF, pues con RESTCONF se puede usar una API RESTful que genere peticiones y configurar equipos de red mediante *DataStores* o capacidades almacenadas en NETCONF.

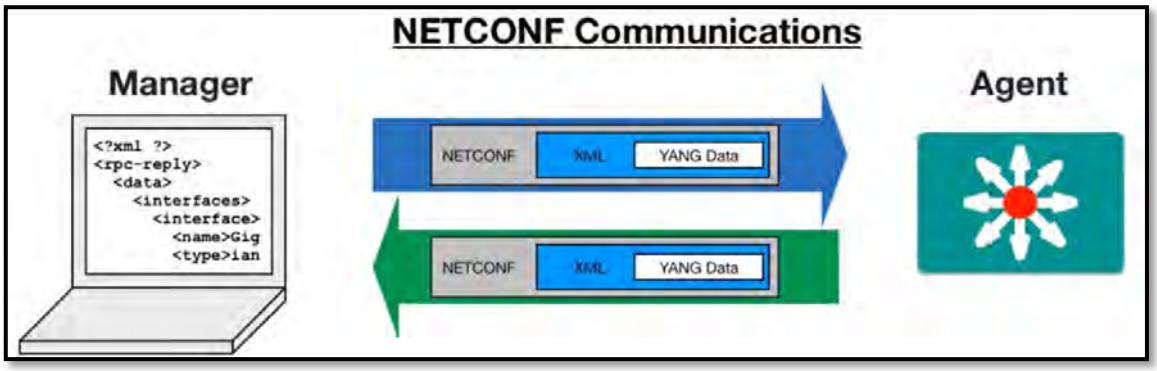


Figura 2-32 Esquema de funcionamiento de NETCONF
***Recuperado de** (Okasha, 2017)*

Tabla 2-5 Operaciones de NETCONF y Métodos de RESTCONF

Función	NETCONF	RESTCONF/RESTFul
Crear una fuente de datos	<edit-config> </edit-config>	POST
Obtener datos/metadatos	<get-config> </get-config>	GET

[19] RESTCONF – RFC 8040: https://tools.ietf.org/html/rfc8040

Crear o reemplazar fuente de datos	<edit-config> (nc:operation="replace")	PUT
Borrar una fuente de datos	<edit-config> (nc:operation="delete")	DELETE

Basado de: (IETF RFC 8040, 2017)

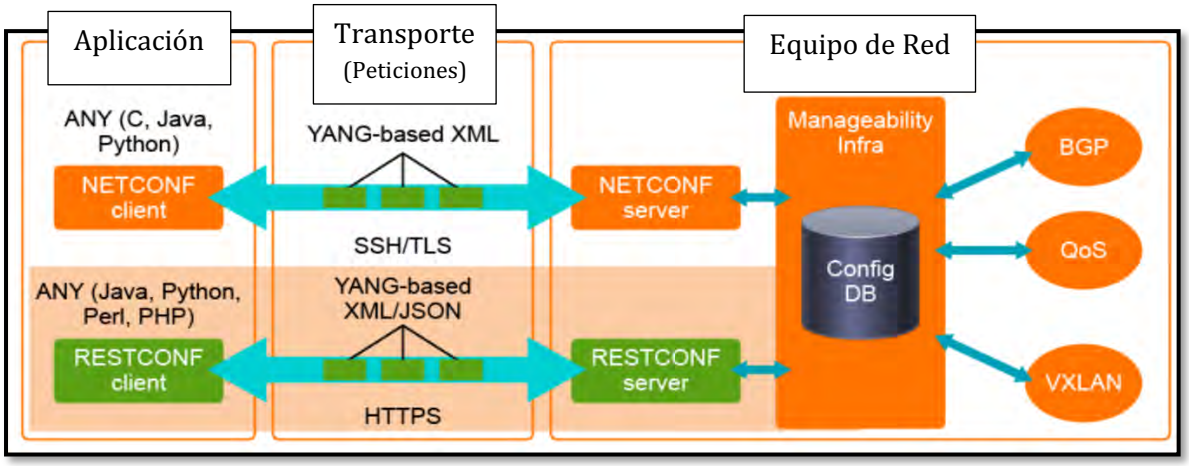


Figura 2-33 NETCONF vs RESTCONF
Recuperado de *(Roman & Bryan, 2018)*

Con el fin de tener una idea más clara de la utilidad y funcionamiento de NETCONF, en el ***Anexo C: NETCONF en la Práctica***, se plantea una Prueba de Concepto exitosa en un *Router* Cisco con IOS-XE (CSR1000v), tanto para telemetría básica (obtener información del equipo), como para configuración mediante programabilidad.

2.3.3 Herramientas para Automatización y Programabilidad de Redes: *Ansible, Puppet, Chef* y *Netmiko-NAPALM-Nornir* como Herramientas tipo *Python Library*

Como lo mencionado en capítulos anteriores, la automatización de las redes ha sido un tópico requerido y anhelado por mucho tiempo, pero debido a su difícil implementación en redes tradicionales, no ha impactado sino en esta última época. Junto con SDN y un ambiente tecnológico adecuado, de forma natural, todo conducirá hacia una automatización y programabilidad en todos los niveles.

Según (Cisco NetAcad DevNet Associate Course, 2020), automatización es usar código para configurar, implementar y administrar aplicaciones junto con la infraestructura tanto a nivel de redes, cómputo y almacenamiento, así como los servicios que corren.

Al momento de escribir la presente tesis, puedo decir que hoy más que nunca es una necesidad para toda infraestructura contar con herramientas que le permitan escalar sus operaciones más fácil y rápidamente, esto gracias a los nuevos requerimientos de usuarios del siglo XXI y de sistemas críticos modernos, requerimientos más necesarios durante y post pandemia COVID-19.

Dispositivos de red como *routers, switches, firewalls, Access point*s, tradicionalmente eran configurados de forma manual y monolítica mediante entornos tipo CLI (*Command Line Interfaces*) o en el mejor de los casos mediante un protocolo *in-band* como SSHv2 o SNMPv3

para mejorar el entorno de configuración a distancia, de todas maneras, todo proceso manual es lento y muy complicado en su adaptación a nuevas capacidades, así como son propensos a tener varios errores humanos que causen discontinuidad en la comunicación o en la entrega del servicio, falta de documentación actualizada o incompleta en tiempo real, difícil de probar y de difícil compaginación con las mejores prácticas modernas.

La automatización es una parte clave de entornos SDN y desde mi punto de vista, es una manera de interoperar entre las redes tradicionales y las redes definidas por *software* debido a que otorgan los siguientes beneficios:

- *Frameworks* para tener un tipo de Infraestructura "*Self-service*", es decir, una infraestructura bajo demanda, tanto a nivel de pruebas de concepto, como para implementaciones completas. Un ejemplo puede verse en el ***Anexo C: NETCONF en la Práctica*** y ***Anexo D: Ansible para entornos NetDevOps en infraestructuras de Red***
- Escalamiento bajo demanda a nivel de APIs, principalmente por la heterogeneidad del tráfico. Plataformas en *Clouds* son muy adaptables en este punto.
- Visibilidad de la infraestructura a gran nivel.
- Mitigación automática de problemas o errores al reconocerlos y en niveles de automatización avanzado, autocorregir el problema.

Tal como se vio en ***2.3.1 El poder de la Automatización de Redes y NetDevOps***, no se puede dejar de hablar del concepto *DevOps* en el contexto de automatizar redes. Es así como hace más de una década atrás, Patrick Debois, dio luces a un concepto denominado *Agile Infrastructure and Operations*, entregando lineamientos y un contexto claro para emplear métodos de Desarrollo en el área de Operación de infraestructura (Debois, 2008). La presentación influyó en cómo automatizar infraestructura física y virtual usando mecanismos de control de versiones, tal como lo que se hace con Git[20]. A partir de 2008, el concepto *DevOps* se volvió muy popular.

Entre los principios elementales de *DevOps* y por supuesto de *NetDevOps* están:

- Enfocado en la automatización de tareas.
- Generación de pruebas de concepto, control de versiones y pensar que el error es parte del camino al desarrollo correcto.
- Mejora constante para generar mejores beneficios empresariales y reducción de costos (Las Tecnologías de la Información son el soporte de todo negocio del siglo XXI), así como mejores tomas de decisiones.
- La implementación de infraestructuras de red va de la mano de concepto de *Lean-Agile*, es decir bajo principios de simplicidad, rapidez, efectividad y siempre tener código/*scripts* funcionales.
- Los diseños de red, hoy en día, siguen patrones tipo MVC[21] (*Model-View-Controller*) con el fin de adaptarse de mejor manera a los cambios tecnológicos mediante el concepto de SoC (*Separate-of-Concerns*) y modularidad, así como el *Observer Pattern*[22] empleado para el envío de cambios de configuración de forma masiva.

[20] Git: https://git-scm.com/

[21] MVC Pattern: https://dotnet.microsoft.com/apps/aspnet/mvc

[22] Observer Pattern: https://www.baeldung.com/java-observer-pattern

Las herramientas más importantes que se encuentran en el área de *NetDevOps* están lideradas por **Ansible, Puppet, Terraform, SaltStack** y **Chef** por nombrar unas cuantas, sin embargo, es posible desarrollar APIs nativas que interactúen con el *fabric* subyacente y permitan dar control, visibilidad y orquestación, muchas de ellas construidas a través de Python.

Existen varias herramientas básicas para automatizar redes, por lo que empezaré a nombrar las más elementales antes de Ansible, herramienta insigne de *NetDevOps*, en la siguiente Tabla.

Tabla 2-6 Bash y SDKs

Herramienta básica de Automatización	Descripción/Características
Bash	Bash (*Bourne Again* Shell) es el nombre del *Shell* de Unix, por lo que su uso es común en varias distros de Linux y en MacOS (actualmente se emplea el *Z-Shell*). Es un *Shell* que se podría encontrar en prácticamente cualquier aplicación o sistema operativo que corra entornos abiertos, por lo que es considerado muy ubicuo. Configurar mediante *Bash scripts* es básicamente igual a hacerlo en una CLI, ya que permite interpretar órdenes, por ello, es posible usarlo en entornos de automatización y programabilidad de infraestructuras
Diversos lenguajes de programación sofisticados con SDKs	Cuando las implementaciones se vuelven más complejas, en especial al tratarse de operar y construir entornos virtualizados, suele emplearse SDKs (*Software Development Kits*) por ejemplo AWS SDK para Python o AWS para Javascript en Node.js, con ello, es posible tener librerías y *built-in features* como JSON, YAML o interacciones tipo REST (APIs) para trabajar con *datasets* complejos, operaciones en paralelo, reconocimiento de errores, etc.

Fuente: (Cisco DevNet - Programming Fundamentals, 2020)

Idempotencia y Características de las Herramientas en *NetDevOps*

Cuando se tiene en mente realizar programabilidad en una red, es imprescindible contar con un procedimiento claro de programación como el usado en la programación imperativa, el cual no es más que aquel enfocado a tener una secuencia de comandos ordenados con el fin de conseguir un objetivo, generando controles de flujo, verificación de estructura, telemetría, etc.

Los *scripts* deben ser efectivos y reusables, por lo que el código tiene que:

- Estar estandarizado tanto en parámetros, banderas, errores, etc.
- Ser jerárquico en su estructura, dividiéndose en tareas de forma lógica y efectiva.
- Ser de alto nivel para toda la implementación y de bajo nivel para el desarrollo en fases.
- Lo más genérico y reusable posible, esto conlleva al concepto de idempotencia.

Uno de los objetivos primordiales de la automatización y programabilidad de redes es tener idempotencia en sus *scripts*, es decir, llegar a un estado deseado luego de la aplicación del código sin importar las condiciones iniciales en que se encuentre la infraestructura.

La idempotencia en *NetDevOps* se basa en los siguientes principios:

1. Asegurar que el cambio que se va a realizar en la infraestructura no se ha realizado anteriormente. Este principio en inglés se llama "*First, do no harm*", en el cual es mejor no realizar ningún cambio, a realizar algo que resulte en un error irrecuperable.

2. Principio de Inmutabilidad y control de cambios en etapas de pruebas de concepto. Este principio verifica todas las posibilidades de cambios no en un entorno de producción, sino en un entorno PoC (*Proof-Of-Concept*). De igual manera se plantea realizar entornos *Infrastructure-as-Code* bajo emulaciones con *software* adecuado.
3. Evitar efectos o daños colaterales, producto de la programabilidad en la infraestructura.
4. Todos los componentes del entorno de programabilidad deben ser idempotentes.

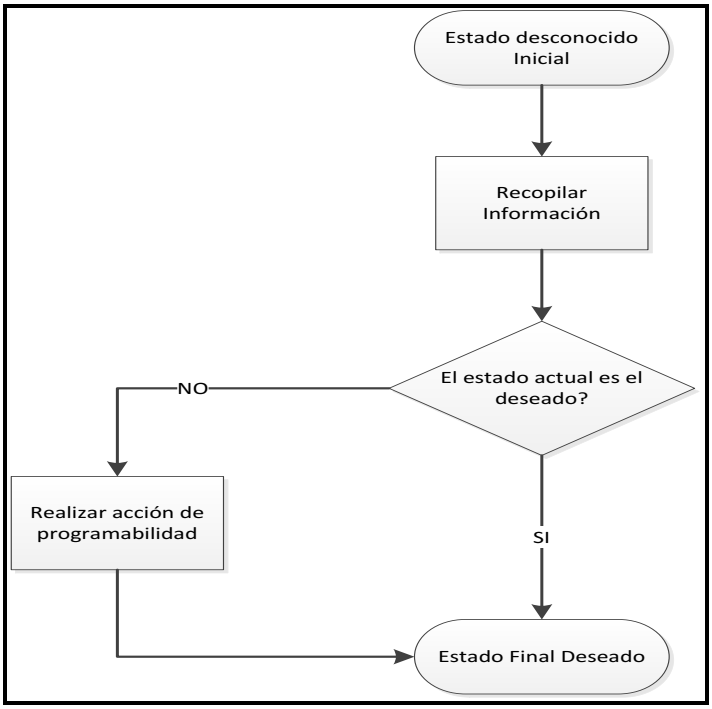


Figura 2-34 Flujograma de Idempotencia en programación de NetDevOps
Basado de *(Cisco NetAcad DevNet Associate Course, 2020)*

Herramientas de automatización y programabilidad como Ansible, Terraform, Puppet o Chef ofrecen capacidades muy poderosas en comparación con mecanismos de configuración y programación tradicionales como Bash o empleando nativamente algún lenguaje de programación como Python aún si se dispone de SDKs, pues estas herramientas reúnen funciones variadas en un entorno tipo API para simplificar y estandarizar su acceso, disminuyendo el hecho de incluir código extenso y tener *scripts* más simples y legibles, pero sin dejar a un lado la posibilidad de acceder al código puro para una revisión profunda.

Las herramientas antes mencionadas permiten además la inclusión de *plugins* totalmente configurables en *Python, Ruby, Go*, o cualquier otro lenguaje que sea aceptado.

Durante estos últimos años se han desarrollado módulos especializados para diferentes *vendors* o marcas de equipos de infraestructura de red y servidores, así como para desarrollar funciones especializadas que puedan correr de forma masiva/escalable como restauración y respaldo de configuraciones, generar copias de código o *snapshots*, etc., sin descuidar aspectos relacionados con seguridad y encriptación de archivos sensibles y críticos bajo entornos cliente-servidor a través de agentes distribuidos.

Una de las ventajas de contar con herramientas de automatización es la posibilidad de obtener datos e información de la infraestructura, comúnmente conocido como generar un inventario o telemetría en *networking*. Entre los datos que se pueden recopilar están las versiones de los

sistemas operativos, detalles de los archivos de configuración, tiempos de reinicio de los equipos, entre otros.

Además, en su mayoría, incluyendo Ansible, Puppet y Chef, son herramientas de tipo *Open Source*, cuyo desarrollo está apoyado en la comunidad de *software* abierto para mejorar y disminuir los *bugs*, así como ser distribuidos mediante *GitHub* o cualquier otro tipo de repositorio público como Ansible Galaxy.

La idea de tener *Infrastructure-as-Code* y SDN obliga a que los desarrollos en temas de programabilidad sean idempotentes, garantizando así que el objetivo general se consiga a través de etapas que sean claras, reversibles y adaptables. De igual manera, esas infraestructuras se adaptan mejor cuando las herramientas se diseñan para ser *stateless* en lugar de *stateful*.

Tanto Ansible, como Puppet, Chef o la mayoría (por no decir todas) las herramientas de *NetDevOps* deben mantener entornos sin-estado, pues en base a análisis y pruebas de concepto realizadas durante la presente tesis, es posible afirmar que las APIs que mantengan el estado (mantengan en el servidor el estado) son inconvenientes para una automatización de tipo *full-stack* (infraestructura y servicios/aplicaciones en su conjunto) principalmente por que el estado será destruido por un *rebuild* típico de la infraestructura, además de que estos entornos son difíciles de actualizar y de realizar migraciones.

En APIs de tipo sin-estado, se tienen aplicaciones que requieren interacciones atómicas o asíncronas entre un cliente y servidor, en donde cada petición tiene una respuesta independiente de otras peticiones anteriores. Un ejemplo de ello son las APIs tipo RESTful. De todas formas, para este tipo de APIs, existe una manera de mantener cierta información externamente tanto al cliente como al servidor. Estos entornos son las aplicaciones tipo HTTP, donde es necesario mantener una *cookie* por motivos de proyección comercial para la interacción, por ello, la *cookie* se almacena en una base de datos que actúa como un intermediario. En este último caso, si se necesita de autenticación basada en *cookie*, implicaría que la información se mantenga en el servidor, convirtiendo de esa manera en una aplicación *stateful.*

Para mayor claridad, la siguiente imagen explica esos tres escenarios:

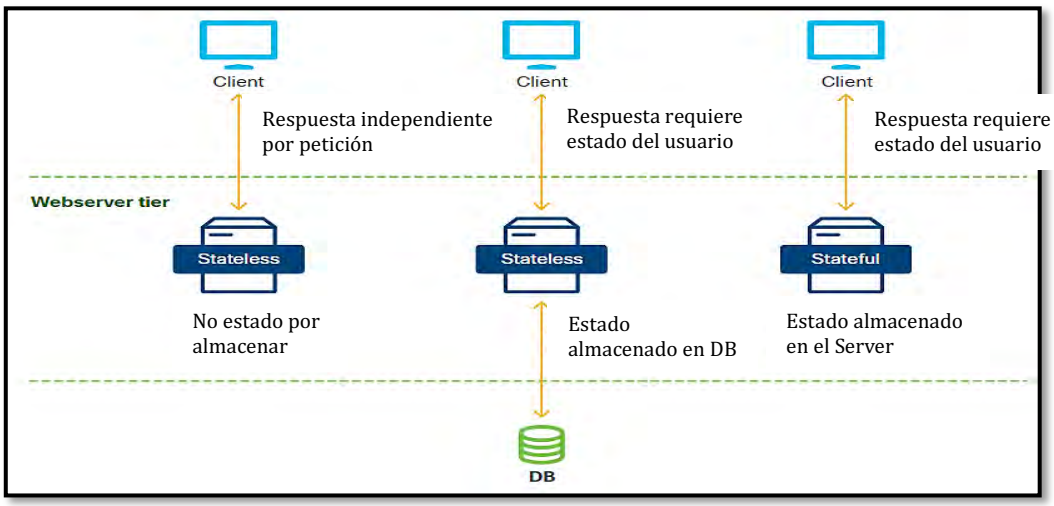


Figura 2-35 Esquema de APIs Stateless y Stateful
Basado en *(Cisco NetAcad DevNet Associate Course, 2020)*

Según (Cisco NetAcad DevNet Associate Course, 2020), la primera herramienta funcional dedicada a la automatización y programabilidad de redes fue Puppet, lanzada en el año 2005 como una herramienta totalmente abierta. Posteriormente se comercializó como Puppet Enterprise por Puppet Labs en el 2011.

Además de que Puppet, Ansible y Chef son las herramientas más utilizadas al momento de escribir la presente tesis, pues las tres poseen las siguientes características en común:

- Curva de Aprendizaje rápida.
- Tienen versiones abiertas (*open source*).
- Tienen la capacidad de administrar, monitorear y configurar equipos de infraestructura de diversas marcas aplicando *plugins* o módulos de fácil uso.

Ansible

Es una herramienta que simplifica muchas tareas en el campo de las Tecnologías de la Información, dando agilidad, aprovisionamiento y automatización en la configuración, implementación y orquestación de servicios.

Una de las premisas clave de Ansible es dar la posibilidad de modelar la infraestructura de TI describiendo la interrelación de todos sus sistemas en lugar de administrarlos un sistema por vez.

Genera un entorno sin agentes o *agentless*, por lo que es muy simple de desplegar usando además un formato de datos como YAML en sus *Ansible Playbooks* lo que lo hace ideal para entornos que desean mantener texto claro en su configuración.

Figura 2-36 Logo de Ansible
Recuperado de *(Huang, 2020)*

Módulos de Ansible

Ansible funciona al conectar distintos nodos (o equipos), enviado programas llamados Módulos de Ansible hacia ellos, siendo estos programas escritos de manera que sean recursos para llegar al estado deseado de la infraestructura.

Ansible ejecuta estos módulos, por lo general mediante transferencia tipo SSH y los remueve cuando termina la ejecución.

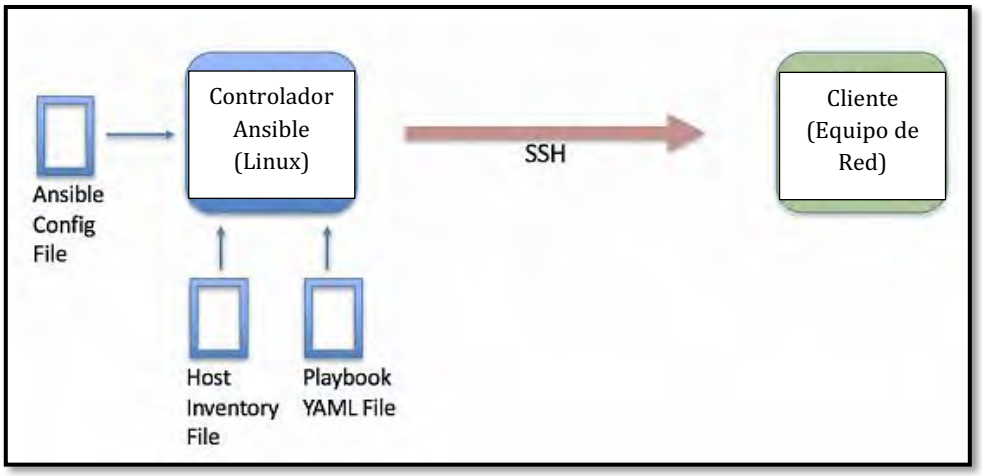


Figura 2-37 Estructura de Ansible
***Recuperado de** (Tamilselvan, Froehlich, & Raghunathan, 2019)*

La librería de módulos puede estar en cualquier máquina, sin la necesidad de un servidor dedicado, uso de demonios o base de datos estática.

Lo módulos para automatizar infraestructura de red multimarca se encuentra en la siguiente documentación de Ansible:

https://docs.ansible.com/ansible/latest/modules/list_of_network_modules.html

En cuanto a la seguridad y control de acceso, es posible usar *passwords* así como llaves SSH sin necesidad de otorgar acceso *root* (concepto de mínimo privilegio); pero se debe tener en cuenta que con el uso de contraseñas, Ansible las mostraría en texto plano, por lo que se sugiere usar llaves SSH o al menos Ansible Vault[23]

En caso de usar llaves SSH, se facilita la automatización de ejecución de *playbooks.*

Ansible *Configuration File*

Este es un archivo predeterminado que contiene parámetros para ser usados con el sistema de automatización de Ansible, sin embargo, puede ser editado y personalizado.

El *path* de su ubicación es **/etc/ansible** y por defecto tiene el nombre de **Ansible.cfg**.

Se emplea el comando **ansible --version** para encontrar su ubicación y versión.

La siguiente figura muestra un Ansible *Configuration File* de ejemplo, en la cual se identifica la ubicación del *Inventory File*, el uso de variables descubiertas en los equipos a automatizar (*gathering*), *timeout* para las conexiones fallidas de SSH (10 segundos), no uso de avisos o advertencias luego de correr Ansible y la no creación de un archivo tipo *retry* en caso de fallas.

[23] Ansible Vault: https://docs.ansible.com/ansible/latest/user_guide/playbooks_vault.html#playbooks-vault

```
[defaults]
inventory = /etc/ansible/hosts
gathering = explicit
host_key_checking = False
timeout = 10
deprecation_warnings = False
retry_files_enabled = False
```

Script 1 Ansible Configuration File
Recuperado de *(Tamilselvan, Froehlich, & Raghunathan, 2019)*

Según el *Script 1 Ansible Configuration File*, no existe verificación de llaves SSH (*host_key_checking = False*), lo que implicaría una debilidad en cuanto a control de acceso. Siempre es preferible considerar una adecuada conexión SSH antes de usar Ansible.

El siguiente *link* da una explicación de todas las opciones posibles en un Ansible *Configuration File*:

https://docs.ansible.com/ansible/2.4/intro_configuration.html

Ansible *Inventory File*

Este es un archivo que contiene información sobre los equipos a automatizar, por ejemplo, sus direcciones IP o variables de configuración para lograr una conexión por SSH (usuario y contraseña). Se debe recordar que es preferible usar llaves SSH, así como activación de *Python Interpreter* al usar *Python Virtual Environments* en entornos de Centros de Datos.

Es además posible agrupar usuarios, equipos de red, servidores que tengan características y configuraciones similares bajo un grupo en particular. Cada grupo se identifica por un nombre entre corchetes **[ ]**. Existe también un grupo general llamado ***ALL***.

```
[IOS]
R1 ansible_host=172.16.101.98 ansible_user=cisco ansible_ssh_pass=cisco

[XR]
R2 ansible_host=172.16.101.99 ansible_user=cisco ansible_ssh_pass=cisco

[ALL:children]
IOS
XR
```

Script 2 Ansible Inventory File
Recuperado de *(Tamilselvan, Froehlich, & Raghunathan, 2019)*

En el *Script 2 Ansible Inventory File*, se identifican dos equipos de red de nombre **R1** y **R2** con sus respectivas direcciones IP y credenciales de SSH. Además, se crearon dos grupos, uno para equipos con **IOS**, grupo al que pertenece R1 y otro grupo para equipos con **XR** al cual pertenece R2. Es importante mencionar que se puede usar el servicio DNS o el archivo *hosts* dentro del equipo de Ansible para evitar poner las direcciones IP en *Inventory File*.

Ansible *Playbooks*

Un *playbook* en Ansible es el archivo que controla las acciones a realizar en los equipos. Están escritos en YAML (ver ***Anexo A: Formatos y Estructuras de Datos para NetDevOps***), el cual es un formato de datos intuitivo y de fácil lectura para el humano.

Tal como se revisó anteriormente, Ansible *Playbooks* hacen uso de módulos para ejecutar tareas específicas en equipos, considerándolos como *scripts* pre-diseñados. Para usar estos módulos, se puede realizar mediante el CLI de Ansible (forma directa), a través del *playbook* o con APIs. Se pueden construir módulos totalmente personalizados.

Entre los módulos más útiles en entornos Cisco están:

- ios_command / ios_config
- iosxr_command / iosxr_config

```
cisco@ansible-controller:~$ ansible-doc -l | grep ^ios

ios_banner : Manage multiline banners on Cisco IOS
devices

ios_command : Run commands on remote devices
running Cisco IOS

ios_config : Manage Cisco IOS configuration sections
```

Script 3 Módulos de Ansible mediante CLI
Recuperado de *(Tamilselvan, Froehlich, & Raghunathan, 2019)*

En caso de ejecutar individualmente un determinado módulo desde el CLI de Ansible, se aplica la siguiente sintaxis:

ansible <dispositivo_*Inventory File*> -m <nombre_módulo> -a <comando_específico>

```
$ ansible IOS -m raw -a "show ip int brief"
R1 | SUCCESS | rc=0 >>Interface
IP-Address      OK? Method Status                Protocol
GigabitEthernet1       172.16.101.98   YES TFTP   up                    up
GigabitEthernet2       10.0.0.5        YES TFTP   up                    up
Loopback0              192.168.0.1     YES TFTP   up                    up
Loopback101            1.1.1.101       YES manual administratively down down
Shared connection to 172.16.101.98 closed.
Connection to 172.16.101.98 closed by remote host.
cisco@ansible-controller:~$
```

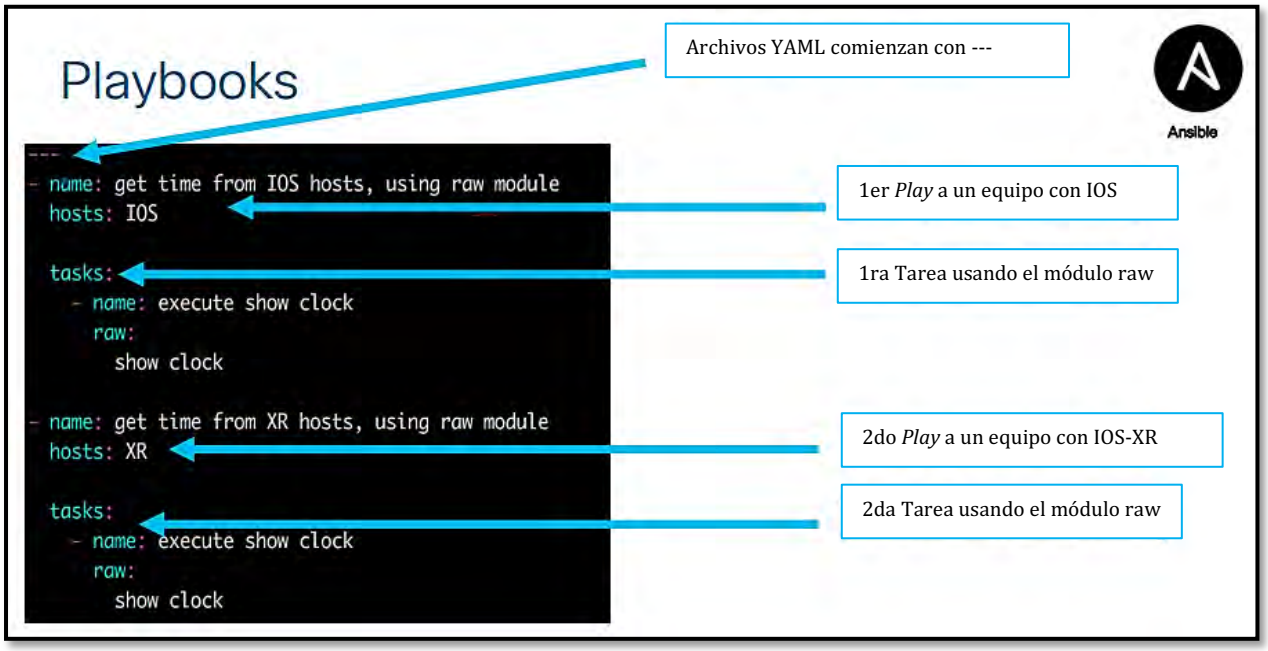
Figura 2-38 Uso de Módulo mediante Ansible CLI
Recuperado de *(Tamilselvan, Froehlich, & Raghunathan, 2019)*

La *Figura 2-38 Uso de Módulo mediante Ansible CLI* permite obtener el resultado del comando *show ip interface brief* de los equipos del grupo IOS, en este caso R1, desde la consola de configuración de Ansible, dando así características de telemetría básica.

Analizando más a profundidad un *playbook* de Ansible, este archivo constituye una colección de corridas del proceso de automatización, también llamado *plays*. Cada *play* es un conjunto de tareas, mientras una tarea es un conjunto de módulos.

Además de los módulos **ios_config** y **ios_commands** para equipos con Cisco IOS, existe un módulo que suele ser empleado, el módulo **raw.**

El módulo raw en Ansible, permite enviar que cualquier comando (según documentación de Ansible, es un *low and dirty command*[24]) al dispositivo y obtener una respuesta rápida en la consola de Ansible.



Script 4 Estructura básica de un Ansible Playbook
Basado en *(Tamilselvan, Froehlich, & Raghunathan, 2019)*

Dentro de un *playbook*, en especial para la etapa de control de acceso y conexión entre Ansible y la infraestructura a automatizar, se utilizan variables, las cuales se las puede definir en el *Ansible Inventory File* o directamente en el *Ansible Playbook*.

Una variable en Ansible se emplea para almacenar información que podría cambiar en los *hosts* o equipos; y para llamarlas en el *Playbook* se usa el símbolo de doble llave entre comillas **"{{ }}"**.

```
---
- name: Backup IOS-XR Config
  hosts: ios
  gather_facts: false
  connection: local

  vars:
      host: "{{ ansible_host }}"
      username: "{{ ansible_user }}"
      password: "{{ ansible_ssh_pass }}"
```

Script 5 Variable (vars) en un Ansible Playbook
Recuperado de *(Tamilselvan, Froehlich, & Raghunathan, 2019)*

[24] Módulo Raw: https://docs.ansible.com/ansible/latest/modules/raw_module.html

Lazos y Condicionales en Ansible *Playbooks*

Un lazo o *loop* es una tarea repetitiva y se usa con valores o variables definidas mediante la estructura ***loop:*** o ***with_items:***. El *Script 6 Lazo (with_items:) en un Ansible Playbook* muestra un lazo para recolectar información de un equipo Cisco (*show versión* y *show runnning-config*).

Un condicional por su parte se emplea para realizar una determinada tarea siempre que se cumpla una condición o estado (al regresar la condición *true*). Funciona mediante la estructura ***when:***. El *Script 7 Condicional (when:) en un Ansible Playbook* muestra el uso del comando *show ip interface brief* de un equipo Cisco siempre que su nombre sea R1.

```
tasks:
  - name: Collect Rtr Ver and Cfg
    ios_command:
       authorize: yes
       commands: "{{ item }}"

    with_items:
          - show version
          - show run
```

Script 6 Lazo (with_items:) en un Ansible Playbook
Recuperado de *(Tamilselvan, Froehlich, & Raghunathan, 2019)*

```
tasks:
 - name: Collect Router Version
   ios_command:
    authorize: yes
    commands:
          - show ip int bri
   when: inventory_hostname == " R1"
```

Script 7 Condicional (when:) en un Ansible Playbook
Recuperado de *(Tamilselvan, Froehlich, & Raghunathan, 2019)*

Reutilización de *Playbooks* y *Jinja2 templates*

La reutilización de un *Playbook* dentro de otro *Playbook* es una acción común. Se realiza importando un archivo de extensión YML mediante ***import_playbook*** o ***import_task***.

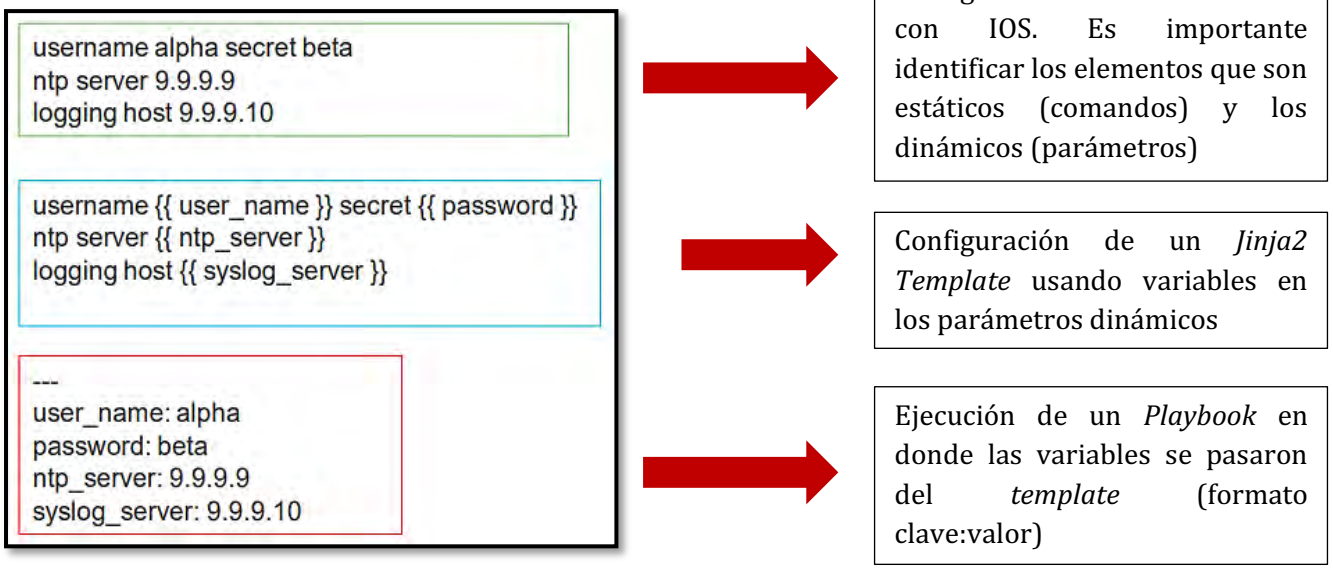
```
---
- name: route summary from IOS routers
  import_playbook: p2-ioscmd.yml

- name: route summary from XR routers
  import_playbook: p3-xrcmd.yml
```

Script 8 Llamada de un Ansible Playbook en un Ansible Playbook
Recuperado de *(Tamilselvan, Froehlich, & Raghunathan, 2019)*

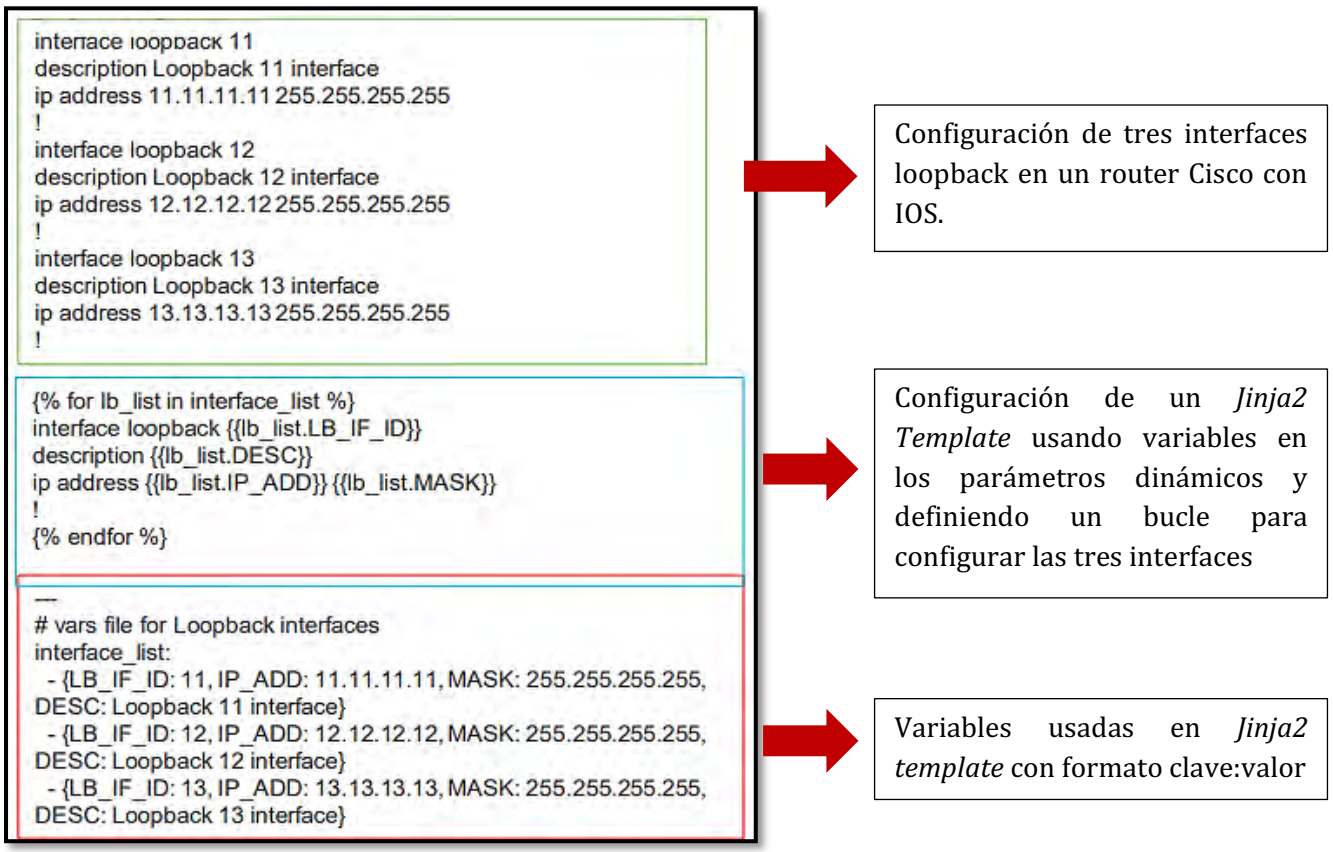
Al igual que importar un *Playbook*, también es posible generar *templates* o scripts pre-diseñados que faciliten las tareas de automatización. Una manera es mediante *Jinja2*, *template* que puede ser llamado e interactuar con *Playbooks.*

Jinja2 es un *template engine* con formato escrito en Python y con extensión .j2, el cual contiene variables y expresiones lógicas que son evaluados para ser reemplazados con valores actuales escritos en las variables declaradas en el *Playbook.*

Las variables escritas en *Jinja2 Template* son declaradas entre doble llaves **{{ }}** y los bucles usando llaves y signo de porcentaje **{% … %}.**



Script 9 Ejemplo básico de Jinja2 Template en un Ansible Playbook
Basado en *(Tamilselvan, Froehlich, & Raghunathan, 2019)*



Script 10 Ejemplo con bucles y variable en Jinja2 Template de un Ansible Playbook
Basado en *(Tamilselvan, Froehlich, & Raghunathan, 2019)*

Con el fin de poner en acción entornos de automatización de configuraciones y procesos de red con Ansible, el ***Anexo D: Ansible para entornos NetDevOps en infraestructuras de Red*** dará una visión general de la utilidad de esta herramienta en una infraestructura **VXLAN** con equipamiento ***Open-hardware*** **(Cumulus Linux).**

De igual manera, Gustavo Salazar, autor de la presente tesis, realizó una conferencia virtual con el tema "***Ansible y SDN en acción: los pilares de la era de la Programabilidad***" en el que se explica la teoría y una demostración de Ansible en entornos Cisco IOS usando GNS3-VM como emulador de redes. El siguiente link muestra dicho evento internacional:

https://community.cisco.com/t5/videos-routing-y-switching/community-live-video-ansible-y-sdn-en-acci%C3%B3n-los-pilares-de-la/ba-p/3993738

Chef y Puppet

La filosofía de *NetDevOps* combina el adecuado desarrollo de *software* con las operaciones de infraestructura en el campo de las Tecnologías de la Información, para lo cual se emplean diversas herramientas que facilitan esa integración y automatización de funciones, una de ellas Ansible tal como se analizó en la sección anterior, por lo que es momento de mencionar a las dos herramientas que junto con Ansible forman el principal ecosistema de desarrollo en el área del *networking:* Puppet y Chef.

Antes de empezar con el análisis, es necesario definir el concepto de *Managers* de Configuración, los cuales no son más que equipos/servidores que dotan de una capa de abstracción a la red, entre la configuración deseada por el administrador de la red y la infraestructura en sí, dando conceptos de idempotencia, monitoreo de acciones y automatización explicado en capítulos anteriores, creando un ecosistema y ambiente efectivo, donde se almacena datos para tomar decisiones y acciones a futuro.

Puppet y Chef están entre los primeros, más conocidos y principales *Managers* de Configuración.

Puppet

Puppet es una herramienta *open-source* de administración y configuración centralizada y automatizada de recursos tanto para el área de servidores y DCs como en el campo de las redes de datos. En la actualidad existen desarrollos comerciales que brindan soporte como *Puppet Enterprise*[25].

Puppet, al igual que Chef, están escritos en base al lenguaje Ruby[26] para establecer sus procesos de automatización, pero para Puppet, se maneja su propio lenguaje de configuración por motivos de simplicidad denominado *Puppet DSL – Domain Specific Language,* creando *scripts* denominados **manifiestos**, en los cuales se define el estado deseado de la infraestructura. La configuración completa, la cual puede estar compuesta por uno o más manifiestos, se conoce como **catálogos**.

Una diferencia entre Puppet y Ansible, es que Ansible es *agentless*, mientras Puppet es *agent-based*, lo que implica que, para conseguir automatizar una red, un agente debe estar instalado en

[25] Puppet Enterprise: https://puppet.com/products/puppet-enterprise/
[26] Ruby: https://www.ruby-lang.org/es/

cada uno de los nodos a ser administrados y configurados, lo que en ciertas ocasiones puede ser una desventaja, especialmente en entornos de múltiples fabricantes, en esos casos se utiliza un elemento denominado *Puppet Proxy Agent*.

La arquitectura base de Puppet se puede ver en la siguiente imagen.

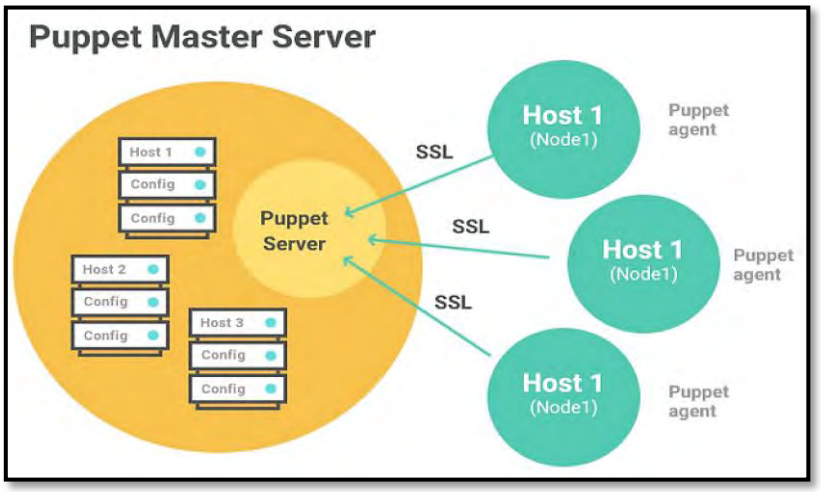


Figura 2-39 Esquema de Puppet
Recuperado de *(Yigal, 2017)*

Puppet funciona bajo en concepto de cliente/servidor, donde los clientes son los nodos administrados que tienen el agente Puppet instalado y el servidor es el equipo que centraliza las configuraciones. Este último se denomina *Puppet Master-Server*.

De igual manera permite tener un control de cambios y versiones al realizar verificaciones periódicas (por defecto cada 30 minutos), dando así visibilidad al administrador, además de control de errores, logs e información sobre los equipos (*Puppet Facter*)

Existe además un repositorio denominado *Puppet Forge*[27] donde la comunidad comparte sus avances, mejoras y adiciones de módulos a ser usados en Puppet.

Conexión entre la infraestructura y Puppet Master-Server

Es claro decir que equipos de red necesitarán de un *Puppet Proxy-Agent*, el cual no es más que un dispositivo que pueda realizar conexiones SSH con la infraestructura (TCP 22) y establecer una conexión al *Master-Server* (TCP 8140) mediante certificados SSL válidos.

El *Puppet Proxy-Agent* debe contar con un archivo que identifique al servidor, así como a los nodos, el primero bajo el nombre de **puppet.conf** y el segundo **device.conf** en el *path* **/etc/puppet**.

```
[agent]
  server = puppet.ejemploPHDTesis.com
```

Script 11 Archivo puppet.conf – Puppet Proxy Agent
Fuente: *Autor*

[27] Puppet Forge: https://forge.puppet.com/

```
[SW1.ejemploPHDTesis.com]
type cisco
url ssh://puppet:password@SW1.ejemploPHDTesis.com/?enable=UNLP
```

Script 12 Archivo device.conf – Puppet Proxy Agent
Fuente: *Autor*

En los *Scripts 11 y 12* se establece la conexión de un *Puppet Proxy-Agent* hacia un *Puppet Master* en la dirección IP correspondiente a puppet.ejemploPHDTesis.com y la conexión hacia un *switch* de nombre SW1 en la dirección IP correspondiente a SW1.ejemploPHDTesis.com. La URL a este equipo debe contener el usuario y contraseña SSH para ingresar, así como el *enable secret* para ingresar al modo de configuración global.

Según la documentación de (Puppet, 2021), se sugiere tener un NTP Server y DNS Server.

Para inicializar la comunicación, en el *Puppet Proxy Agent* se requiere del comando **puppet device** y en *Puppet Master* **puppet cert sign SW1.ejemploPHDTesis.com**

```
$ puppet device SW1.ejemploPHDTesis.com --verbose
applying configuration to SW1.ejemploPHDTesis.com at
ssh://SW1.ejemploPHDTesis.com/
Info: Creating a new SSL key for SW1.ejemploPHDTesis.com
Info: Caching certificate for ca
Info: Creating a new SSL certificate request for SW1.ejemploPHDTesis.com
Info: Certificate Request fingerprint (SHA256):
65:FC:AA:E3:F5:E5:5D:05:D0:D9:…
Info: Caching certificate for ca
```

Figura 2-40 Ejecución comando puppet device
Fuente: *Autor*

Puppet *Manifest* en Puppet Master

Un manifiesto es el archivo donde se encuentra la configuración que será enviada a los nodos en Puppet y tienen extensión .pp. La característica fundamental es que al tener un lenguaje propio de Puppet (*Puppet DSL)*, se emplea el paradigma de programación declarativa tipo JSON o YAML en lugar de imperativa como en bash o Python.

```
# Configurando la Interfaz e1/1 de SW1 como puerto L2 en Puppet
cisco_interface { "Ethernet1/1" :
 switchport_mode => enabled,
}
```

Script 13 Archivo ConfSW1.pp – Puppet Manifest
Fuente: *Autor*

Se debe tener en cuenta que es factible el uso de módulos e importarlos de *Puppet Forge* para facilitar ciertas configuraciones típicas, como en un *switch* la creación de VLANs o en un *router* instanciar el enrutamiento mediante OSPF.

Chef

Junto con Ansible y Puppet, Chef es una de las herramientas *open-source* más usadas en el contexto de *NetDevOps*. Tiene características similares con Puppet, pues está escrito en Ruby, utiliza un modelo basado en agente y tiene configuración declarativa, aunque es posible alinearlo más al estilo de Ruby (imperativo).

El *Script 14 Puppet Manifest vs. Chef Cookbook – Hello World* muestra la comparación de *Hello World* escrito como Manifiesto (de Pupper) y *Cookbook* (de Chef).

```
# Puppet
file { '/tmp/hello'
 ensure => 'present',
 content => "hello, world!",
}
```

```
# Chef
file '/tmp/hello' do
 content 'hello, world!'
end
```

Script 14 Puppet Manifest vs. Chef Cookbook – Hello World
Recuperado de *(Yigal, 2017)*

Como se observa en la *Figura 2-41 Esquema de Chef*, existe un dispositivo denominado *Chef Workstation*, el cual es una máquina que administrará la red; en dicha computadora, estarán las herramientas necesarias para automatizar la infraestructura. El nombre del archivo que contiene la configuración de los equipos se llama *recipes* o recetas en español. El conjunto de recetas, librerías, archivos, módulos, pruebas, se denomina *Cookbook*.

La siguiente figura muestra la estructura básica de Chef.

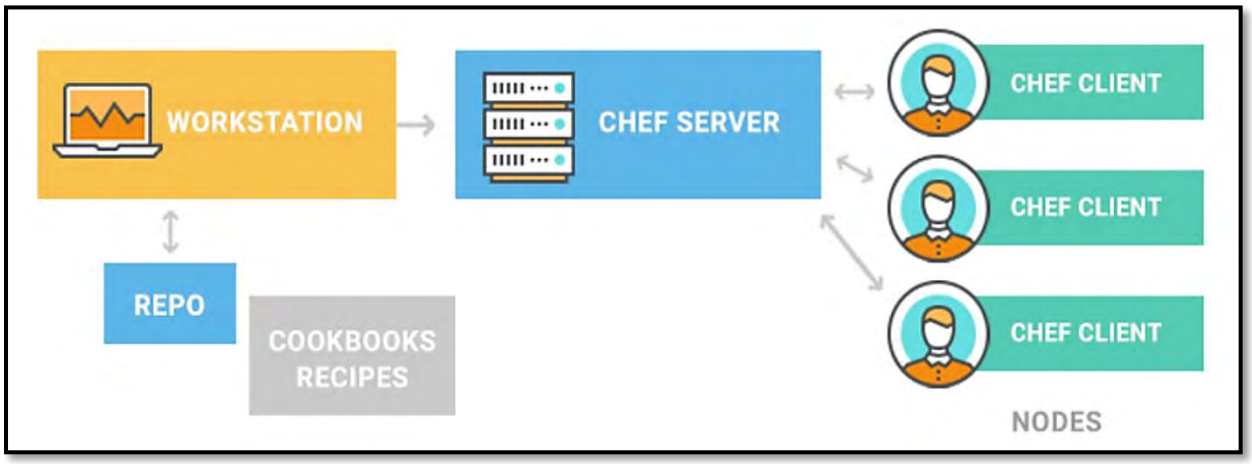


Figura 2-41 Esquema de Chef
Recuperado de *(Yigal, 2017)*

Otra de las herramientas presente en *Chef Workstation* es *Chef Repo*, el cual es el repositorio general en donde tanto las recetas y *Cookbooks* son creados y se sugiere se tenga en conjunto con un control de versión como Git.

La línea de comando de *Chef Workstation* se conoce como *Chef* y *Knife* y esta corre en una instancia denominada *Chef Infra-Server* o *Chef-Server*, el cual es considerado el punto de central de control y administración de la infraestructura.

Por otro lado, los nodos o dispositivos a configurar deben contar con un *software* que los convierta en *Chef Infra-Clients*. Muchos fabricantes tienen en sus NOS (*Network Operating Systems*) soporte integrado para Chef, tanto en ambientes físicos o virtuales. Según (Jackson, Gooley, Iliesiu, & Malegaonkar, 2020), Cisco Systems, F5, Arista, entre otros, cuentan con el cliente de Chef integrado en sus equipos.

Para la comunicación *Chef Infra-Server* y *Chef Infra-Client*, se utiliza un mecanismo de registro mediante pares de llaves públicas RSA, con el fin de sincronizar *Cookbooks* sólo con los dispositivos debidamente autenticados.

Chef cuenta con una herramienta con las mismas funciones de recolección de información (Utilización de CPU, memoria, OS, etc) que *Puppet Facter*, el cual se llama *Chef Ohai*[28]

Para el uso de módulos prediseñados y facilitar la configuración mediante Chef, existe *Chef Supermarket*[29].

Ejemplo de *Chef Cookbook*

Para ilustrar la configuración de un par de *Cookbooks*, los siguientes *scripts* sirven para la configuración IP de una interfaz L3 y el segundo *script* para la asignación de una interfaz L2 a la VLAN 10 en equipos Cisco con IOS.

```
cisco_interface 'Ethernet1/1' do
  action :create
  ipv4_address '192.168.10.10'
  ipv4_netmask_length 24
  ipv4_proxy_arp true
  ipv4_redirects true
  shutdown false
  switchport_mode 'disabled'
end
```

Script 15 Chef Cookbook para la configuración IP de una interfaz L3
Fuente: *Autor*

```
cisco_interface 'Ethernet1/1' do
  action :create
  access_vlan 10
  shutdown false
  switchport_mode 'access'
  switchport_vtp true
end
```

Script 16 Chef Cookbook para la configuración de una interfaz L2 como parte de VLAN 10
Fuente: *Autor*

[28] Chef Ohai: https://docs.chef.io/ohai/
[29] Chef Supermarket: https://supermarket.chef.io/

Netmiko, NAPALM y Nornir: Herramientas tipo Python *Library*

El contexto de *NetDevOps* entrega una gran cantidad de herramientas útiles para tener automatización y programabilidad en las redes, la pregunta ahora sería, cuál de todas usar. La respuesta depende tanto de la situación, equipos, tipo de infraestructura o incluso de las habilidades de los profesionales en Tecnologías de la Información.

Una herramienta adicional es el uso de librerías en programas escritos en Python, lenguaje considerado OOP u *Object-Oriented Programming* tanto para trabajar con datos estructurados y no estructurados, así como funcionar como herramienta *NetDevOps*, para con ello obtener libertad total de administración, configuración y automatización.

Con el fin de lograr la comunicación entre el dispositivo central de configuración (o controlador de configuración) y la infraestructura de TI, es imprescindible un protocolo seguro que permita el envío y recepción de mensajes, inmediatamente, saltarán las siglas SSH (*Secure Shell*). **Paramiko**[30] es una implementación de SSH para Python usado para interconectar distintos *hosts*, sin embargo, resultó ser complicado para ciertos equipos en especial en procesos de autenticación, es así que surgió **Netmiko**[31]**,** una librería Python construida sobre Paramiko diseñada para solventar esas dificultades.

Ahora, una vez que se tiene claro la forma de conexión, se debe definir cuál es la manera de generar procesos automáticos basado en programabilidad a través de Python.

NAPALM[32], considerado una especie de capa de abstracción o API que se ubica sobre cualquier protocolo, diseñado para establecer comunicación entre el equipo de automatización y la infraestructura (SSH, SSL, TLS, etc.) dando la posibilidad de obtener, fusionar o reemplazar una configuración enviada a distintos NOS (*Network Operating Systems*), así como realizar acciones de telemetría y obtención de datos. NAPALM tiene la posibilidad de trabajar junto con Ansible/Puppet/Chef para simplificar los *Playbooks*/Manifiestos/*Cookbooks*.

NAPALM es una excelente librería para automatización, pero sigue dando inconvenientes al momento de generar inventarios de equipos, recolectar datos de la red, así como tiene poca capacidad de *multi-threading*, es decir, bajo desempeño para ejecutar tareas en cientos de equipos a la vez, es ahí donde se considera a **Nornir** una buena opción.

Nornir[33] provee la misma funcionalidad que Ansible, pero en un entorno 100% escrito en Python, lo cual es ideal para tener libertad de programación y generar entornos mucho más efectivos a medida que la cantidad de nodos a programar aumenta. Es un *framework* que podría combinarse con NAPALM/Netmiko y así obtener un entorno realmente flexible.

Para verificar la factibilidad de uso de Netmiko y NAPALM con equipos tradicionales reales, el ***Anexo E: Pruebas de Concepto de Netmiko y Napalm – Telemetría*** permite analizar la ejecución de un par de PoCs de estas herramientas en un entorno con infraestructura tradicional.

[30] Paramiko: http://www.paramiko.org/
[31] Netmiko: https://pypi.org/project/netmiko/
[32] NAPALM: https://napalm-automation.net/
[33] Nornir: https://nornir.readthedocs.io/en/latest/

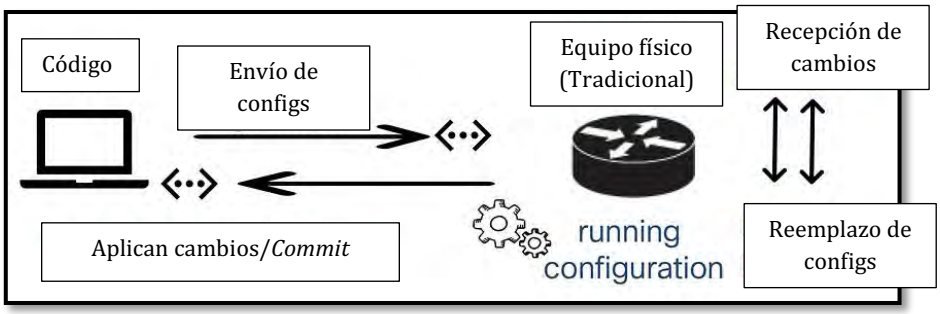


Figura 2-42 Esquema de NAPALM
Recuperado de *(Clark, 2019)*

2.4 Comparación entre SDN y otras formas de virtualización

Hank Preston, Cisco DevNet *Developer Evangelist*, identificó las cuatro Eras del *Networking*, estableciendo su evolución, las cuales se pueden resumir en la *Figura 2-43*.

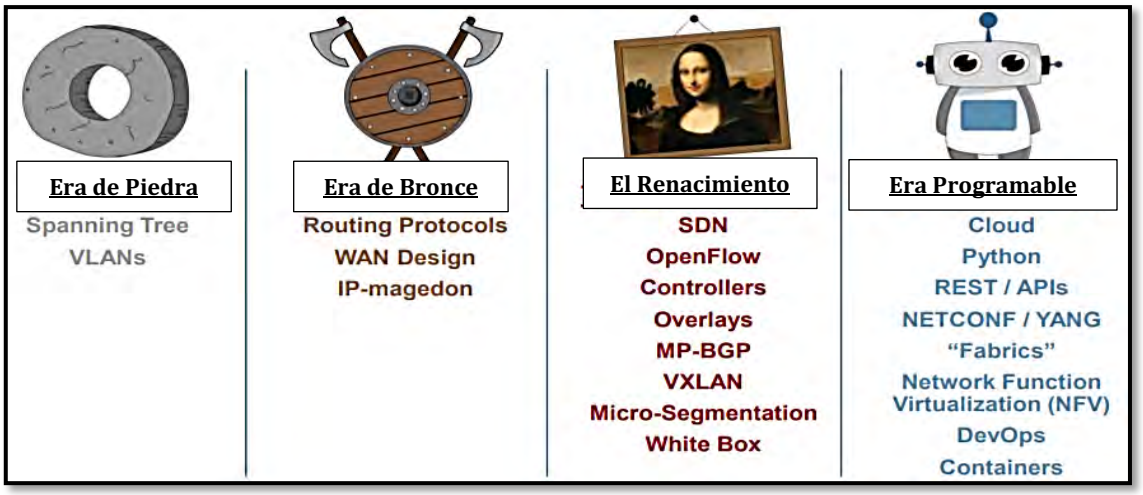


Figura 2-43 Las Cuatro Eras del Networking
Recuperado de *(Salazar-Chacón, Naranjo, & Marrone, 2020)*

La "Era de Piedra" y la "Era de Bronce" del *networking* son las eras de las redes tradicionales y monolíticas. MPLS es considerado el eslabón entre la era de Bronce y la "Era del Renacimiento".

La "Era del Renacimiento" y la "Era Programable" son las nuevas eras del *Networking*.

Tal como se ha mencionado anteriormente, el futuro de las redes de datos es la automatización, y *NetDevOps* tiene en su núcleo a la automatización mediante APIs.

El *Networking* requiere de cierta innovación tecnológica para hacer posible esta programabilidad y automatización, el primer paso fue permitir que el *software* controle plenamente el *hardware*, pero para que ello ocurra, es necesario del uso de *hardware* genérico y estándar que fomente la innovación, dando así origen a *OpenNetworking* y *Software Defined Networking* (SDN); con este ecosistema, se llegaría a una automatización de red completa, por lo tanto SDN es el eslabón para migrar de la "Era del Renacimiento" a la "Era de la Programabilidad".

Software Defined Networking

Los retos principales de las redes tradicionales son su elevado CAPEX y OPEX en su implementación, presión externa por parte de proveedores de contenido, crecimiento exponencial de uso de ancho de banda en las infraestructuras, provisión manual tanto para despliegue como monitoreo y resolución de problemas, además de equipos y protocolos propietarios que no interactúan adecuadamente, así como el uso masivo del *cloud*, redes extremadamente complejas, poco escalables y lentas, por nombrar los retos más relevantes que han sido explicados en páginas anteriores.

Estas presiones a la infraestructura motivaron a la innovación en las redes, dando cabida en primer lugar a SDN y a NFV, ya que las redes tradicionales no operan adecuadamente en términos de rendimiento con las redes empresariales y más aún en DCs y SPs modernos.

SDN según lo analizado implica la separación del plano de control del plano de datos o funciones de envíos de paquetes. En base al criterio de Ed Tittel en (Tittel, 2018), SDN define esencialmente el *"big picture"* de las redes del presente y futuro: una infraestructura deseada, en donde se tiene control mediante políticas inteligentes y visibilidad total, justamente esa característica es el nexo con *NetDevOps*, contar con un control centralizado.

Los elementos clave de SDN se establecen en la *Tabla 2-7*

Tabla 2-7 Puntos Clave de SDN

Puntos clave de SDN	Descripción/Características
Control de Red Programable	Habilidad de aprovisionar nuevos elementos de red y equipos o reconfigurar los existentes desde un conjunto de interfaces programables, permitiendo a los administradores de la red configurarla de manera fácil, empleando herramientas de *scripting* o mediante APIs propias o de terceros y entornos GUI.
Agilidad en la respuesta (Acción reactiva inmediata)	SDN permite el ajuste dinámico de los flujos de datos con el fin de adaptarse rápidamente a las necesidades particulares de la infraestructura y la demanda de usuarios.
Inteligencia Centralizada (Controladores)	Los controladores, implementados a nivel de *software* (virtualizados) o *hardware*, mantienen una vista coherente de toda la red. Desde el punto de vista de las aplicaciones, SDN sería como "un solo *switch* lógico".
Configuración Programable	Los administradores de la red podrán configurar, controlar, asegurar y modificar los recursos de red usando programas SDN automáticos, incluso creando sus propias APIs a través de protocolos e interfaces estandarizadas.
Estandarización y *Vendor-Neutral*	La estandarización, el uso de *Open-Hardware-OpenNetworking* y la interoperabilidad es un aspecto extremadamente relevante.

Fuente: (Tittel, 2018)

Network Functions Virtualization (NFV)

NFV lleva la virtualización y abstracción de la red a un próximo nivel al separar no sólo el plano de control del plano de datos, sino toda función que un equipo puede hacer del *hardware* que lo contiene, es decir, mediante VMs un equipo puede funcionar como *router***,** en otro momento como *switch, firewall, load-balancer,* o cualquier otra función requerida en el momento, por lo que las funciones de red se ejecutan en cualquier nivel y lugar de la red, desde las periferias hasta el núcleo, aceptando, reenviando, realizando *shaping* o filtrando el tráfico de red a medida que este cursa la infraestructura.

Los puntos más importantes de NFV se indican en la *Tabla 2-8*.

Tabla 2-8 Puntos Clave de NFV

Puntos clave de NFV	Descripción/Características
Software virtualizado en lugar de *hardware* dedicado	Esta característica significa que los servicios de red como enrutamiento, conmutación, balanceo de carga, optimización WAN, pueden ser reemplazados por VMs, las cuales se gestionan y orquestan bajo un hipervisor o controlador si la infraestructura subyacente es SDN.
Reducción substancial de CAPEX y OPEX	Los servicios de red, cualquiera que estos se requieran, podrán correr bajo servidores con procesador ARM, x86 o de cualquier tipo, reduciendo así los gastos de capital y operativos. Debido a la flexibilidad que representa tener un servidor, principalmente porque es posible aumentar o disminuir su capacidad en base a las necesidades del *software*, no es necesario sobredimensionar una red para momentos de gran demanda como tradicionalmente se hacía. De igual manera genera ahorros en cuanto a espacio y consumo eléctrico (Xu, Liu, Wang, & Xu, 2016).
Iniciativa generada por la Industria	NFV es una iniciativa propuesta en sus inicios por AT&T, BT (*British Telecom*), Deutsche Telekom y otros grandes telecos a nivel mundial. Actualmente NFV está coordinado y estandarizado a través de ETSI (*European Telecommunications Standard Institute*)

Fuente: (Tittel, 2018)

SDN y NFV usan abstracciones de red, pero cada uno con una aproximación y ejecución distinta, tiene similitudes y diferencias las cuales se resumen en la *Tabla 2-9*.

Tabla 2-9 Similitudes y Diferencias entre NFV y SDN

Similitudes	
Emplean Abstracciones de Red	El uso de la virtualización es evidente tanto en SDN como en NFV, sin embargo, cada uno lo hace a un nivel de profundidad distinto. El primero separa las funciones del plano de control del plano de datos, mientras el segundo se basa en la total separación de cualquier función del *hardware.*
Pueden funcionar en	Es ideal, hoy en día, ejecutar SDN sobre una infraestructura NFV. En este caso, SDN reenvía los paquetes de un dispositivo a otro al mismo

conjunto: SDN y NFV	tiempo: las funciones del plano de control para el enrutamiento, establecimiento de políticas, conmutación, corren en un VM en algún lugar de la red o en el *Cloud*. En este caso, NFV entrega dichas funcionalidades de red denominadas VNF (*Virtual Network Functions*) y SDN los controla y orquesta de forma programática.
Mejoran la efectividad de la infraestructura	SDN y NFV mejoran sustancialmente el desempeño de la red, tanto a nivel operativo, funcional como económico y en experiencia de usuario.
Crecimiento del Mercado	El estudio de proyección realizado por *Infonetics Research* y publicado en (Ingram Micro, 2017), el mercado global para *hardware y software* creció de $USD500 millones en 2013 a $11 billones en 2018 y se prevé que crezca mucho más hasta el 2025, debido a efectos de la pandemia COVID-19.
Diferencias	
Niveles de abstracción	Una de las diferencias entre SDN y NFV radica en la manera en cómo separan y abstraen los recursos de la red. SDN abstrae recursos de red físicos como *routers* y *switches*, envía la toma de decisiones al plano de control centralizado, el cual decide cómo enviar el tráfico y el *hardware* implementa en sí el plano de datos. NFV, por el contrario, virtualiza todos los recursos físicos de red gestionados a través de un hipervisor, permitiendo de esa manera el crecimiento exponencial de la red sin adicionar más equipos físicos.
Áreas de Operación	SDN fue diseñado para funcionar en una red de campus, DC o incluso en un ambiente enfocado al *Cloud*, mientras NFV se enfoca más a infraestructuras de proveedores de servicio.
Organizaciones y Protocolos que los apalancan	SDN, tal como lo hemos visto, está soportado por ONF, generando *OpenSDN* implementado mediante *OpenFlow* como protocolo *Southbound*, mientras NFV por ETSI NFV *Working Group*

Fuente: (Tittel, 2018)

2.5 OpenDaylight Project y NFV

OpenDaylight (ODL)[34], proyecto soportado por *The Linux Foundation* desde enero del 2018, nació en abril del 2013 con el fin de avanzar en la adopción y penetración de SDN, así como generar una base fuerte para NFV, todo con un enfoque de programabilidad y automatización.

Figura 2-44 OpenDaylight – Linux Foundation
Recuperado de *(OpenDaylight - The Linux Foundation Projects, 2021)*

[34] ODL: https://www.opendaylight.org/

ODL se ha convertido en la plataforma tipo controlador SDN más ampliamente implementada en tan solo 7 años de existencia. Hasta la fecha de escritura de la presente tesis, ODL ha generado 14 *releases*, tiene más de 1000 autores de contenido y la comunidad *OpenDaylight* más de un billón de subscriptores según *The Linux Foundation Projects*.

Entre los *vendors* y fabricantes de tecnología fundadores del proyecto están *Arista Networks, Big Switch Networks, Brocade-Lumina, Cisco, Citrix, Ericsson, HP, IBM, Juniper Networks, Microsoft, NEC, Nuage Networks, PLUMgrid, RedHat* y *VMware*.

Aluminium, lanzamiento número 13 de ODL (22 de septiembre del 2020), ha consolidado un logro que demuestra el comprometimiento de la comunidad en lograr un entorno abierto, escalable e interoperable en el mundo de SDN ya que ahora soporta nativamente *features* como *edge* y *cloud-native*, así como compatibilidad con proyectos relacionados como ONAP (*Open Network Automation Platform*)[35], Kubernetes[36], OpenStack[37] y OPNFV[38]. Cabe decir que existe un nuevo *realese* de nombre *Silicon* (lanzamiento 14 el 2 de abril del 2021), pero a la fecha de la escritura de esta tesis, la versión estable es la 13.0 *Aluminium*.

Acerca de *The OpenDaylight Foundation*

El objetivo de ODL *Foundation* es facilitar la colaboración entre desarrolladores, usuarios finales, compañías miembros de LFN (*Linux Foundation Networking*), proyectos *open source* en general y así producir soluciones tecnológicas relevantes y confiables.

El comité de dirección técnica de ODL *Foundation* (TSC)[39] provee un liderazgo sobre el camino técnico a seguir, así como da las pautas para prácticas colaborativas. El TSC es elegido por la comunidad ODL, quedando en funciones con renovación anual.

Acerca de la Arquitectura ODL

El núcleo de ODL es el llamado MD-SAL por las siglas de *Model-Driven Service Abstraction Layer,* con el que es posible representar los dispositivos de red de la infraestructura subyacente (*underlying*) como objetos, cuyas interacciones se procesan mediante el llamado SAL.

SAL es el mecanismo de intercambio de datos y modelo de adaptación entre los dispositivos de red representados por modelos YANG y las aplicaciones del entorno. En este punto es importante mencionar que los Modelos YANG proveen una descripción generalizada de las capacidades de una aplicación o dispositivo sin requerir detalles de su implementación, lo que es ideal para SAL, pues estos modelos u objetos son simplemente definidos por sus roles en una determinada interacción.

Dentro de los roles están el de productor, el cual implementa una API generando datos, mientras el rol de consumidor usa dicha API y consume los datos. Estos roles describen de mejor manera la interacción dentro de un SAL de ODL que los mismos términos de SDN *Northbound* y *Southbound* respectivamente.

ODL es considerado un entorno abierto modular y multiprotocolo, entregando el máximo de flexibilidad en un controlador al permitir la inclusión de servicios automatizados, además de

[35] ONAP: https://www.onap.org/
[36] Kubernetes: https://kubernetes.io/es/
[37] OpenStack: https://www.openstack.org/
[38] Open Platform for NFV: https://www.opnfv.org/
[39] TSC de ODL: https://www.opendaylight.org/about/governance/tsc

soportar perfectamente protocolos como *OpenFlow*, OVSDB, NETCONF, BGP, entre otros, tal como se aprecia en la *Figura 2-45.*

El nuevo *release*, *Magnesium*, da la posibilidad además de integrarse nativamente con el *networking* tradicional y determinístico mediante dos nuevos proyectos: *DetNet* y *Plastic*, así como ha mejorado los servicios para un proveedor de servicios mediante *TransportPCE* y *BGP-CEP*, incluyendo mejoras e integración con *Segment Routing* (mejoras al RFC 5440 y BGP-LS).

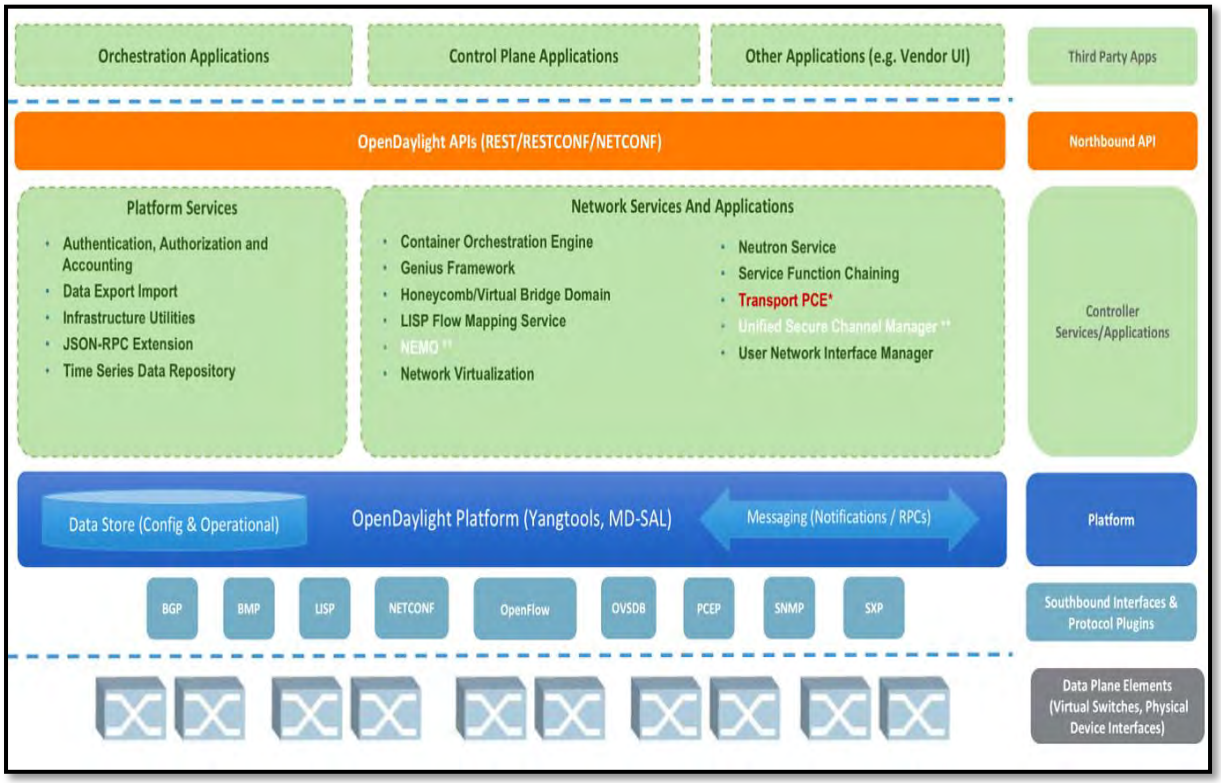


Figura 2-45 OpenDaylight Architecture– Linux Foundation
Recuperado de *(OpenDayLight - Fluorine Release, 2018)*

Seguridad en ODL

La seguridad es un eje transversal en toda comunicación, razón por la cual ODL ha mejorado en todos sus *releases* este aspecto, específicamente en tres áreas denominadas S3P o Seguridad, Escalabilidad y Desempeño.

Entre los procesos de mejora están conceptos de AAA (*Authentication, Authorization and Accounting*), así como aprovisionamiento de seguridad automática en dispositivos de red y controladores, respondiendo así a las nuevas vulnerabilidades encontradas, trabajando en conjunto con la gran comunidad ODL.

NFV: Caso de Uso

La promoción de uso de *Frameworks* y arquitecturas abiertas que brinden soporte a un nuevo ecosistema de tecnologías disruptivas ha dado la oportunidad a que tanto el empleo de *Clouds* como de NFV (*Network Functions Virtualization*) puedan ser solucionadas con la adopción temprana de una red SDN tanto en el ámbito empresarial como de proveedores de servicio (Sistemas Autónomos de Tránsito).

En el núcleo de estas nuevas infraestructuras de comunicaciones y redes está el controlador SDN, el cual mediante APIs abiertas, un conjunto de aplicaciones, soporte a un gran número de dispositivos de red y con la habilitación de programabilidad y automatización, dan paso a un control y administración de la red de forma inteligente.

La migración a este tipo de "nueva generación de redes" tiene muchos obstáculos en su camino, principalmente en el área de la estandarización, interoperabilidad y soporte con múltiples fabricantes/*vendors*, así como interoperar junto a redes tradicionales, razón por la cual la administración, monitoreo y diseño se ve impactado.

Por estas razones, se está evaluando desde el punto de vista de *carriers* e ISPs el uso de una arquitectura SDN común, la cual conciba un control dinámico, flexible y basado en políticas, pero bajo entornos programáticos; es ahí que dichos operadores ven a NFV como una solución práctica para reducir la dependencia de un hardware y herramientas de gestión dedicado, manteniendo entornos virtualizados y controlados mediante automatización.

Migrar hacia el denominado *virtualized networking* implica un cambio substancial y ODL es la arquitectura ideal para apalancar a SDN.

OpenDayLight provee abstracción y programabilidad abierta para infraestructuras definidas por software, así como total integración con entornos *cloud* de próxima generación mediante *OpenStack*, por ejemplo, a través de las *open* APIs como Neutron[40] y Neutron/Multi-L2[41] tal como se observa en la *Figura 2-46*.

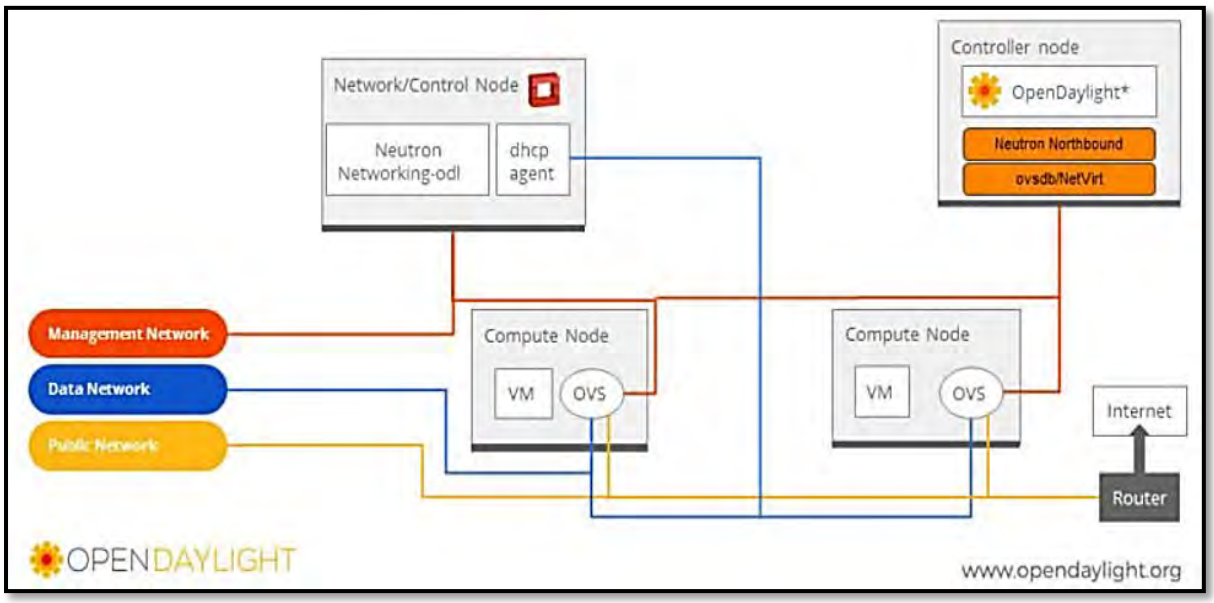


Figura 2-46 OpenDaylight y OpenStack
Recuperado de *(OpenDayLight, 2018)*

Modernos y futuros ISPs y CSPs (*Communications Service Providers*) desearán una implementación adecuada de NFV sobre SDN, y ODL es una de las plataformas en las cuales se han desarrollado pruebas de concepto o PoCs, entre ellas las series de *ETSI NFV Proof of Concepts*[42] y en particular la prueba No. 19 denominada *Service Acceleration of Network*

[40] Neutron API (OpenStack): https://wiki.openstack.org/wiki/Neutron
[41] Neutron/ML2: https://wiki.openstack.org/wiki/Neutron/ML2
[42] NFV Proof of Concepts: https://www.etsi.org/technologies/nfv/nfv-poc

Functions in Carrier Networks realizada con ODL para demostrar cómo SDN puede mejorar la adopción de NFV y orquestación de servicios a través de OpenStack. Este PoC fue auspiciado por AT&T y múltiples fabricantes de *vSwitches*.

OpenDayLight es seleccionada como una de las plataformas claves en OPNFV (*Open Platform for NFV*), proyecto *open source* administrado por la *Linux Foundation*.

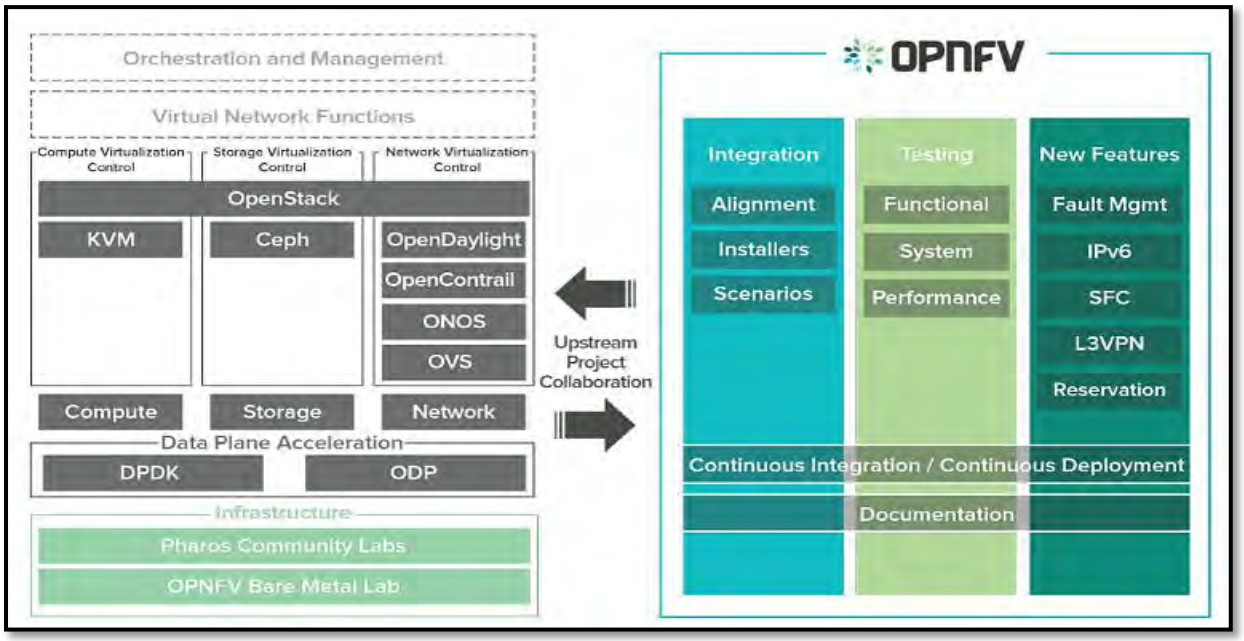


Figura 2-47 OpenDaylight y NFV: OPNFV
Recuperado de *(OpenDayLight, 2018)*

2.6 Funcionamiento de las Redes SDN basadas en OpenFlow (Open Networking Foundation)

En ambientes SDN, la abstracción del Plano de Control del Plano de Datos se da gracias al controlador SDN, en el cual se definen las políticas de enrutamiento y conectividad necesarias para que una aplicación corra dentro de cualquier infraestructura sin preocuparse de elementos físicos o ASICs (*Application-Specific Integrated Circuits*). Esas políticas son enviadas al Plano de Datos usando diversos protocolos *southbound*, entre ellos *OpenFlow*.

Cuando la infraestructura emplea *OpenFlow*, dicha infraestructura SDN se denomina *OpenSDN.*

Con el fin de implementar las políticas definidas en el controlador, el Plano de Datos que en este caso serían *OpenFlow Switches*, tienen una o más tablas de flujos (*Flow-table*), un *Meter-Table* por cada flujo y un *Group-Table*, este último para métodos adicionales de envío de datos. Las tablas de un equipo *OpenSDN* permiten realizar búsquedas rápidas de paquetes y envío de datos según políticas preestablecidas.

A nivel físico, un equipo *OpenSDN* según (Open Networking Foundation - OpenFlow Switch Spec, 2013), puede tener puertos tanto físicos, lógicos o reservados, sean necesarios, los cuales sirven como entrada o salida de flujo de datos.

La *Figura 2-48* muestra una esquematización de los componentes de un equipo (*switch*) *OpenSDN*.

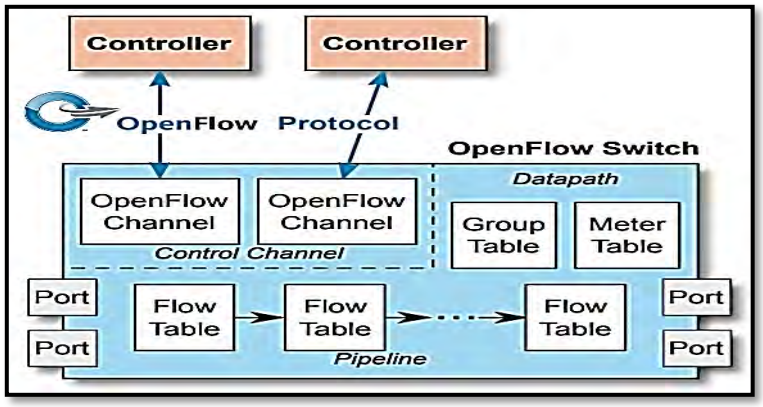


Figura 2-48 Principales componentes de un OpenFlow Switch
Recuperado de *(Open Networking Foundation, 2016)*

El controlador que usa OpenFlow en realidad no cambia ninguna configuración de los equipos del plano de datos, simplemente actualiza o modifica sus tablas de flujo, por ello, el controlador podría añadir, actualizar o borrar entradas a esas tablas con efectividad.

Para evitar únicos puntos de falla, múltiples controladores pueden coexistir en la misma infraestructura, en esos casos, *OpenFlow-Channels* son usados.

OpenFlow Entries

Existen dos entradas principales en las Tablas de flujo:

- Entrada de flujo reactivo: Es un tipo de entrada dinámica necesaria para la conectividad de extremo a extremo en la infraestructura OpenSDN.
- Entrada de flujo proactivo: Es un tipo de entrada previamente programada en el controlador mediante una determinada política de enrutamiento y *forwarding*. Esta entrada está en constante avance, pues es factible contar con inteligencia artificial para predecir comportamientos (infraestructuras *Next-Generation* o aquellas capaces de gestionar, monitorear y controlar el comportamiento antes, durante y después de un evento en la red).

2.6.1 Proceso de *Forwarding* en OpenSDN

La Tabla de flujo está compuesta por una serie de entradas las cuales se usan como campos de coincidencia según el tráfico que ingresa al equipo.

Es importante mencionar que un *OpenFlow Switch* podría tener más de una *flow table*, por ello, el proceso de *matching* o coincidencia empieza en la primera tabla de flujo o *Flow Table 0* y continúa secuencialmente a través de las demás tablas en la llamada *Table Pipeline*.

Las entradas de la tabla son secuenciales basadas en una prioridad: a mayor prioridad, el proceso de *matching* o coincidencia ocurre primero.

La actualización de los datos que ocurre de tabla a tabla se denomina *OpenFlow Metadata*.

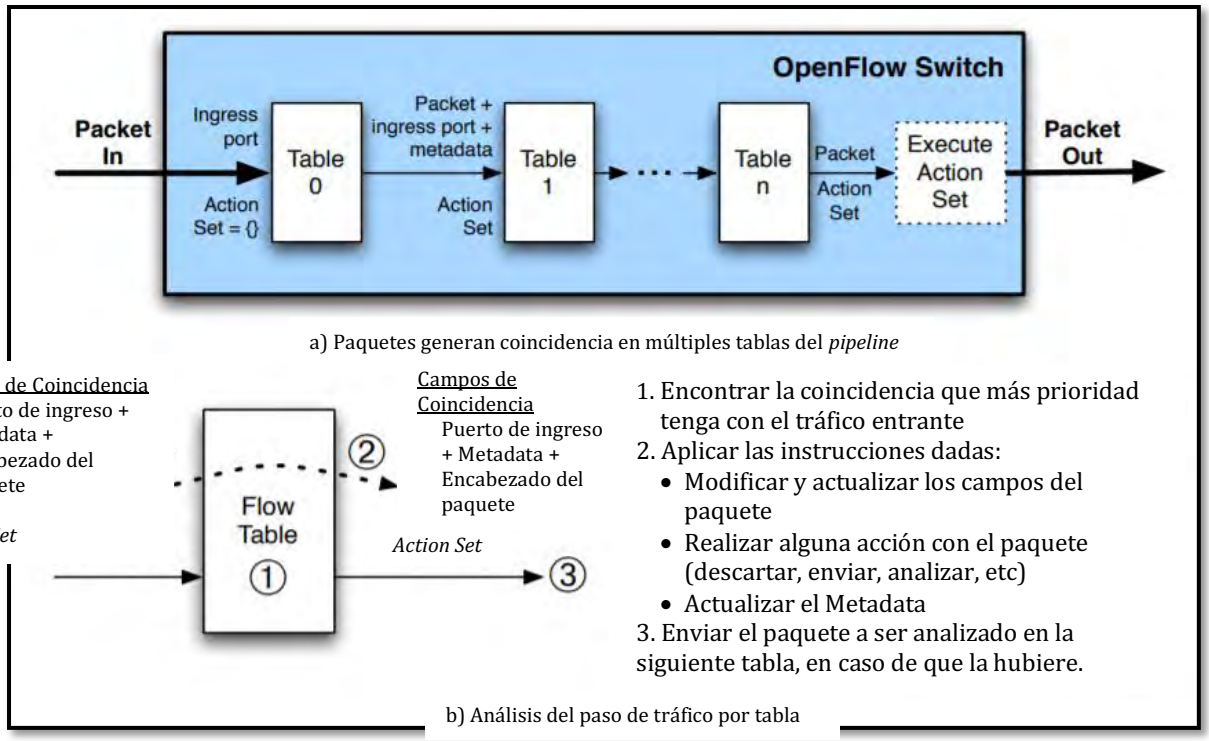


Figura 2-49 Proceso de Matching en OpenFlow
Basado en *(Open Networking Foundation - OpenFlow Switch Spec, 2013)*

En el momento en que una coincidencia se da, una acción es ejecutada, entre ellas, realizar el envío, eliminar el paquete, filtrarlo, realizar una priorización QoS, generar un *tag* de VLAN, VXLAN o MPLS, entre otras acciones. Si no existe coincidencia, la acción depende de la configuración de la tabla denominada *Table-miss Flow Entry*.

La *Figura 2-50* esquematiza el proceso de *Table Pipeline,* mientras la *Figura 2-51* el diagrama de flujo del *matching* o coincidencia de un flujo.

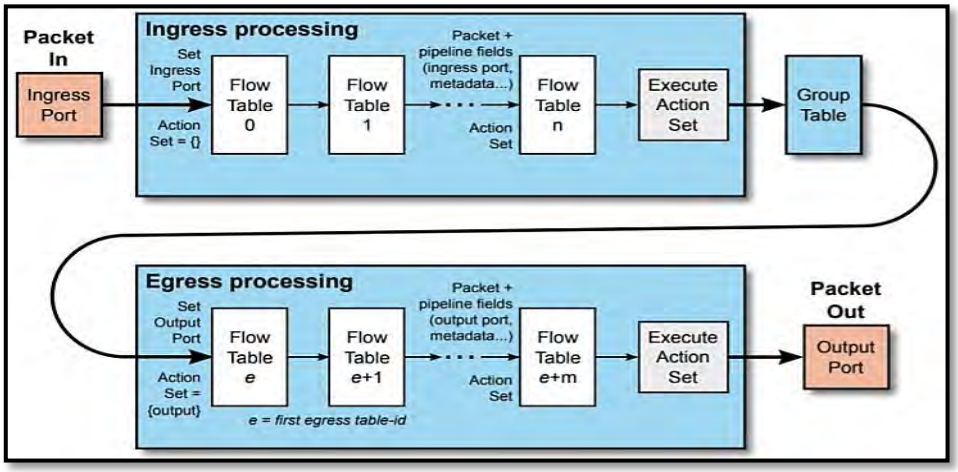


Figura 2-50* Proceso de *OpenFlow Table Pipeline
Recuperado de *(Open Networking Foundation, 2016)*

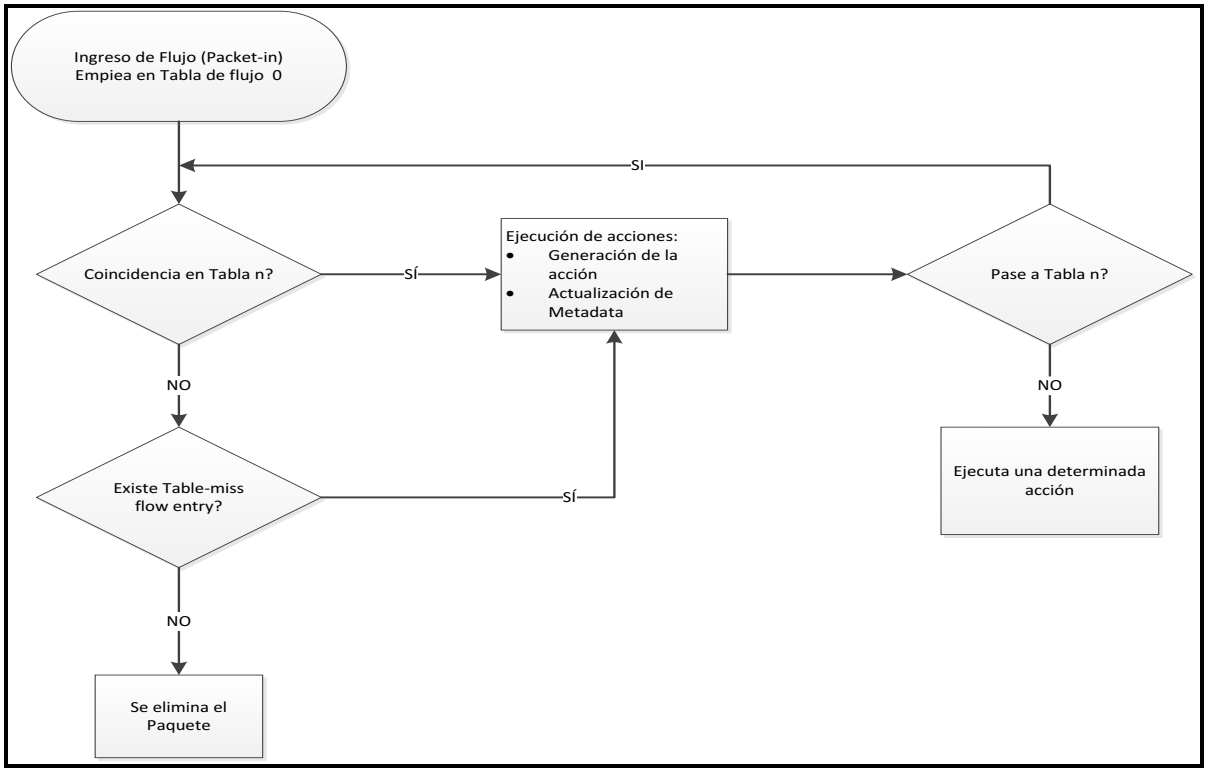


Figura 2-51 Diagrama de Packet Flow en un OpenFlow Switch Basado en (Open Networking Foundation, 2016)

2.6.2 Canal OpenFlow y Mensajes OpenFlow

El canal *OpenFlow* es el término empleado para referirse a la interfaz que conecta un *OpenFlow switch* a un controlador SDN. A través de esta interfaz, el controlador configura y administra el dispositivo que pertenece al plano de datos, así como recibe eventos y logs por parte del *switch* para enviarlas en sentido *Northbound* y notificar al administrador de la red.

El canal de control del *switch* puede soportar tanto un solo Canal *OpenFlow* con un único controlador, o múltiples canales permitiendo que más de un controlador administre el *OpenFlow Switch*.

Con el fin de establecer esa comunicación dispositivo-controlador, el protocolo *OpenFlow* define tres tipos de mensajes según (Open Networking Foundation - OpenFlow Switch Spec., 2015):

- Mensajes *controller-to-switch*
- Mensajes asincrónicos
- Mensajes simétricos

Mensajes *Controller-to-Switch*

Estos mensajes los envía primero el controlador y pueden o no tener respuesta del *switch*.

Entre los mensajes de este tipo están:

- *Features*: Uno de los mensajes iniciales en el intercambio entre controlador-switch. El controlador intentará solicitar la identidad y características básicas del *switch* a controlar enviando *feature-requests*. La respuesta por parte del *switch* es el *feature-reply*. Este proceso ocurre en el establecimiento del canal *OpenFlow*.

- *Configuration*: El controlador puede solicitar parámetros de configuración en el *switch*, para ello usa este mensaje.
- *Modify-State*: Son mensajes enviados por el controlador para administrar el estado de un *switch*, mediante las acciones de añadir, borrar o modificar entradas de flujo y setear propiedades del puerto del *switch*.
- *Read-State*: Mensajes usados por el controlador para recolectar información y estadística a nivel de Plano de Datos.
- *Packet-out*: Usados por el controlador para enviar mensajes a través de un puerto del *switch*, así como reenviar paquetes recibidos mediante mensajes *Packet-in.* Los mensajes *Packet-out* deben contener el paquete entero a enviar, junto con una lista de acciones a ser aplicadas. En caso de una lista vacía de acciones, el paquete es eliminado.
- *Role-Request y Asynchronous-Config*: Mensajes empleados para reconocer el rol que se tiene dentro de un canal OpenFlow (*role request*) y para aplicar filtros adicionales en mensajes asíncronos (*Assynchronous*-Config). Son especialmente usados cuando existen entornos multicontrolador.

Mensajes Asincrónicos

Este tipo de mensajes se envían sin una solicitud explícita del controlador al *switch* cuando llega un paquete o hay un cambio en su estado.

Entre los mensajes más importante de este tipo están:

- *Packet-in*: Mensaje utilizado para transferir el control de un paquete al controlador.
- *Flow-Removed:* Mensaje que informa al controlador sobre la remoción de un *flow entry* de la tabla de flujos que tenga la bandera OFPFF_SEND_FLOW_REM, normalmente generadas cuando se recibe una petición de borrado de flujo.
- *Port-Status*: Informa al controlador sobre el cambio en un puerto.
- *Role-Status*: Usado cuando el controlador informa de un cambio en su estado, por ejemplo, al tener un entorno multicontrolador y cuando uno de ellos se elige como *master,* el *switch* enviará un mensaje de tipo *role-status* al antiguo controlador.
- *Controller-Status*: Sirve para informar cuando ocurre un cambio en el canal *OpenFlow*, por ejemplo, cuando un controlador pierde contacto con el plano de datos.
- *Flow-Monitor*: Sirve para informar al controlador sobre un cambio en el *flow table* en un tipo de *tracking* nativo.

Mensajes Simétricos

Los mensajes simétricos se envían como parte del desenvolvimiento y operación del protocolo *OpenFlow* y sin que sean solicitados. Entre los más importantes de este tipo podemos mencionar:

- *Hello*: Intercambiados entre el *switch* y controlador una vez que empieza el proceso de conexión.
- *Echo*: Utilizado como mecanismo para saber si la conexión *switch*-controlador sigue activa. Se lo puede usar para medir latencia y ancho de banda de la red SDN.
- *Error*: Mensaje empleado como notificación de problemas de conexión, mayormente enviado por el *switch* para indicar una falla en cualquier petición iniciada por el controlador.

- *Experimenter:* Mecanismo estandarizado utilizado por *switches OpenFlow* para ofrecer funcionalidades adicionales o implementaciones futuras del protocolo.

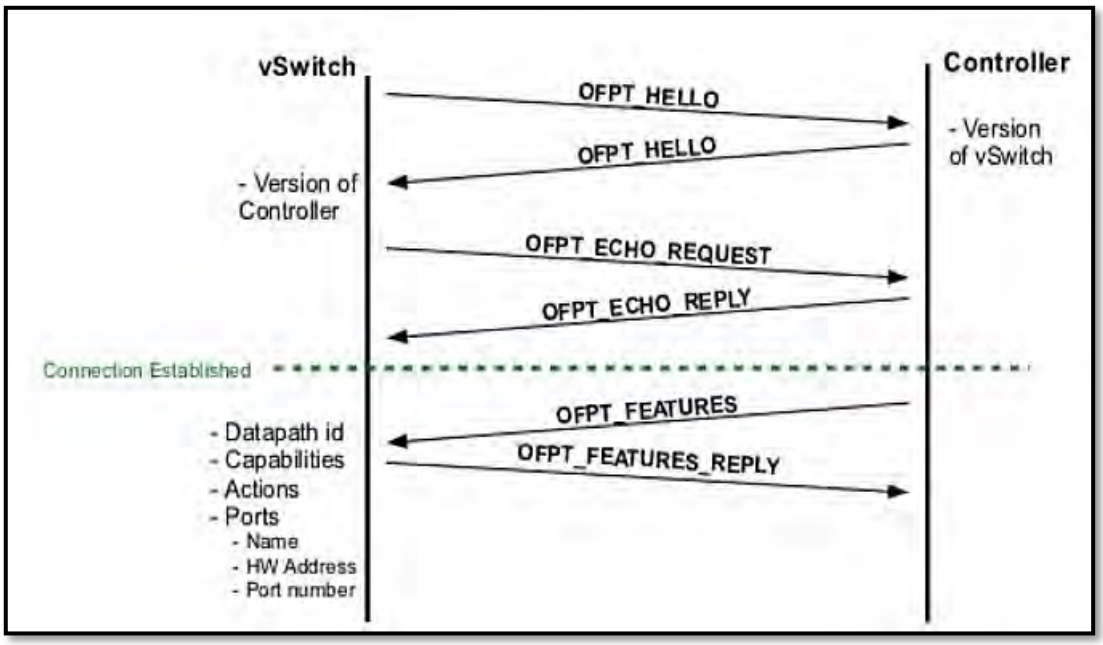


Figura 2-52 Intercambio de mensajes OpenFlow entre Switch y Controlador - Setup
Recuperado de (Curran, 2012)

2.6.3 Manejo de Mensajes OpenFlow

El protocolo *OpenFlow* tiene un concepto de entrega y procesamiento confiable de mensajes, pero sin que ello implique el uso de ACKs o acuses de recibo de forma automática o asegurar un procesamiento de mensajes en el mismo orden que salen del origen.

Entrega y Envío de Mensajes

Los mensajes tienen garantía de entrega a menos que el canal *OpenFlow* se caiga por completo, en cuyo caso el controlador no contará con ninguna información sobre el estado del *switch*, por lo que el plano de datos entraría en un estado conocido como *fail standalone mode* o modo de funcionamiento independiente por falla tal como se observa en la *Figura 2-53*.

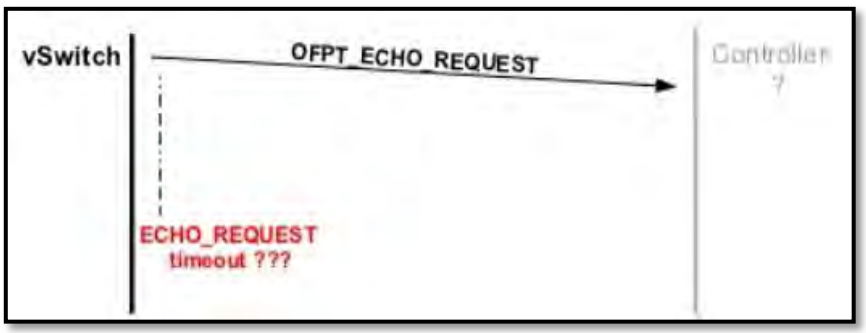


Figura 2-53 Fail Standalone Mode OpenFlow Switch
Recuperado de (Curran, 2012)

Procesamiento de Mensajes

El 100% de los mensajes enviado por el controlador al *switch* deben ser procesados, muchos de los cuales requieren un *reply*, pues en caso de que el *switch* no sea capaz de procesar algún mensaje, éste enviará un mensaje de error.

De igual manera, los *switches* deben enviar al controlador todos los mensajes asíncronos generados por cambios de estado *OpenFlow*, eliminación de un flujo, cambio de estado de puerto o recepción de mensajes *packet-in*, generando así consistencia entre el estado real del *switch* y la visión que tiene el controlador del plano de datos.

Por su parte el controlador podría ignorar ciertos mensajes que recibe, pero con la aclaración de que debe responder a mensajes *echo* para prevenir que el *switch* termine la conexión por falta de comunicación.

Agrupación y Orden de Mensajes

El controlador tiene la capacidad de agrupar mensajes mediante los mensajes opcionales *Bundle messages*, considerados como una unidad de mensajería y son procesados todos en conjunto, mientras el ordenamiento de los mensajes se lleva a cabo por los denominados *barrier messages*.

En caso de no contar con estos *barriers*, el *switch* podrá arbitrariamente ordenar los mensajes para maximizar su desempeño.

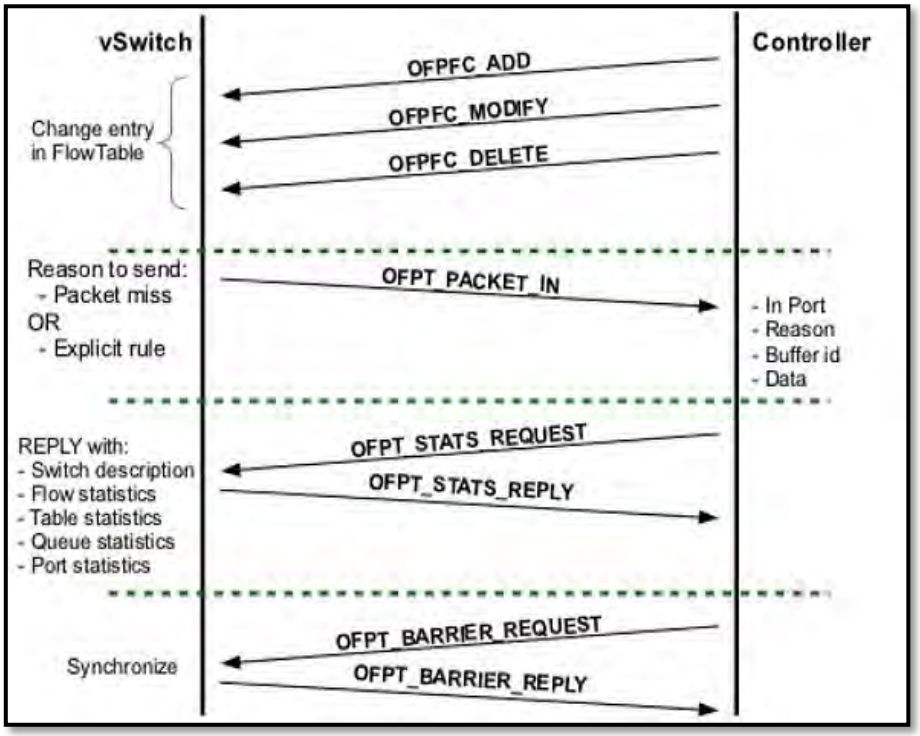


Figura 2-54 Intercambio de mensajes OpenFlow entre Switch y Controlador – Actualización de flujos Recuperado de (Curran, 2012)

2.6.4 Conexiones en el Canal *OpenFlow*

El canal *OpenFlow* es usado para intercambiar mensajes *OpenFlow* entre el *switch* y el controlador, un canal por cada *switch* que el controlador tenga a su mando, mientras el *switch* puede tener más de un canal *OpenFlow* en entornos multicontrolador (*multiplexing*).

El requerimiento necesario es que exista conectividad TCP/IP entre *switch*-controlador ya sea que estén directamente conectados o no y para su instanciamiento, se puede emplear TLS o TCP en texto plano, aunque esta última opción no se recomienda.

El dispositivo que usualmente empieza la comunicación será el *switch* por motivos de seguridad, de todas maneras, es posible que un controlador establezca la comunicación primero, pero en caso excepcionales donde hay cambios de estado en la red.

Conexión URI

Es importante que el *switch* identifique de forma única la conexión con su controlador. Esto se logra mediante una URI (Identificador único de recursos) de conexión.

Como se explicó en ***2.3 Automatización, Orquestación y Programabilidad de las Redes: DevOps y Redes Basadas en Contexto*** acerca de una URI, el formato de una URI para SDN está dado por el RFC 3986[43], el cual consta de al menos las siguientes partes:

Protocolo://nombre-o-direcciónIP:puerto

- Protocolo: En el caso de OpenSDN, define el protocolo usado como transporte de los mensajes *OpenFlow*, por ejemplo, TLS o TCP para conexiones primarias y TLS, DTLS, TCP o UDP para conexiones auxiliares, ya que estas últimas pueden no requerir transporte confiable (orientado a la conexión).
- Nombre o Dirección: Se refiere a la Dir. IP (IPv4/IPv6) del controlador o su respectivo *hostname.* En caso de que sea Dir. IPv6, esta debe ir entre corchetes según e RFC 2732.
- Puerto: Puerto de transporte empleado por el protocolo *OpenFlow*. En caso de no estar presente, se asume su puerto por defecto 6653.

Fragmentación, Control de Flujo y Seguridad

El tamaño máximo de un mensaje *OpenFlow* es de hasta 64KB, lo que supera el tamaño de 1.5KB de MTU tradicional y *OpenFlow* no posee mecanismos de fragmentación, por ello, el protocolo de transporte, de ser el caso, podría realizar el proceso de fragmentación y reensamblaje, así como el mecanismo de control de flujo en la recepción de mensajes.

Cabe recalcar que *OpenFlow* no implementa por sí mismo niveles de seguridad (confidencialidad, integridad, disponibilidad y anti-repudio), es así como el protocolo de transporte debe proveer de la seguridad necesaria.

En el 2020, se presentó una investigación titulada "***OpenSDN Southbound Traffic Characterization: Proof-of-Concept Virtualized SDN-Infrastructure****",* la cual se publicó en IEEE Xplore, explicando y demostrando el comportamiento de OpenSDN y de *Openflow* (Salazar-Chacón & Marrone, 2020)

[43] RFC 3986 – URI: https://tools.ietf.org/html/rfc3986

2.7 Controladores SDN

Durante el transcurso de la presente tesis, se ha dejado en claro que SDN es un nuevo paradigma para administrar, operar y enviar datos en una infraestructura de red, en la cual existe un controlador central encargado de brindar la inteligencia y toma de decisiones, liberando así de esa responsabilidad a los equipos, así como de la necesidad de intercambio de mensajes en el plano de datos, consiguiendo de esa manera mayor efectividad en la red.

El éxito de esta forma de envío de datos depende de que el Plano de Control esté separado y centralizado a través de un Controlador.

No obstante, SDN ya tiene algunos años de pruebas y desarrollos, por ese motivo, los controladores han ido evolucionando en el tiempo también. Actualmente se puede decir que existen dos conjuntos grandes:

- Controladores SDN para entornos NFV, DCs, WAN y Acceso modernos.
- Controladores SDN tradicionales para administrar el plano de Datos de una red.

Existe una lista de muchos controladores SDN que se adaptan a cualquiera de esos dos conjuntos, muchos incluso han servido como punto de partida de otros debido a la naturaleza mayoritaria de una comunidad *open-source*. En base al artículo de (Velrajan, 2019), la mayoría de los controladores SDN actuales funcionan bajo *OpenFlow*.

El listado según (Velrajan, 2019) es el siguiente:

1. **OpenDayLight (ODL):** Es el controlador *Open-Source* más popular. Tiene el soporte para al menos una docena de *Southbound* APIs, entre ellas NETCONF y PCEP para configurar y administrar dispositivos del plano de datos. Muchos fabricantes de infraestructura y *vendors* reconocidos a nivel mundial como *Lumina Networks, Ericsson* y *Cisco Systems* basan sus soluciones en ODL.
2. **Open Network Operating System (ONOS):** Según la ONF, es considerado un controlador modular para SDN/NFV de próxima generación, construida para dar soporte a las necesidades actuales de proveedores y empresas, generando redes dinámicas con interfaces programáticas sencillas, incluso con la nueva innovación de ONOS *Cloud Controller*. Para más información, es posible entrar a: **https://opennetworking.org/onos/**
3. **Network Operating System (NOX):** Es el controlador basado en OpenFlow original, sirviendo como fuente del desarrollo de otros Controladores escritos en C++ para Linux. Su versión *opensource* puede ser descargada de: **http://www.noxrepo.org/**
4. **Network Operating System in Python (POX):** Es un desarrollo basado en NOX pero escrito en Python, adecuado para implementarse en Windows, MacOS y por supuesto Linux. Su uso es principalmente para entornos educativos y de investigación. Puede descargase de: **https://github.com/noxrepo/pox**
5. **Beacon:** Es un controlador abierto escrito en Java de tipo modular y con soporte para varias plataformas. Ha sido usado en una gran cantidad de investigaciones en el que se ha demostrado en pruebas de concepto administrar 100 vSwitches, 20 switches físicos y corriendo por meses sin que exista caídas de servicio. Beacon puede descargarse de: **https://openflow.stanford.edu/display/Beacon/Home**

6. **Big Switch Network Controller – DANZ Monitoring Fabric:** Es una Plataforma de *networking* para SDN que provee una inteligencia unificada y alta disponibilidad para administrar y monitorear una red. *Big Switch Controller* de Arista migró a DANZ *Monitoring Fabric*, considerado un *Network Packet Broker* (NPB) basado en los conceptos de IoT para generar NetOps, DevOps y SecOps. Para más información: **https://www.arista.com/en/products/danz-monitoring-fabric**
7. **Maestro:** Es una plataforma escalable escrita en Java multihilo para Switches OpenFlow. Provee una interfaz modular para dotar de automatización y entornos programáticos para gestionar una red de datos moderna. Puede descargarse de: **http://zhengcai.github.io/maestro-platform/**
8. **RYU:** Es un sistema operativo dedicado para arquitecturas SDN. Según sus creadores, provee un control centralizado y una API bien estructurada que facilita el uso de los operadores para administrar una red SDN. Fue desarrollado por NTT Labs. Para más información de este controlador, así como de su estructura: **https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201408fa4.html**
9. **Floodlight:** Uno de los más conocidos controladores SDN desarrollados en Java por la comunidad, el cual tiene soporte para OpenFlow 1.0 hasta la versión 1.5, pero no ha tenido mayor contribución desde el 2017, principalmente por ser el corazón del controlador Big Switch. Para más información del proyecto Floodight, puede acceder a **https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview?homepageId=1343545**
10. **Faucet:** Es un controlador SDN para entornos OpenFlow 1.3 de tipo compacto que permite a los administradores de una red operarla tal como un *clúster* de servidores, moviendo funciones de red como enrutamiento, descubrimiento de vecinos y algoritmos de *switching* a un *software* independiente y abierto con el fin de administrar de mejor manera la infraestructura. Para más información de este controlador: **https://faucet.nz/**
11. **Virtual Application Networks (VAN):** Es un controlador de lo más exitoso en los inicios de SDN, tanto bajo HP como bajo Aruba (parte de HP Enterprise), considerado por muchos como el corazón de infraestructuras SDN, el cual ofrece inteligencia, automatización y monitoreo total de la red. Presenta total soporte para el protocolo OpenFlow. Actualmente no tiene soporte de HPE, pero se puede encontrar documentación relevante sobre su instalación y características en: **https://support.hpe.com/hpesc/public/docDisplay?docId=a00003658en_us&docLocale=en_US**
12. **NEC *ProgrammableFlow Controller*:** Es un controlador desarrollado bajo OCP (*Open Compute Project*) que trata de dar un entorno similar al *networking* tradicional. En teoría, es un controlador con la capacidad de soportar hasta 10000 *switches*. Es posible encontrar información en: **https://www.necam.com/SDN/**
13. **Ericsson Cloud SDN Controller:** Diseñado para una comunicación inter-intra DCs para grandes flujos de tráfico de equipos físicos como virtuales. Combina el uso de OpenDayLight con capacidades de enrutamiento avanzadas y enfoque hacia NFV. Para más información: **https://www.ericsson.com/en/portfolio/digital-services/cloud-infrastructure/cloud-sdn**

Sin importar el tipo de controlador SDN, sea este abierto o fabricado por algún *vendor*, la idea general de este equipo es aquella en la que:

- Administra el estado de la red en conjunto con la estructura de datos (tablas y/o bases de datos) de un equipo. Esta estructura de datos sirve como un repositorio de información de los equipos controlados.
- Provee de un modelo de alto nivel que captura las imágenes de la topología y sus elementos. En la actualidad, se usa mucho el modelo YANG.
- Otorga una moderna API, normalmente tipo RESTful, exponiendo así los servicios que da el controlador a una aplicación amigable con el administrador de la red. Idealmente, esta API es producto del Modelo YANG de la topología.
- Debe generarse una sesión de comunicación segura entre Controlador y Agentes.

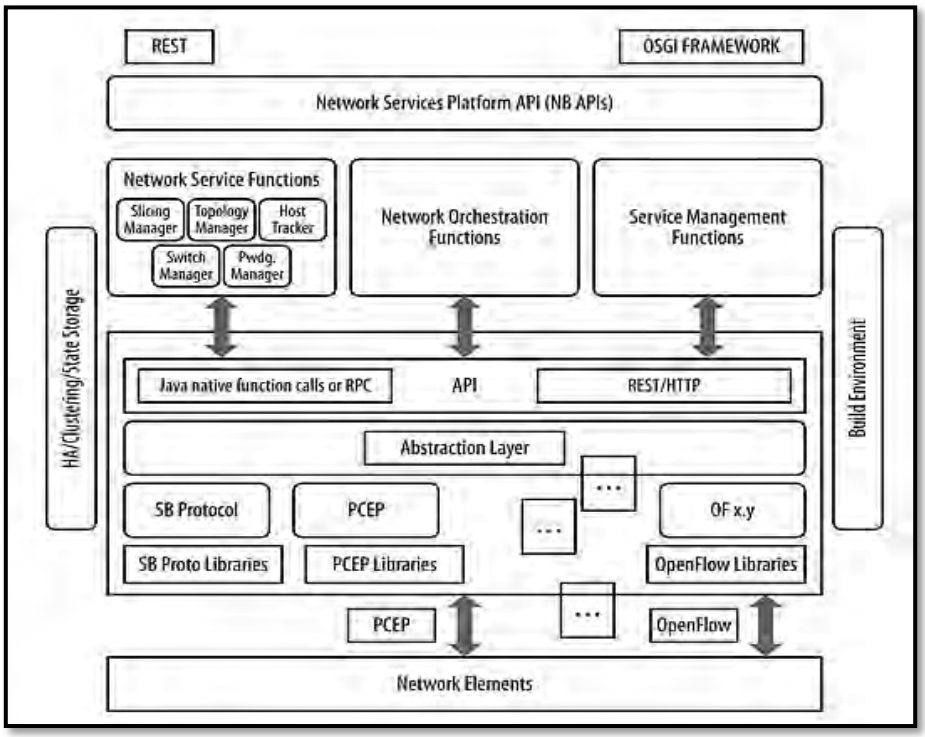


Figura 2-55 Esquematización de un Controlador SDN
Recuperado de (Nadeau & Gray, 2013)

Basado en (Salman, Elhajj, Kayssi, & Chehab, 2016), se desarrolló el cuadro comparativo entre los distintos Controladores SDN disponibles, el cual se puede apreciar en la ***Tabla 2-10*** Características de Controladores SDN.

Según la investigación planteada por (Salman, Elhajj, Kayssi, & Chehab, 2016), en base a *testbeds* realizados en Cbench[44], se determinaron las siguientes tendencias en cuanto a *Throughput* y latencia (Ver ***Fig. 2-56***).

[44] Cbench: https://pypi.org/project/cbench/

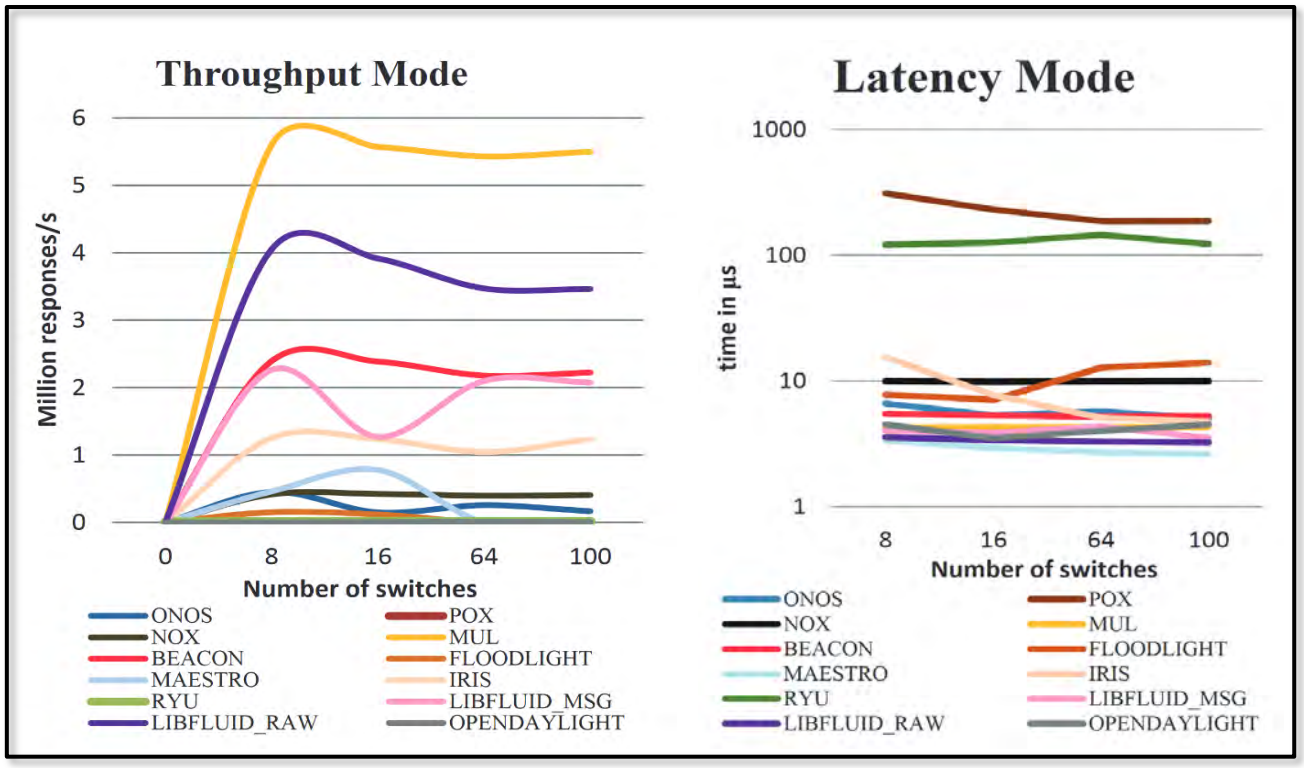


Figura 2-56 Comparación de Throughput y Latencia - Controladores SDN
Recuperado de (Salman, Elhajj, Kayssi, & Chehab, 2016)

Dicho estudio varió la cantidad de *switches*, así como la cantidad de peticiones hacia el controlador.

Según (Salman, Elhajj, Kayssi, & Chehab, 2016), los controladores SDN codificados en lenguaje C tienen mejor desempeño en general (Mul y LibFluid) y bajo ese desempeño los desarrollados en Java (Beacon, Iris y Maestro), sin embargo, en cuanto a latencia, Maestro tiene los mejores registros, principalmente debido a su modo de procesamiento adaptativo. Por su lado, aquellos basados en Python, no mostraron una mejora significativa.

No está por demás mencionar que la latencia y *Throughput* no deben ser los únicos parámetros para tomar en cuenta, pues la modularidad presente en ONOS y OpenDayLight hacen que sean los Controladores SDN preferidos tanto en ambientes PoC como comerciales.

La inclusión de nuevos protocolos *Southbound* del mundo IoT también es un factor para considerar, es ahí donde ONOS y ODL sobresalen.

Tabla 2-10 Características de Controladores SDN

Controlador SDN	Desarrollado en (Lenguaje Programación)	Interfaz con el Administrador (FrontEnd)	Info disponible	Modularidad	Soporte OS	Southbound APIs	Northbound APIs	Partner empresarial	Soporte OpenStack	Ideal para
ONOS	Java	HTTP-Web	Buena	Alta	Linux, MAC y Windows	OF 1.0, 1.3; NETCONF	RESTful	AT&T, Ciena, Cisco, Ericsson, Huawei, Intel, NEC, Sk Telecom, Fujitsu, ON.Lab	No	DC, WAN e ISP
ODL (OpenDayLight)	Java	HTTP-Web	Muy buena	Alta	Linux, MAC y Windows	OF 1.0, 1.3, 1.4; NETCONF, YANG, OVSDB, PCEP, BGP-LS, LISP, SNMP	RESTful	Linux Foundation con membresias para empresas como Cisco, IBM, NEC, etc.	Sí	DC, WAN, Campus e ISP
NOX	C++	Python	Poca	Baja	Linux	OF 1.0	RESTful	Nicira	No	Campus
POX	Python	Python	Poca	Baja	Linux, MAC y Windows	OF 1.0	RESTful	Nicira	No	Campus
RYU	Python	HTTP-Web	Adecuada	Adecuada	Linux	OF 1.0, 1.2, 1.3, 1.4, NETCONF, OF-CONFIG	RESTful	Nippo Telegraph, Telephone Corp.	Sí	Campus

Beacon	Java	HTTP-Web	Adecuada	Adecuada	Linux, MAC y Windows	OF 1.0	RESTful	Universidad de Standford	No	Investigación y PoCs
Maestro	Java	No	Poca	Adecuada	Linux, MAC y Windows	OF 1.0	RESTful	RICE, NSF	NO	Investigación y PoCs
Flood-Light	Java	HTTP-Web (Java-Based)	Buena	Adecuada	Linux, MAC y Windows	OF 1.0, 1.3	RESTful	Big Switch Networks	No	Campus
Iris	Java	HTTP-Web	Adecuada	Adecuada	Linux, MAC y Windows	OF 1.0, 1.3; OVSDB	RESTful	ETRI	NO	ISP
MUL	C	HTTP-Web	Adecuada	Adecuada	Linux	OF 1.4, 1,3. OVSDB, OF-CONFIG	RESTful	Kulcloud	Sí	DC
Lib-FLUID	C++	No	Adecuada	Adecuada	Linux	OF 1.0, 1.3	-	ONF	NO	-

Fuente: (Salman, Elhajj, Kayssi, & Chehab, 2016)

2.8 SDN como plataforma para la revolución de las tecnologías emergentes: IoT, Cloud y Digitalización

Los momentos actuales requieren de catalizar la transformación de la industria del *networking* pues es el soporte de cualquier negocio y actividad del siglo XXI, todo ello ahondado con requerimientos de empresas y de todos los sectores estratégicos como salud, educación, administración de recursos públicos en época de pandemia COVID-19, es así que el consorcio sin fines de lucro denominado *Open Networking Foundation* (ONF) ha desarrollado alianzas estratégicas para fomentar la evolución de la infraestructura de datos en entornos abiertos y colaborativos con miras al futuro, cubriendo campos investigativos, evangelización de SDN y educativos.

Una ola de innovación marcada por la demanda creciente de *White boxes* en lugar de productos propietarios y cerrados y de plataformas abiertas está surgiendo. Según (ONF - Open Networking Foundation, 2020) este ecosistema del "nuevo *networking*" representa una oportunidad única para operadores, así como fabricantes de tecnología, reduciendo los costos de inversión en un 25% al 40% y en la cadena de suministro, representa una oportunidad de mercado SDN de $73 Billones de dólares para el 2025.

Entre los proyectos más significativos de la ONF están:

- Redes Móviles
  Enterprise Edge 5G/4G EdgeCloud-as-a-Service
  O-RAN *Compliant SD-RAN*
  OMEC (*Open Mobile Evolved Packet Core – EPC*)
- *Cloud* y Servicios en la Nube (IoT)
  CORD y XOS (*Edge Cloud Infrastructure Platform*)
- Infraestructura SDN
  Trellis (un *fabric* distribuido para SDN/NFV en arquitecturas *spine-leaf*)
  ODTN (Transporte Óptico)
  NG-SDN (*Next-Generation SDN*) con soporte para lenguaje P4.

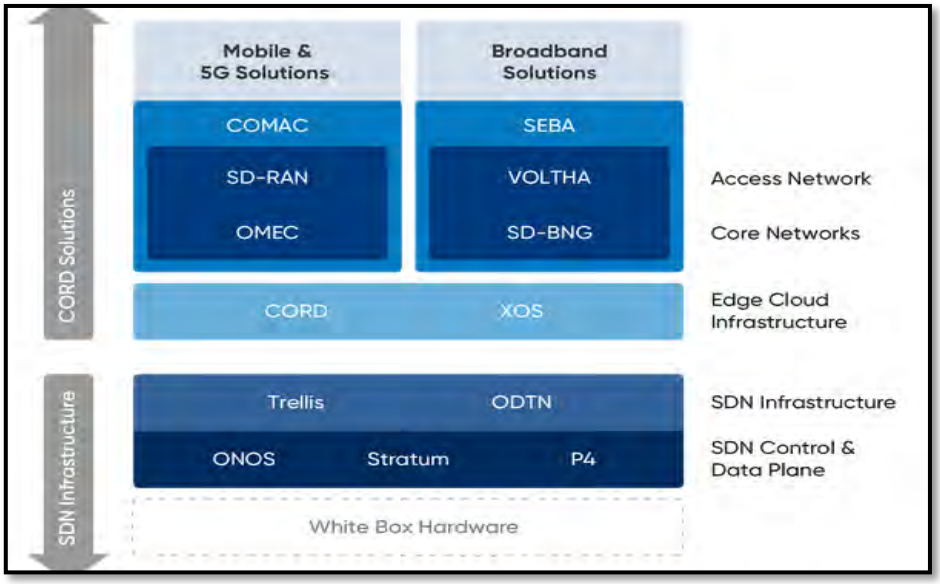


Figura 2-57 Proyectos SDN para la ONF: NG-SDN
Recuperado de (Open Networking Foundation - ONF, 2020)

Junto con los proyectos antes mencionados, la ONF trabaja para mejorar el emulador de redes basados en controlador denominado Mininet.

Según la ONF, Mininet provee un ambiente para *testbeds* virtuales sobre SDN, permitiendo tener emulaciones de grandes prestaciones en cualquier laptop, servidor, PC o en la nube, dando lugar a una migración rápida del entorno de prueba hacia el mundo real.

Mininet corre código real, así como aplicaciones tipo UNIX/LINUX, aplicativos con Kernel Linux y cualquier protocolo del stack de *networking.*

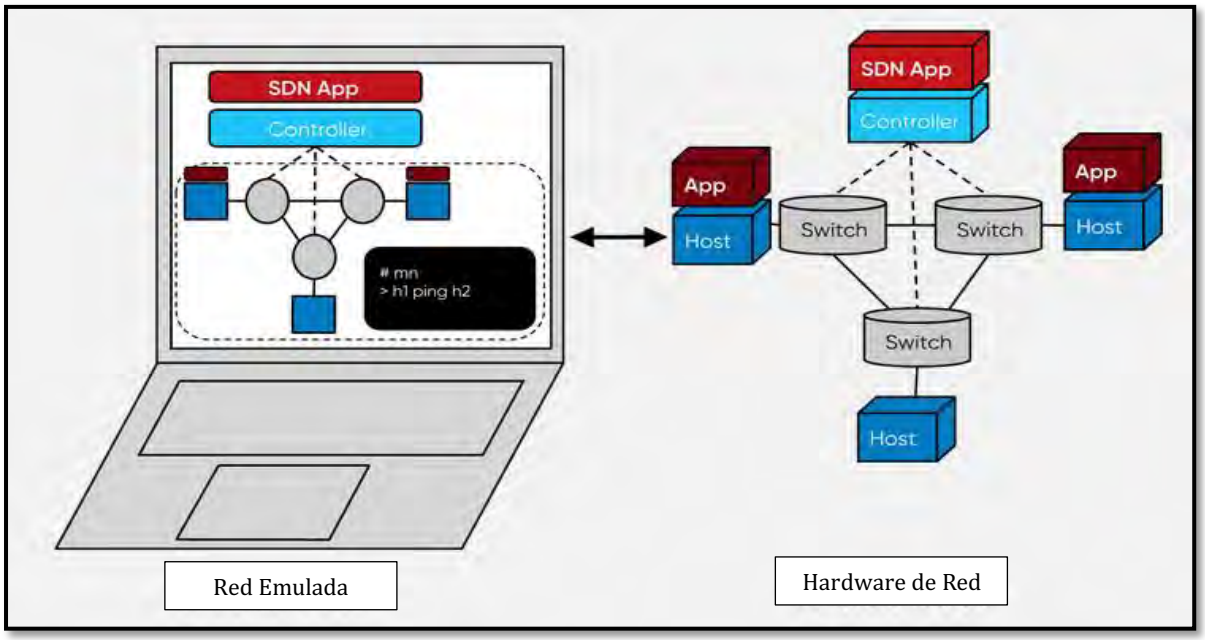


Figura 2-58 Mininet: Componentes
Recuperado de (ONF - Open Networking Foundation, 2021)

Mininet consiste de:

- Hosts Aislados
  Grupo de procesos a nivel de usuario enviados al entorno de emulación de red que proveen interfaces, puertos y tablas de enrutamiento.
- Enlaces Emulados
  Es factible emular enlaces lo más apegado a la realidad debido a *Linux Traffic Control* (tc). Cada host cuenta con su propia interfaz emulada Ethernet.
- Switch Emulados
  Tanto el Linux Bridge como un OVS (Open vSwitch) corriendo en modo Kernel puede implementarse en Mininet

En el ***Anexo B: Guía de Instalación GNS3-VM, EVE-ng y Mininet*** se explica el proceso de instalación de Mininet.

2.8.1 Fundamentos de NG-SDN: SDN de próxima Generación

NG-SDN es uno de los proyectos más relevantes de la ONF, plataforma que integra los mayores avances en cuanto a SDN se refiere, por ejemplo, *Open Hardware*, infraestructuras programables, implementaciones tipo *zero-touch* con capacidad de funcionar en la nube para dar mayor flexibilidad, control y efectividad a la red.

Los elementos abiertos que componen esta solución son:

- uONOS: Plataforma de configuración y control centralizado para administrar *switches* Stratum.
- Stratum: Sistema Operativo para *switches* SDN basado en modelos de configuración abiertos, interfaces SDN y programación denominada P4.
- Motor de Verificación NG-SDN: Mecanismo para dota de visibilidad y control general y granular de la red (chequeo de cada paquete/flujo de datos), otorgando *insights* a la red.
- *Framework* para Operacionalización de NG-SDN: Herramienta para incluir el entorno CI/CD y DevOps a la red, dotando de orquestación, programabilidad, verificación y cambios automáticos.

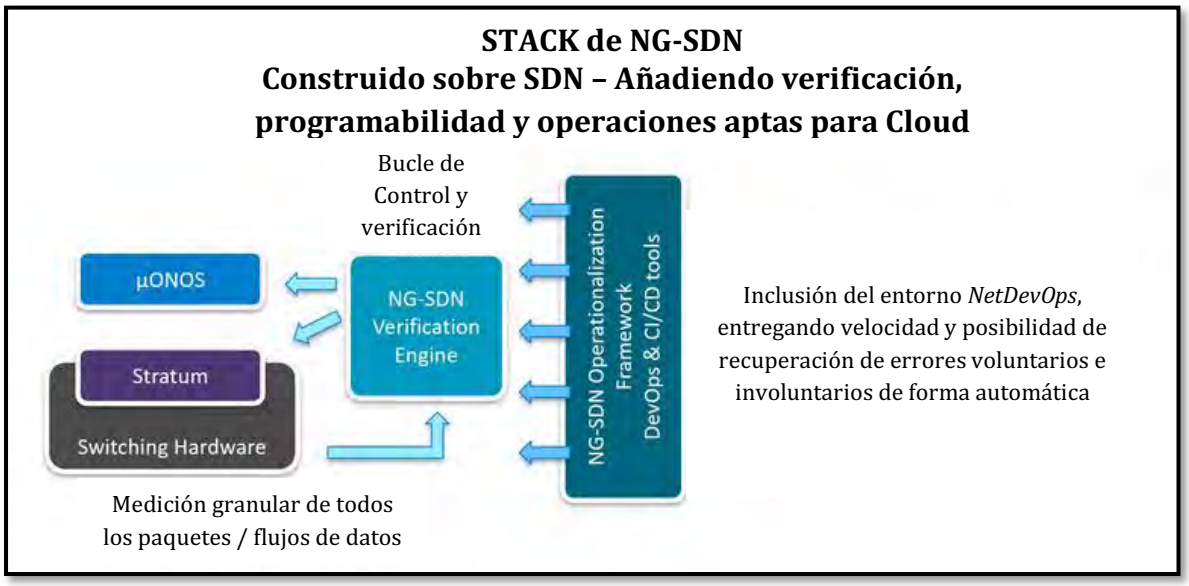


Figura 2-59 NG-SDN Stack
Recuperado de (ONF - Open Networking Foundation, 2021)

2.8.2 Programación en NG-SDN: P4 y *Stratum*

P4, cuyo nombre proviene de *Programming Protocol-Independent Packet Processors,* es un lenguaje que permite expresar cómo los paquetes se procesan por el plano de datos en un equipo de *networking* físico o virtual (*switch, router* o cualquier dispositivo de red). Como se ha visto en capítulos pasados, el *networking* tradicional posee ASICs muy específicas y propietarias que no permiten modificar el comportamiento del plano de datos, es así que en el *Open Netwoking*, con el uso de cajas blancas u *Open Hardware* con NOS abiertos se dio un avance significativo en desarrollos e investigaciones, dando mayor adaptabilidad y programación específica.

En el ecosistema SDN, es necesario un lenguaje que diera la facilidad de programar únicamente la funcionalidad del plano de datos, ya que el controlador y en general el plano de control, son gestionados mediante *Northbound* APIs.

En otras palabras, programas como en P4, describen el comportamiento de envío, programa que luego es convertido por un compilador en meta data que utiliza el plano de control y el plano de datos para la transmisión de datos. Esto se conoce como *"Top-Down Approach"*, en contra posición del tradicional y cerrado *"Bottom-up"*.

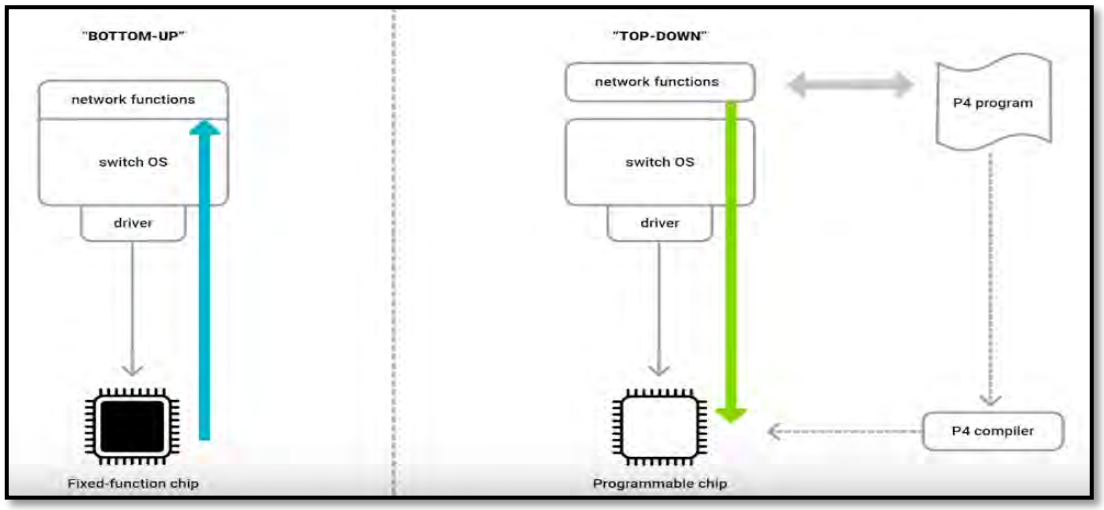


Figura 2-60 Aproximación Top-Down en el ecosistema SDN con Open Hardware
Recuperado de (CodiLime, 2020)

Una de las ventajas de P4 es su apertura a los aportes de la comunidad, pero gestionada bajo un modelo de gobernanza adecuado con el fin de que los mejores aportes sean integrados a la solución de programación. ONF apoya este proyecto mediante vínculos de membresía y colaboración.

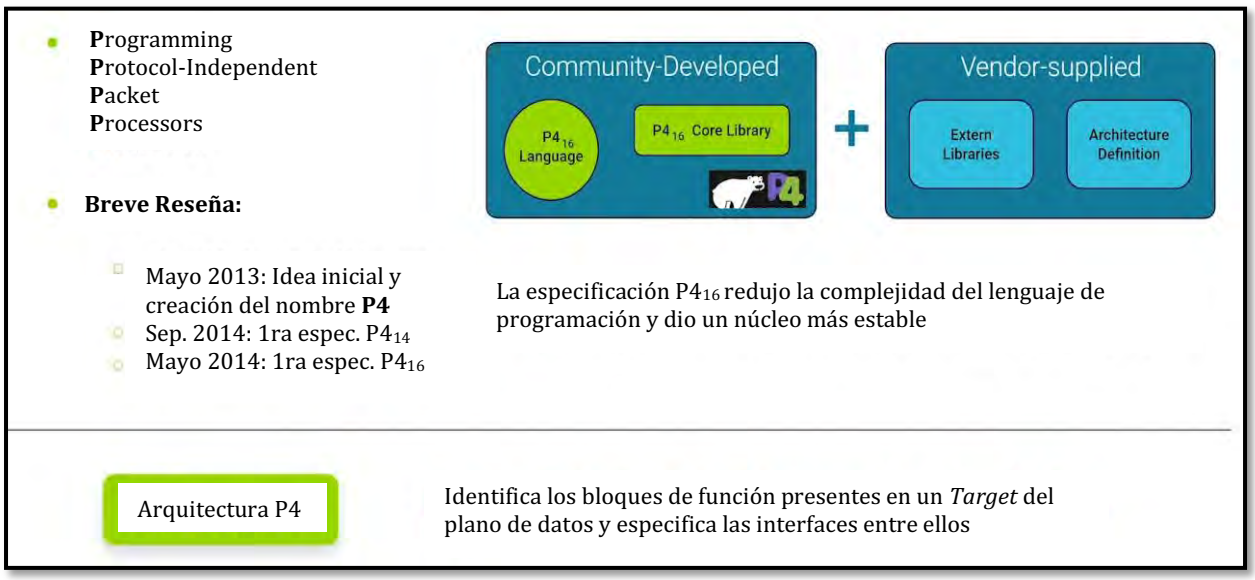


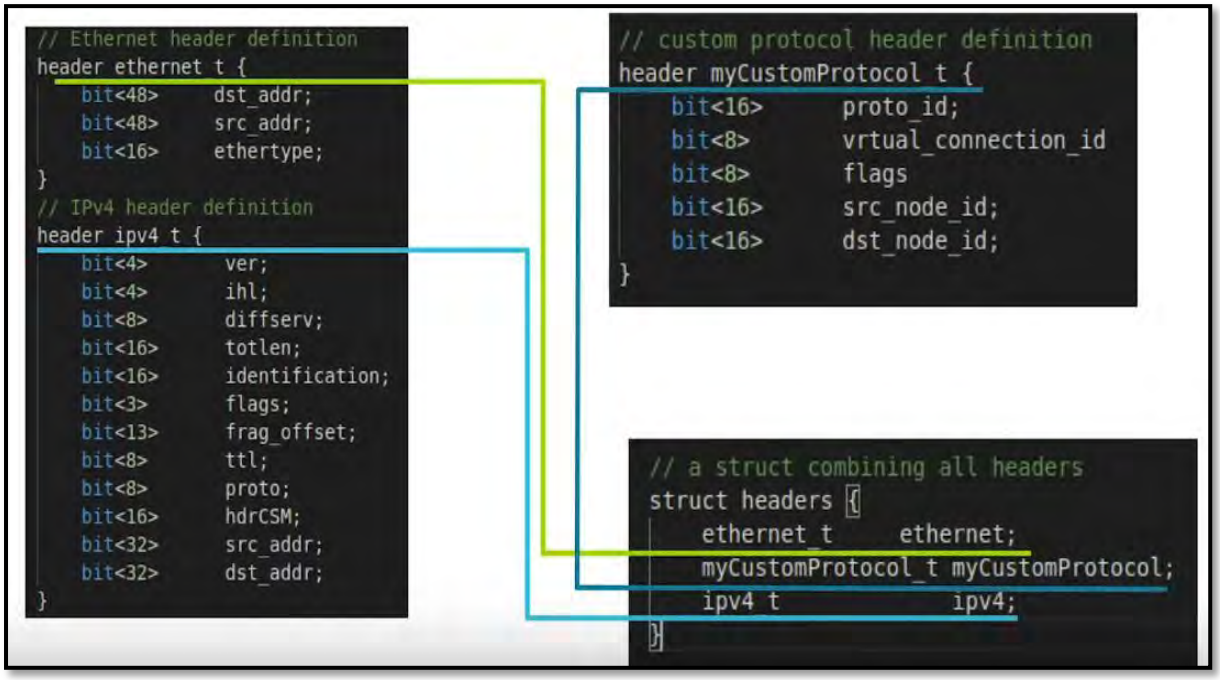
Figura 2-61 Lenguaje de Programación P4 – Bloques de función en Targets
Recuperado de (CodiLime, 2020)

Para P4, los *Targets* u objetivos son los dispositivos del plano de datos, como *switches* o SmartNICs programables. los cuales recibirán algún dato encapsulado. Ese dato, debe ser identificado y representado como Bloque de función programable en base a sus características (cabeceras del proceso de encapsulación).

Los bloques de función programables son de dos tipos:

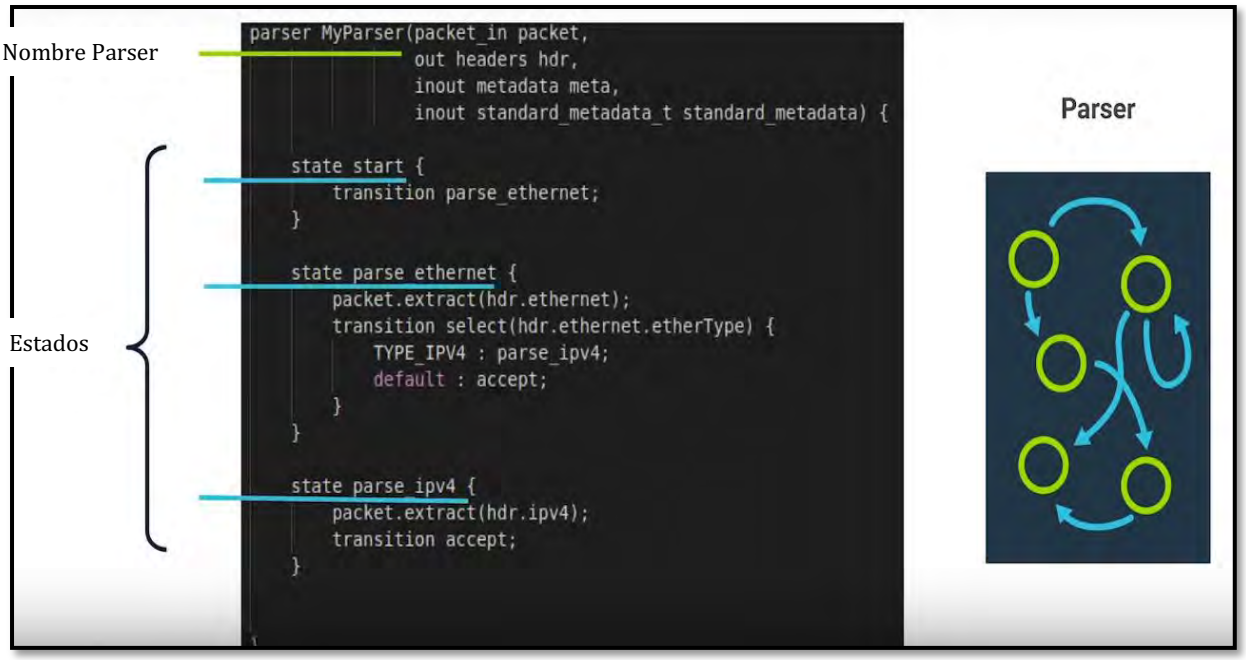
- Parser - Analizador:
  Encargado de identificar las cabeceras presentes en cada paquete.
  En el programa P4 se debe definir la estructura y comportamiento del flujo de datos.
  La representación de las cabeceras más importantes pasa al primer bloque de Control
- Control:
  Realiza las acciones de envío en base a tablas (envío, bloqueo, eliminación), control de coincidencias (*match-action behavior*), verificación de errores.

Como ejemplo de Bloques tipo *parser* se tiene esta definición de cabeceras para trama Ethernet, Paquete IPv4 y para un protocolo personalizado, así como una estructura que combina todos esos *parsers*.



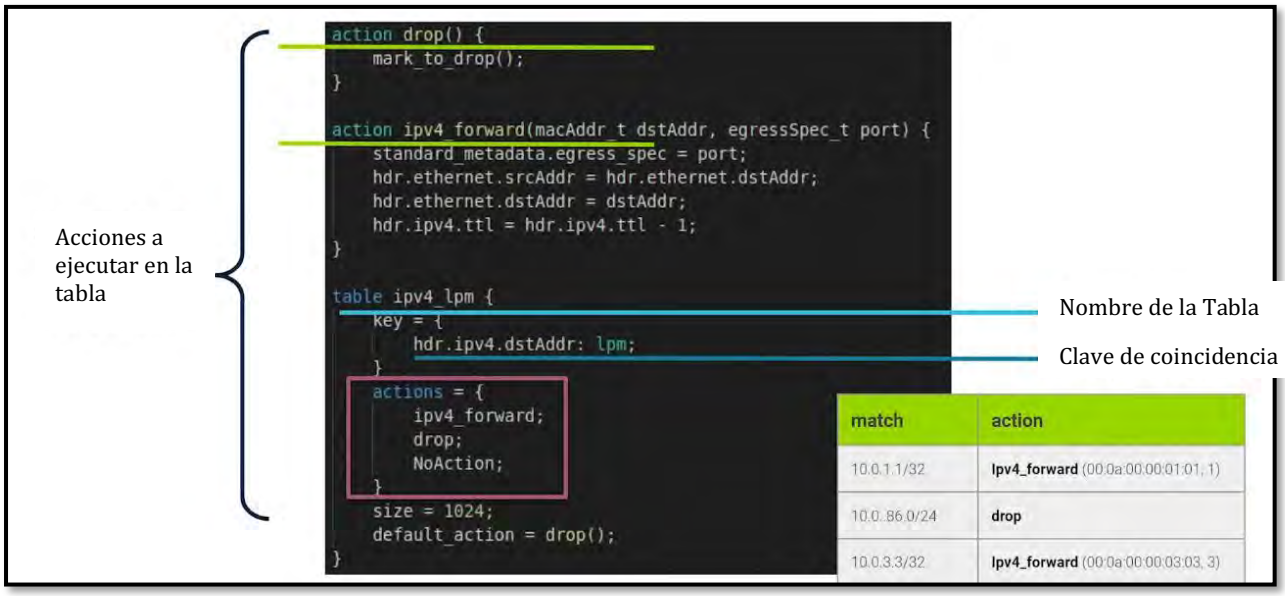
Script 17 Bloques de función Parser en P4: Cabecera Ethernet, IPv4, Protocolo personalizado Recuperado de (CodiLime, 2020)

Para implementar un Bloque Parser, se le debe dar un nombre y establecer estados iniciales en los datos tal como se visualiza en el siguiente *script*.



Script 18 Implementación de Bloques de función Parser en P4
Recuperado de (CodiLime, 2020)

El funcionamiento de P4 para programar el plano de datos se basa en comparaciones con tablas, con las cuales realiza un proceso de *match-action*, por ejemplo, si se desea enviar tráfico proveniente de una determinada IPv4 y con una determinada dirección MAC:



Script 19 Proceso Match-Action en P4
Recuperado de (CodiLime, 2020)

Cabe decir que cada bloque de control debe tener un sub-bloque tipo *apply*, el cual especifica cómo los paquetes van a ser procesados.

La compilación y ejecución de P4 en un equipo de red se puede resumir en este esquema:

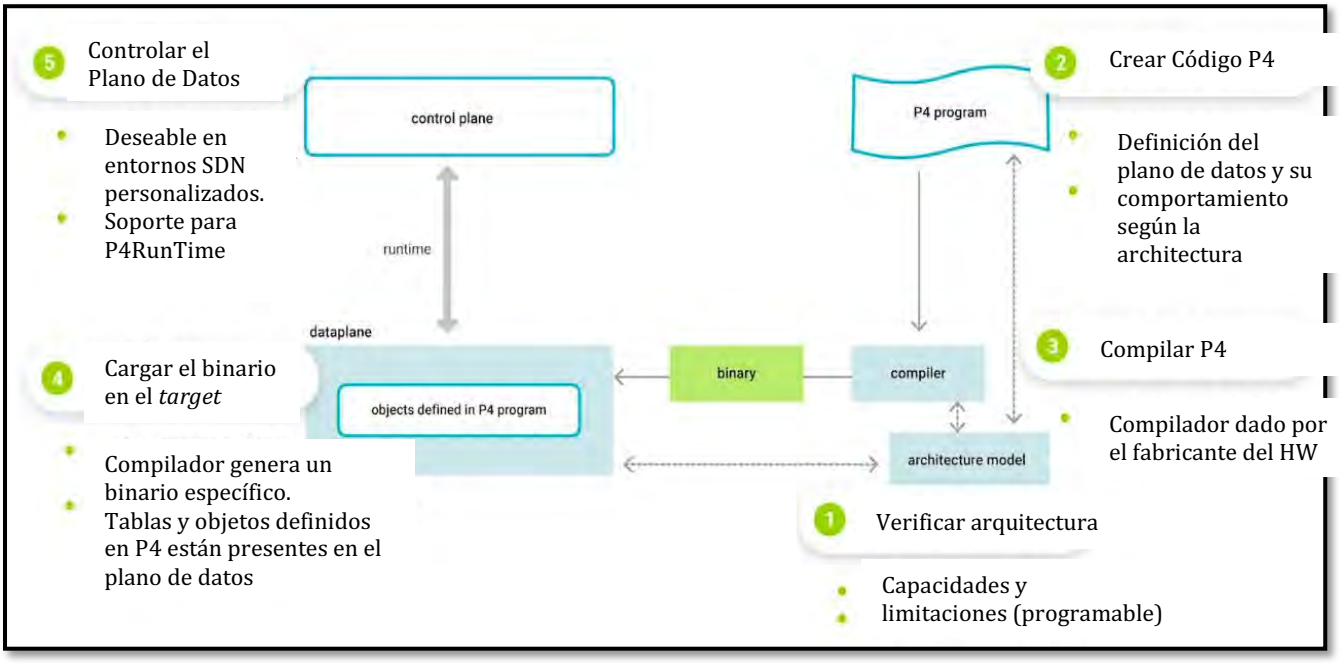


Figura 2-62 Esquema de Funcionamiento de P4 en PISA (Protocol-Independent Switch Architecture)
Recuperado de (CodiLime, 2020)

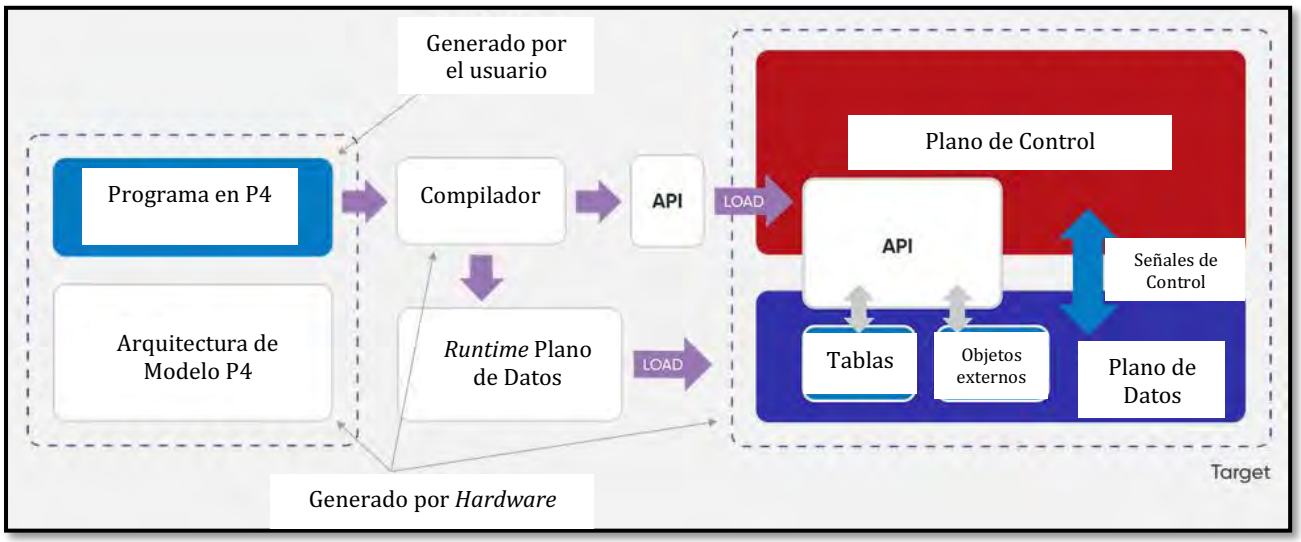


Figura 2-63 Arquitectura de P4
Recuperado de (ONF - Open Networking Foundation, 2021)

Para más información sobre P4, la ONF ha publicado los siguientes sitios y repositorios:

P4 Wiki: **https://wiki.opennetworking.org/display/COM/P4**

Página Web de P4: **https://p4.org/**

GitHub de P4: **https://github.com/p4lang/**

Por su parte, el *hardware* juega un rol muy importante dentro de este entorno de NG-SDN, en este caso, ***Stratum*** es el NOS totalmente independiente del equipo subyacente empleado (cajas blancas) para redes SDN con el apoyo de la ONF, organismo que lo considera un componente clave de las soluciones SDN del futuro.

Stratum implementa las interfaces *Northbound* más novedosas en el mundo SDN, incluyendo a P4RunTime (API para la programación P4 del plano de datos), gNMI[45]/OpenConfig y gNOI[46].

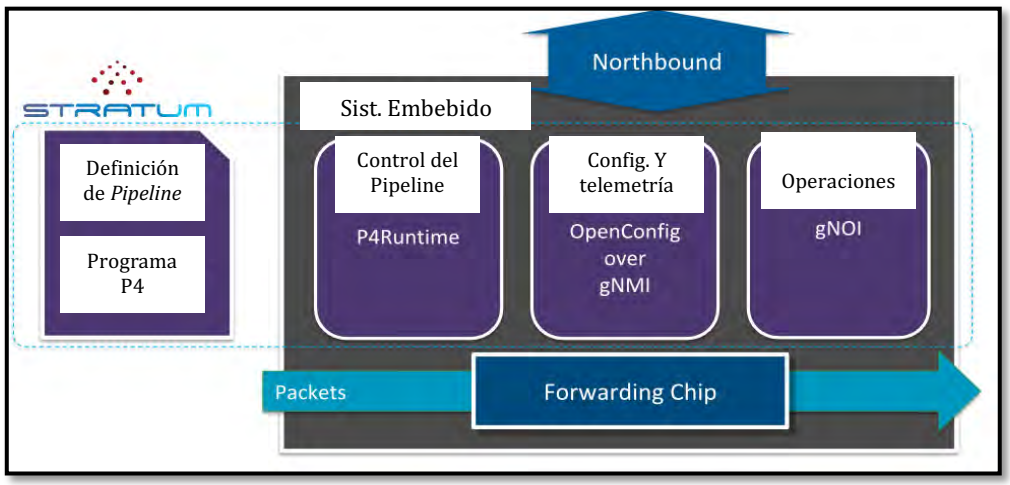


Figura 2-64 Esquematización de Stratum
Recuperado de (O'Connor, 2018)

La esquematización de NG-SDN para la ONF sería la siguiente:

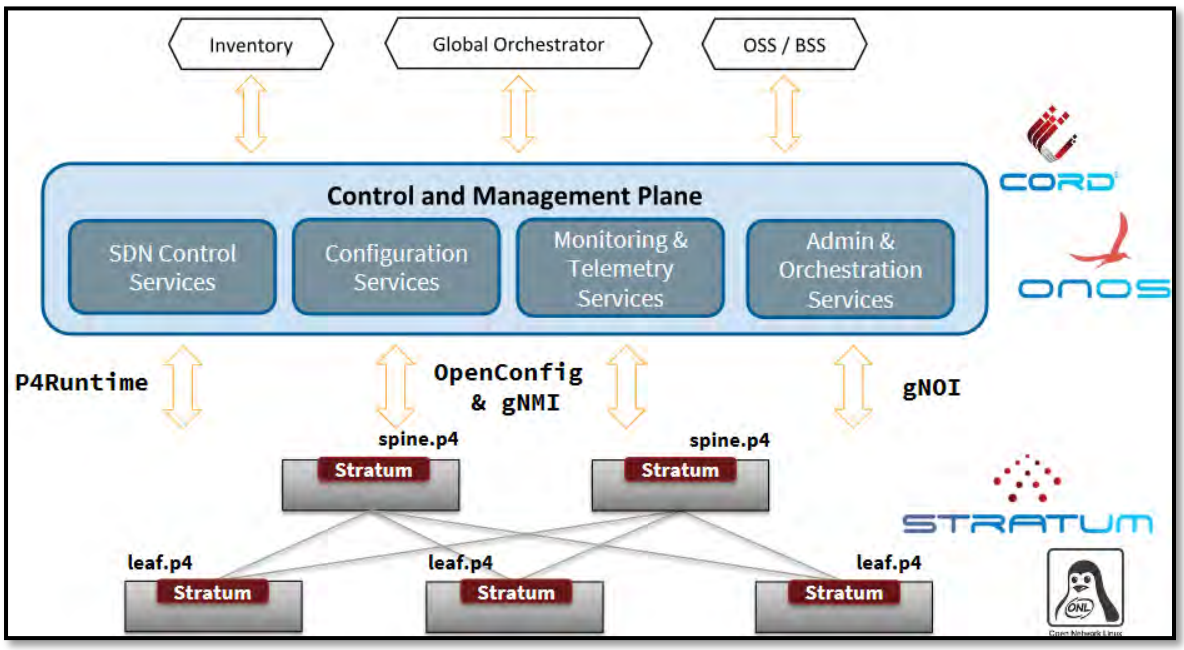


Figura 2-65 Esquematización de NG-SDN de ONF
Recuperado de (O'Connor, 2018)

[45] gNMI: gRPC Network Management Interface
[46] gNOI: gRPC Network Operation Interface

Capítulo 3

3 Redes Híbridas: Infraestructura Tradicional y SDN

La evolución de las redes de datos, como se ha analizado a lo largo del **Capítulo 2**, ***Fundamentos de las Redes Definidas por Software*****,** es un proceso que establece el cómo las redes modernas deben diseñarse, administrarse y cómo deben operar en un mundo de Transformación Digital y es ahí el verdadero reto de los profesionales en el campo de las Tecnologías de la Información, encontrar cómo adaptar las redes tradicionales hacia el ecosistema SDN mientras estamos en esta transición.

Las tecnologías del tipo *OpenNetworking,* NG-SDN, *NetDevOps* e *Intent-Based Networking* están ya rompiendo esquemas en el campo de las Telecomunicaciones y en lo que respecta a las redes de datos, la Redes tipo Campus o empresariales jugarán un rol determinante para dar soporte a los nuevos requerimientos de clientes.

Con el paso del tiempo, protocolos tradicionales como STP (*Spanning-Tree Protocol*) han ido cambiados por soluciones como VSS (*Virtual Switching System*) o vPC (*Virtual Port-Channel*) con el fin usar al máximo los recursos disponibles en una infraestructura. Eso mismo está ocurriendo con el resto de los protocolos, incluso modificando los esquemas de diseño de redes, pasando de un tradicional Modelo de 3-capas a un diseño *Spine-Leaf*.

Es innegable pensar que las redes debían evolucionar, pues con solo pensar en el incremento de dispositivos conectados a Internet y los requerimientos de usuarios, empresas y gobiernos durante la pandemia de COVID-19, hacen que sea más que una evolución, una revolución en las redes de datos.

Según (Nefkens, 2020), para el final del 2021, América Latina tendrá un promedio de cerca de 3 dispositivos electrónicos per-cápita conectados a Internet y Norteamérica cerca de 13, lo cual es realmente un reflejo de la inclusión de IoT en nuestras vidas, más allá de solamente el campo industrial o de sensores.

Tabla 3-1 Número de Dispositivos per-cápita

Región	2016	2021	Crecimiento Anual
Asia Pacífico	1.9	2.9	8.3%
Europa	5.3	8.9	10.9%
América Latina	2.1	2.9	7%
Norteamérica	7.7	12.9	11%
África	1.1	1.4	5.4%
Promedio Mundial	2.3	3.5	8.5%

Fuente: (Nefkens, 2020)

Debido a este aumento de dispositivos y elementos conectados, la complejidad es otro factor por considerar en las redes modernas.

Unos años atrás, las redes de tipo campus o empresarial eran simples, contaban con VLANs, asignación de puertos y enrutamiento entre las sedes, era una configuración básicamente estática, sin embargo, en los momentos en que nos encontramos, el dinamismo es parte de las redes empresariales, así como la inclusión de elementos necesarios en una empresa, por ejemplo VoIP, colaboración, uso de la nube, es decir, la inclusión de políticas tanto de seguridad como de Calidad de Servicio para dar soporte a las tecnologías mencionadas.

De ello, se puede decir que las redes modernas, requieren de entornos que sean fácilmente administrables, con capacidad de visibilidad total y que tengan funciones proactivas para encajar plenamente en un mundo de transformación digital, donde el talento humano es el centro.

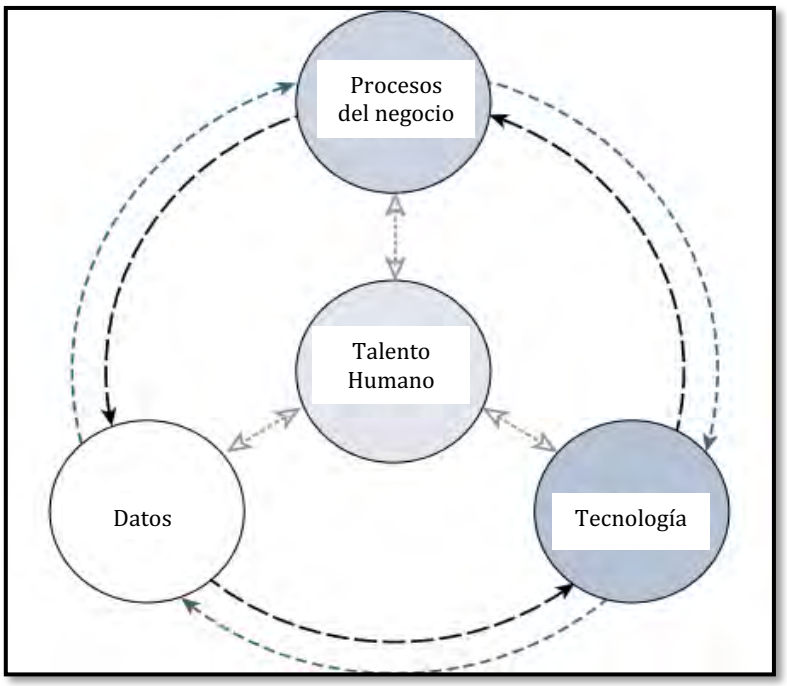


Figura 3-1 Partes de una Infraestructura empresarial moderna – Transformación Digital Basado en (Nefkens, 2020)

La mayoría de las redes creadas y diseñadas en el año 2021, año de publicación de esta tesis, aún requieren de configuraciones equipo por equipo, lo que implica tiempo de configuración manual, lamentablemente, eso ya no es deseable, la inmediatez es un requisito hoy en día, las tecnologías evolucionan, los datos se incrementan para soportar los procesos del negocio, lo que implica una mayor capacitación del talento humano.

Pero estos cambios no solo han afectado a las redes tipo Campus, también a las WAN y a las redes de acceso, es así como, en el presente capítulo, se conceptualiza y se demuestra la evolución de las tecnologías que permitan una migración adecuada hacia el mundo de la programabilidad, automatización y SDN, incluso en etapa en que las redes tradicionales deben convivir con las redes modernas.

3.1 Segment-Routing (SR) y Grupo de trabajo SPRING en Redes Tradicionales y SDN, una alternativa a MPLS

Según (Salazar Ch., Naranjo, & Marrone, 2018), las plataformas de red deben ofrecer una infraestructura de TI adaptable que permita dar soporte a las necesidades del negocio, por tal motivo, una estrategia de modernización es necesaria, tomando en cuenta al cambio de paradigma hacia SDN.

Es un hecho que las Telecomunicaciones están evolucionando con miras a entornos definidos por software, y *Segment-Routing* (SR), ha emergido como una solución práctica y eficiente del lado de la WAN, otorgando control flexible, con posibilidad de manejo de flujos de tráfico en una red de proveedor de servicio, razones por las que muchos SPs están ideando planes de migración.

Figura 3-2 Logo de Segment Routing
Recuperado de (Segment Routing, 2021)

Es importante mencionar que la idea básica empleada en SR no es para nada un concepto nuevo, es más, *source-routing* es una manera de enviar datos en una red IP idealizada hace un par de décadas atrás, pero su implementación en entornos de tráfico masivo es relativamente nueva. La idea fue concebida desde la publicación del RFC 791 que habla sobre IPv4 a inicio de los 80s.

Source-Routing o envío desde la fuente, es una técnica de envío de paquetes en la que el *router* emisor o *router* de primer salto selecciona la mejor ruta de una manera predefinida contenida en el propio paquete en lugar de emplear protocolos de enrutamiento basados en el destino.

La ruta seleccionada mediante *source-routing* es un camino arbitrario y puede diferir completamente de la mejor ruta seleccionada mediante enrutamiento tradicional enfocado en el destino, pero ese comportamiento no es en sí algo malo, pues puede servir para mecanismos de solución de problemas o para encontrar diversas rutas alternas al camino principal.

Se puede tomar como ejemplo el planteado en la siguiente figura, donde se tiene el mejor camino establecido por el intercambio de mensajes mediante OSPF (en verde) y el camino seleccionado arbitrariamente por el *router* de origen.

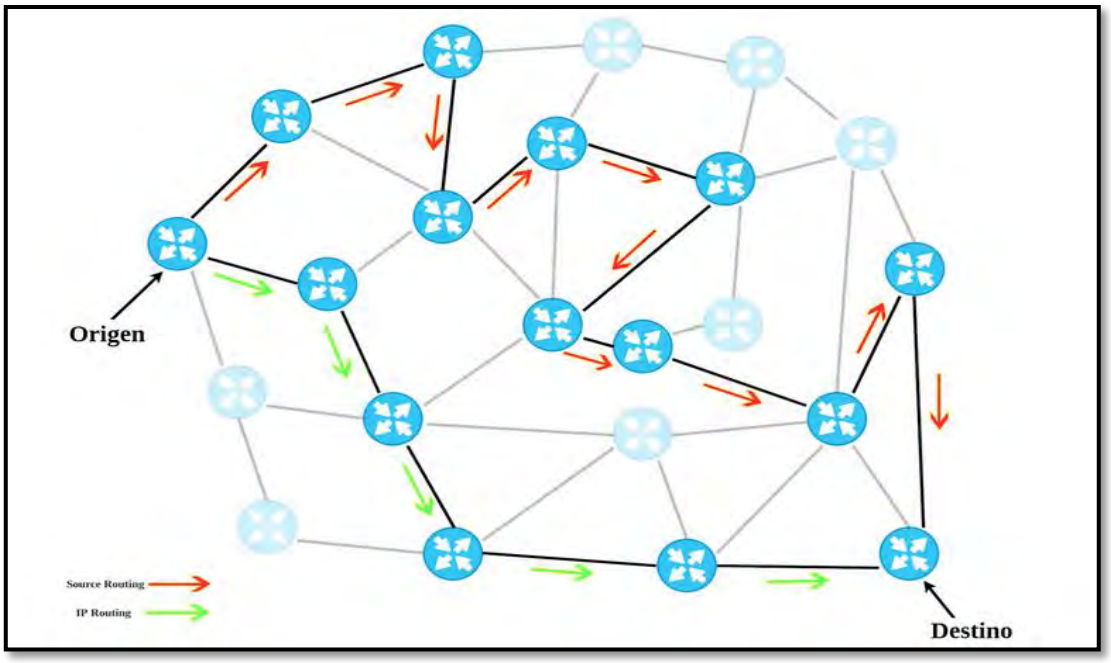


Figura 3-3 Source-Routing en un entorno IP
Recuperado de (Peñaloza, 2018)

En el RFC 791 se plantean dos tipos de *source-routing*:

- Loose Source and Record Route (LSRR): Camino definido en el origen, en el que los próximos saltos pueden o no estar directamente conectados. Esta forma de expresar la ruta es similar al funcionamiento de un GPS, donde se establecen puntos en el trayecto hacia el destino final.
- Strict Source and Record Route (SSRR): Es similar a LSRR, pero es necesario que los próximos saltos estén directamente conectados, caso contrario, el paquete es descartado.

El fin último de esta forma de enviar paquetes era contar con un mecanismo para resolución de problemas y así sea factible encontrar caminos alternos, aunque no los óptimos, hacia el mismo destino, pero sin almacenar estos caminos en una estructura de datos o tabla, como la tabla de enrutamiento, sino que estos caminos estén contenidos en un encabezado del paquete.

Con el paso del tiempo, la IETF entendió que no era adecuado llevar dentro de un encabezado la ruta a seguir, principalmente por motivos de seguridad, por ello, los *routers* modernos tienen deshabilitado por defecto la opción de *source-routing*.

3.1.1 Segment-Routing: Fundamentos y Origen

La idea de SR fue propuesta por Cisco Systems bajo el liderazgo de Clarence Filsfils en noviembre del 2012 (Davidson, 2017) y la IETF formó el grupo de trabajo SPRING (*Source Packet Routing in Networking*) en octubre del 2013 para el continuo desarrollo de esta tecnología de forma estándar, creando el RFC 8402[47] desarrollado en Julio del 2018.

La idea inicial de Clarence Filsfills, según un conversatorio dado por él en el año 2016 y publicado por *Tech Field Day* (Filsfils, 2016) permite migrar a un entorno SDN con mayor facilidad, pero esta idea surgió de analizar y comprender cómo un equipaje de aeropuerto se

[47] RCF 8402 – Segment Routing Architecture: https://tools.ietf.org/html/rfc8402

marcaba en el origen y tenía poca probabilidad de pérdida, a pesar del gran número de equipajes, viajes, rutas y trasbordos, lo que sin duda para Filsfils, revolucionará el envío de tráfico en ambientes WAN dentro de los próximo cinco a diez años.

En base al artículo de (Salazar Ch., Naranjo, & Marrone, 2018), se indica que SR es una tecnología de generación de túneles inteligentes u *overlays* que elegantemente cumple los requerimientos de conectividad WAN y soluciones de ingeniería de tráfico sin recurrir a otros protocolos, dando sencillez a la comunicación entre sedes, pero manteniendo el concepto de etiquetado, por ello, muchos consideran a SR como el futuro de MPLS.

SR emplea el paradigma de envío *source-routing* pero siguiendo una política de envío compuesta por instrucciones denominadas segmentos. Un segmento representa cualquier instrucción topológica identificada por su *Segment-Identifier* (SID). Este segmento es incluido en un *Segment-Routing Header* (SRH) dentro de un paquete IP en el *router* de origen, el cual es analizado a medida que dicho paquete viaja por la infraestructura, evitando mantener un estado del paquete en una estructura de datos en los nodos intermediarios, otorgando gran flexibilidad y sencillez en las políticas de enrutamiento.

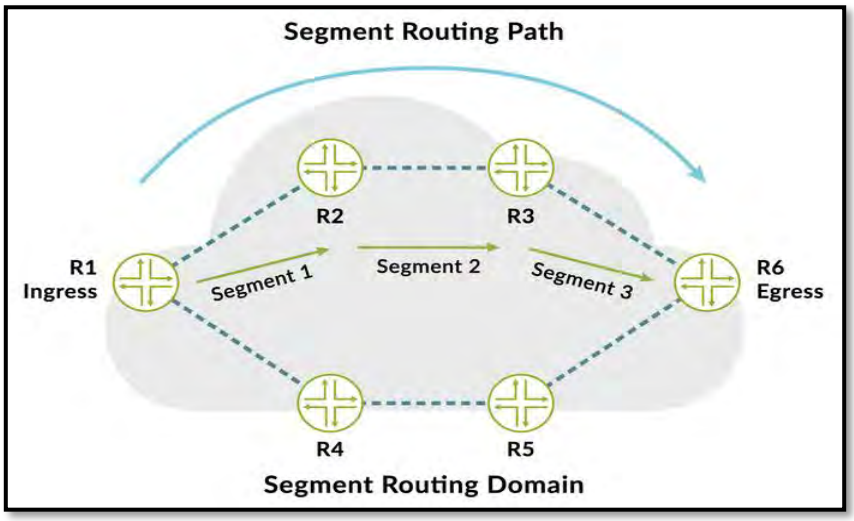


Figura 3-4 Source-Routing Path y Segment Routing Domain
Recuperado de (Juniper Networks, 2021)

3.1.2 Segment-Routing: Plano de Datos y Plano de Control

Plano de Datos

El plano de datos define el cómo se genera el proceso de encapsulación en SR, generando su encabezado (SRH) dentro de un paquete IP.

Según el RFC 8402 y por la investigación de (Salazar-Chacón & Reinoso, 2021), SR tiene dos tipos de implementaciones en cuanto al plano de datos u *underlay* se refiere:

- SR sobre MPLS: Funciona en conjunto con MPLS-LDP; y
- SR sobre IPv6 (SRv6): Funciona nativamente en un entorno IPv6.

Un SRH contiene una lista ordenada de segmentos según su SID.

Existen dos tipos de SIDs:

- SID con significado global: SID que tiene relevancia dentro de todo el dominio SR, lo que implica que todos los nodos de la nube SR conocen de su valor y la acción a realizar asociada al LFIB. Los rangos reservados para este SID van de 16000 a 23999. Este rango tiene un nombre especial, *Segment Routing Global Block* (SRGB), pero este rango puede diferir entre fabricantes.
- SID con significado local: SID que tiene relevancia en el segmento de origen (enlace entre el *router* que lo genera y el próximo salto) y toman un valor fuera del SRGB.

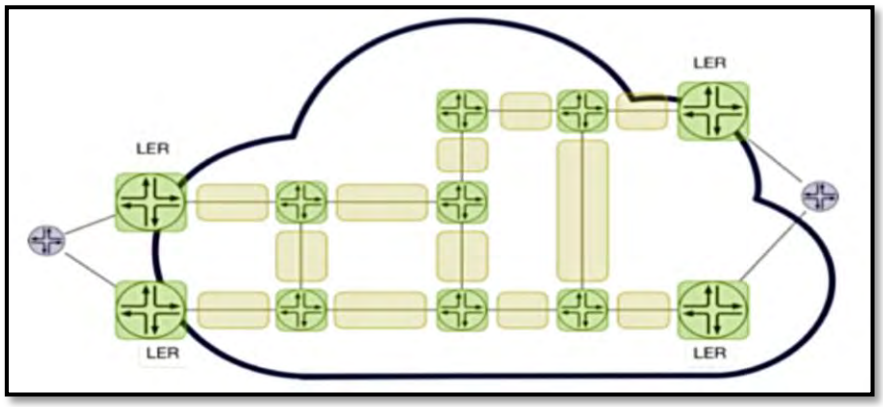


Figura 3-5 SIDs: Globales y Locales
Recuperado de (Arista Networks, 2021)

De acuerdo con (Filsfils, Michielsen, & Talaulikar, 2017), existen varios tipos de segmentos:

- *Node-SID*: Segmento de significado global, distribuido por un IGP de estado de enlace (IS-IS/OSPF), por ello, representa el camino más corto a un destino o prefijo especificado y tiene como símil el salto especificado en *Loose Source Routing*. Cabe decir que la configuración del valor de SID que representa a este segmento es configurada por el administrador de la infraestructura y es su responsabilidad mantenerlo único. Se recomienda configurar un Node-SID a las interfaces *Loopback*. Debido a estas características, se lo llama también IGP *Prefix Node Segment.*

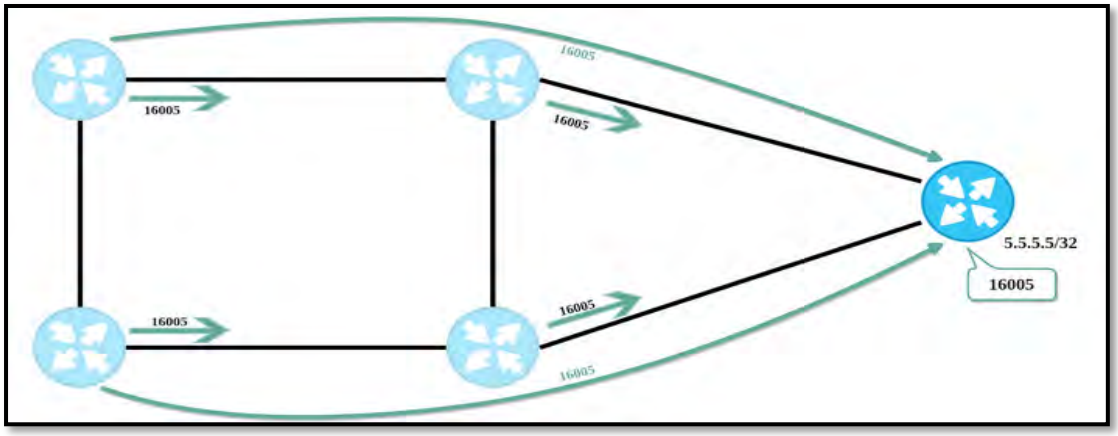


Figura 3-6 Node-SID
Recuperado de (Peñaloza, 2018)

- *Adjacency-SID*: SID distribuido por los IGPs de Estado de Enlace el cual describe una adyacencia en *routers* bajo el mismo IGP. A diferencia de los *Node-SID*, este tipo de segmento es auto asignado por el *router* y permite el cambio de etiquetas dentro de un *Node-SID*. Su función es símil a un *Strict Source Routing*.

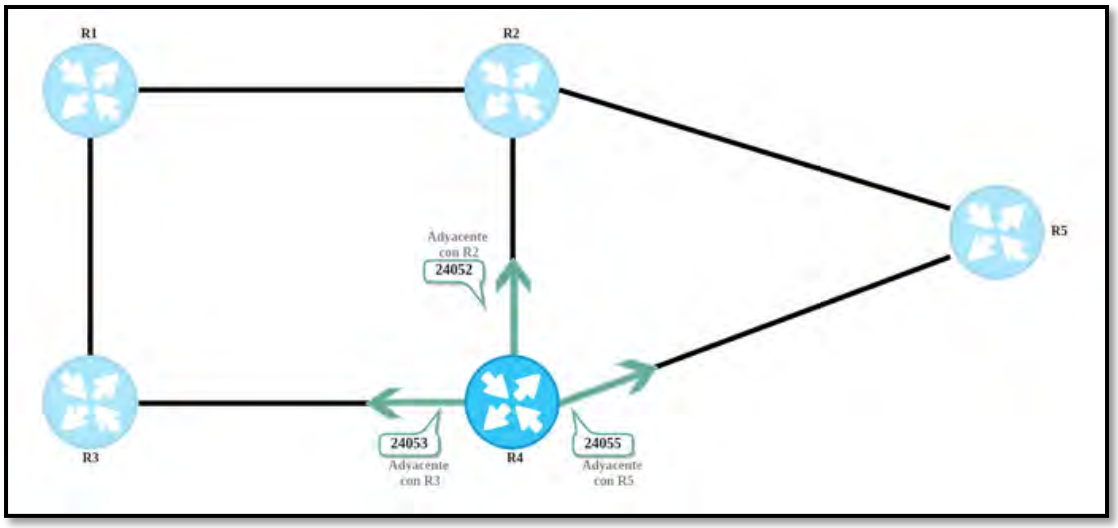


Figura 3-7 Adjacency-SID
Recuperado de (Peñaloza, 2018)

- *Service-SID*: Identificador de un servicio específico. Es un SID Local.
- *BGP-SID*: Similar a un IGP-SID, pero en lugar de IGP, muestra el camino a un prefijo aprendido por BGP y es anunciado por este protocolo de enrutamiento.

Todos estos segmentos pueden ser combinados e impuestos en un paquete en una transmisión en el dominio SR, logrando así todas las características que *source-routing* tiene, pero sin la limitación de nuevo saltos, así como proveer mejoras a nivel de ECMP (*Equal-Cost Multipath)* y *Fast-Reroute*.

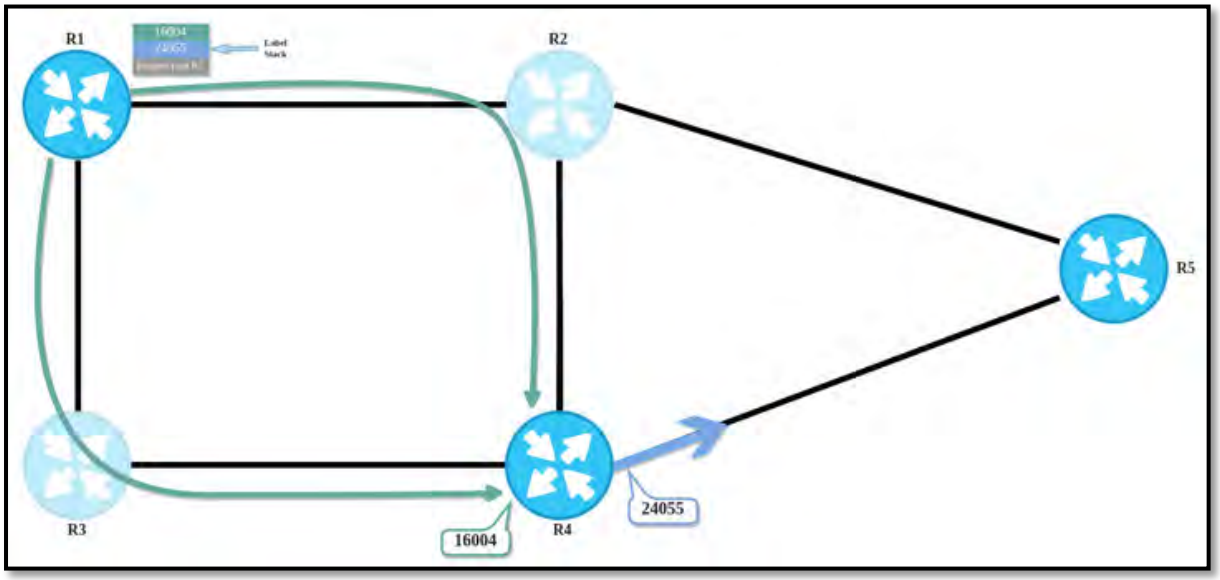


Figura 3-8 Combinación de Node-SID y Adjacency-SID en un mismo Dominio SR
Recuperado de (Peñaloza, 2018)

De igual manera que en MPLS-LDP, la facilidad de la inclusión de segmentos en un paquete a manera de etiquetas da un mejor entendimiento a SR, por eso, las operaciones que un nodo puede realizar con los segmentos en el dominio SR pueden mapearse con las operaciones en MPLS-LDP así:

Tabla 3-2 Operaciones en SR vs Operaciones en MPLS-LDP

Operación en SR	Operación en MPLS-LDP	Función/Característica
*SR *Header* (SRH)	*Label Stack*	Lista de segmentos o etiquetas
**Active Segment*	*Top Most Label*	Segmento/Etiqueta en uso actual durante la transmisión
Push	*Push*	Actualiza la lista de segmentos, ubicando el segmento activo en lo más alto de la lista
Continue	*Swap*	Cambio de segmento o etiqueta a medida que el paquete transita la nube SR
Next	*Pop*	Marca el siguiente segmento como segmento activo o quita el segmento de ser el caso

*No son operaciones, pero son parte fundamental de SR
Fuente: (Salazar Ch., Naranjo, & Marrone, 2018)

Es importante decir que *Segment Routing* puede ser implementando sobre una red MPLS-LDP sin cambiar el plano de datos de MPLS.

Plano de Control

El plano de control, por otra parte, define el cómo la información de los SIDs es enviada y actualizada en el dominio SR.

Node-SIDs y *Adjacency-SIDs* son anunciados en la red por IGPs que soportan módulos de SR, actualmente, IS-IS y OSPF e incluso una familia de enrutamiento (*address-family)* en BGP.

Tal como se menciona en (Salazar Ch., Naranjo, & Marrone, 2018), uno de los principales propósitos del plano de control es indicar al nodo de ingreso cómo seleccionar el camino. En la actualidad hay tres métodos para lograrlo:

- Configuración Manual: Un túnel estático de SR es configurado entre los nodos del dominio SR.
- Empleo de IGPs de Estado de Enlace como Underlay: El *router* de ingreso al dominio SR calcula el SPF (*Shortest Path First*) hacia el destino como base de una política de enrutamiento.
- A través de un Controlador SDN: SR provee un entorno que permite el uso de un controlador SDN como ODL mediante PCEP (*Path Computation Element*), ofreciendo gran capacidad de instanciar políticas de ingeniería de tráfico.

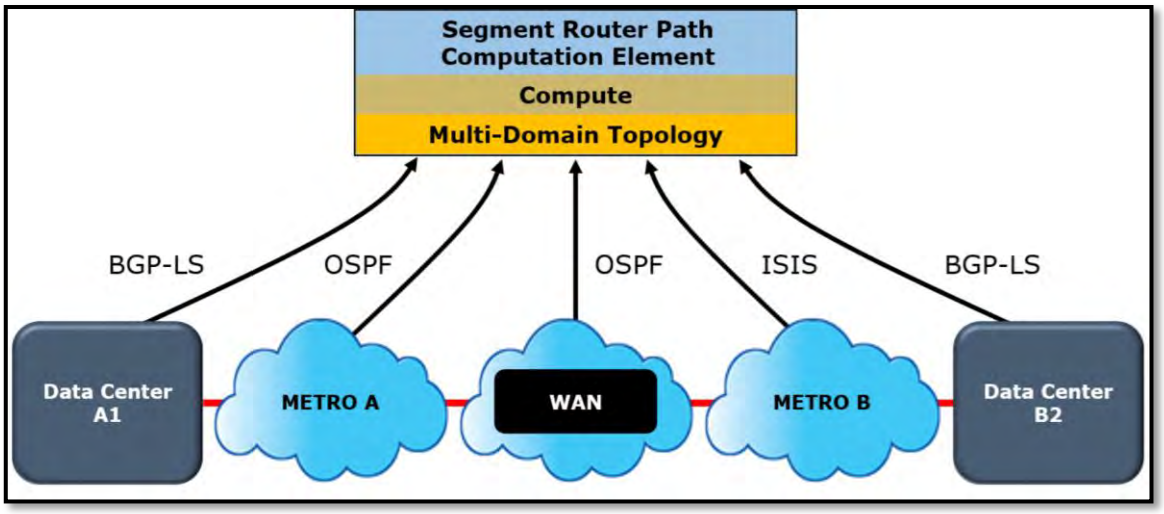


Figura 3-9 Esquematización de SR PCE bajo un concepto SDN
Recuperado de (Perrin, 2017)

En ***4.1 Emulación de Segment-Routing con MPLS de Plano de Datos*** , se realiza una emulación tanto bajo BGP como en una nube MPLS-LDP con el fin de probar el comportamiento de esta tecnología en escenarios PoC avalados en congresos internacionales, poniendo en práctica lo mencionado en cuanto al Plano de Datos y de Control de SR.

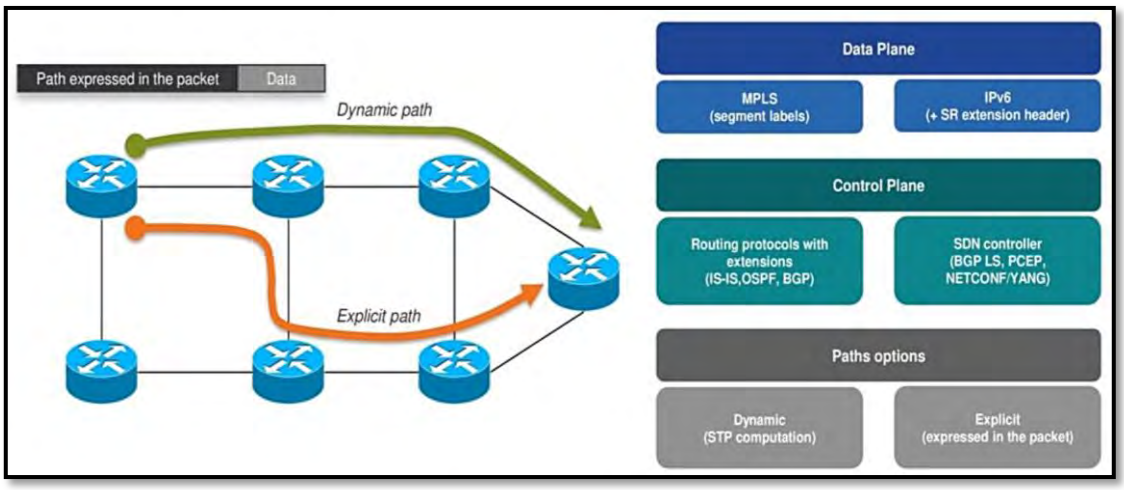


Figura 3-10 Plano de Datos y Plano de Control en SR
Recuperado de (Chaloupka, 2017)

3.2 Software-Defined Access/Software-Defined Branch y Software-Defined Data Center

Las redes del presente y futuro, además de ser vistas como una colección de dispositivos, medios y servicios, deben ser vistas como un sistema holístico, casi omnipresente en la mayoría de actividades humanas.

La escalabilidad de las redes ha llegado a niveles nunca pensados y el uso del tradicional CLI no permite la configuración y menos el control de todos los elementos de una red, incluso, métodos tradicionales de administración de redes como SNMP, han quedado obsoletos en la era de la Transformación Digital.

El personal a cargo de una infraestructura de TI busca maneras de controlar la red de forma sencilla, rápida y segura, es por ello por lo que, SDN y el análisis de Controladores de red son focos de estudio e investigación.

Un controlador SDN debe ofrecer la capacidad de administrar una red como si este fuera un sistema, en otras palabras, abstrayendo y automatizando las políticas presentes.

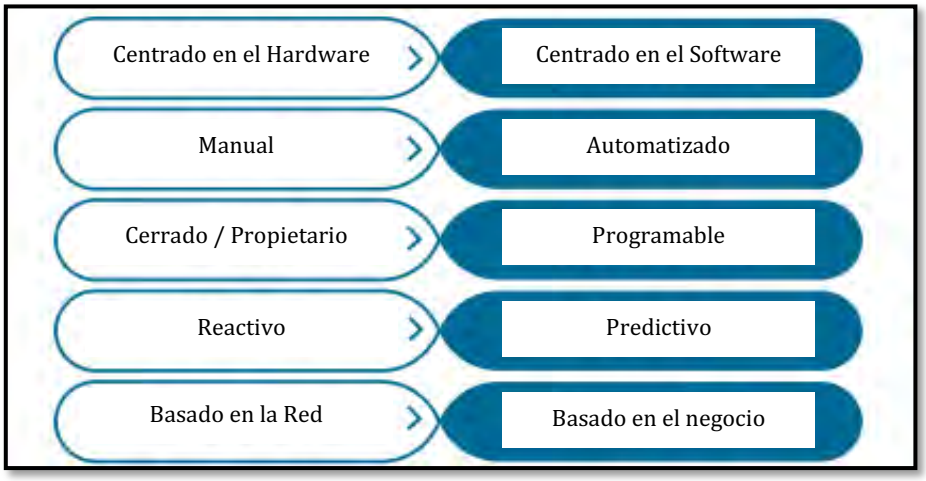


Figura 3-11 Evolución de los requerimientos de una red IT Basado en (Gooley, Hasan, & Vemula, 2021)

Para dar solución a estos cambios, un *framework* interesante se ha abierto camino, Cisco Systems lo denomina *Intent-Based Networking* o Redes Basadas en la Intención, mientras Huawei lo denomina *Intent-Driven Networks*[48], pero sin importar su nombre, estas tecnologías determinan lo que requiere un negocio para automáticamente traducirlo a tareas que debe realizar la red para dar sustento al negocio. Según (Gooley, Hasan, & Vemula, 2021), esta es una tarea con lógica circular con el fin de generar políticas de enrutamiento, seguridad, QoS, pero no basta con solo crear las políticas, el siguiente paso es orquestarlas ya que en la actualidad existen tanto elementos físicos como virtuales, la capacidad de monitoreo y de *insights* es fundamental, dando así visibilidad completa a la red.

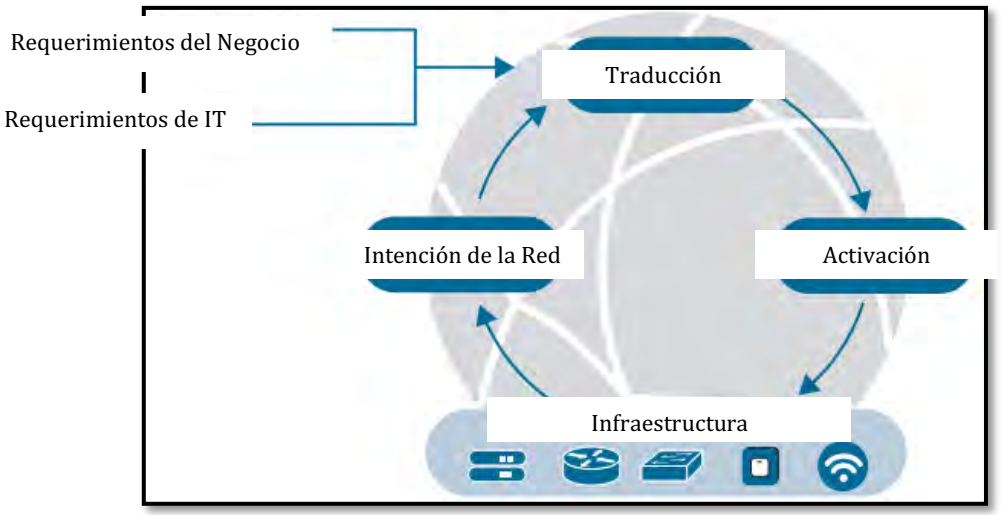


Figura 3-12 Conceptualización de Redes Basadas en la Intención Recuperado de (Gooley, Hasan, & Vemula, 2021)

[48] Intent-Driven Networks: https://www.huawei.com/en/news/2018/2/Huawei-Launches-the-Intent-Driven-Network-Solution

Diseñar una arquitectura de esas características sin duda es un reto para todos los profesionales en TI, por ello, la infraestructura de red es dividida en áreas funcionales que den lugar a un entorno SDN basado en la intención:

- Gestión de la WAN mediante SD-WAN
- Gestión de la red de campus o red empresarial mediante SD-Access/SD-Branch
- Gestión del Centro de Datos y el *Cloud* mediante SD-DC

Esta separación en áreas funcionales enfocadas a la digitalización y con miras a entornos SDN, muchos fabricantes lo denominaron *Digital Network Architecture*, arquitectura que se basa en la intención, está informada por contexto, siempre aprendiendo y con seguridad como eje transversal.

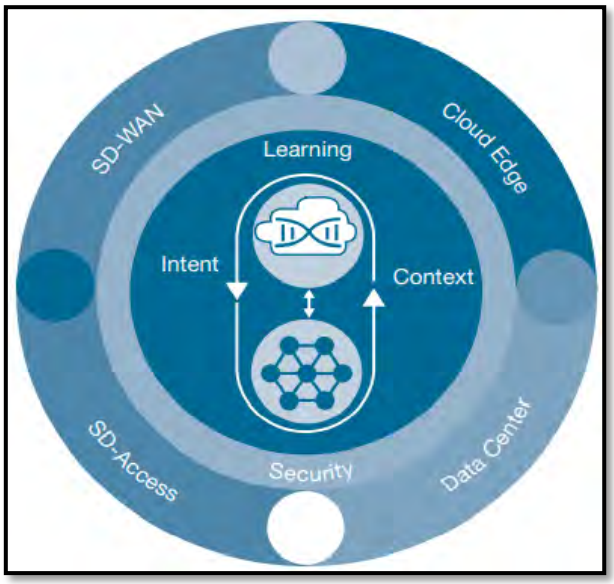


Figura 3-13 Cisco DNA – Digital Network Arquitecture
Recuperado de (Gooley, Hasan, & Vemula, 2021)

3.2.1 SD-Access/SD-Branch: Multidominios y Políticas comunes

Las demandas de los usuarios, incrementados por el contexto de pandemia COVID-19, han hecho que soluciones pasadas traten de solventar problemas de las redes del día de hoy. Es más, el campo del *networking* no se ha adaptado en la misma medida que las necesidades en tecnologías de la información: Las redes sirven para transportar datos y nada más, lo cual, durante esta tesis, se ha demostrado que no es su única función.

Una de las tecnologías que más se analiza su real beneficio es *Spanning-Tree Protocol* (STP), diseñado para evitar bucles L2, no obstante, puede ocasionar problemas en la red en caso de no ser manejado apropiadamente. Si bien STP ha evolucionado a PVST+, RSTP, MST, sigue generando dificultades, en especial al bloquear recursos. De igual manera, hay que tener cuidado cuando una red con STP tiene mecanismos de redundancia de primer salto o FHRP, siglas de *First Hop Redundancy Protocol*, ya que puede ocasionar problemas de envío.

Las redes, para evitar complicaciones, han optado por tener entornos L3 desde el acceso, lo que mejora las prestaciones y desempeño, de todas maneras, hay que tomar en cuenta que en un entorno L3, las VLANs no se propagan, se debe tener consideraciones especiales cuando haya redes inalámbricas con WLC, así como considerar que se aumentará el valor de la inversión.

Un diagrama de una típica red empresarial, bajo el modelo de diseño jerárquico de tres capas (Acceso, Distribución y Núcleo) se observa en las siguientes figuras:

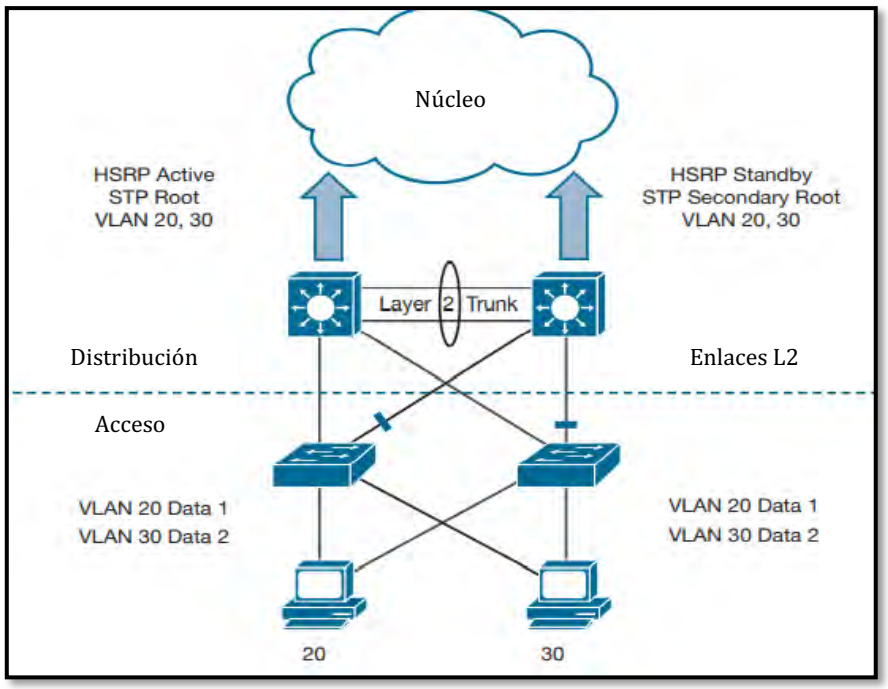


Figura 3-14 Diseño de red clásica empresarial mediante modelo jerárquico de tres capas Recuperado de (Gooley, Hasan, & Vemula, 2021)

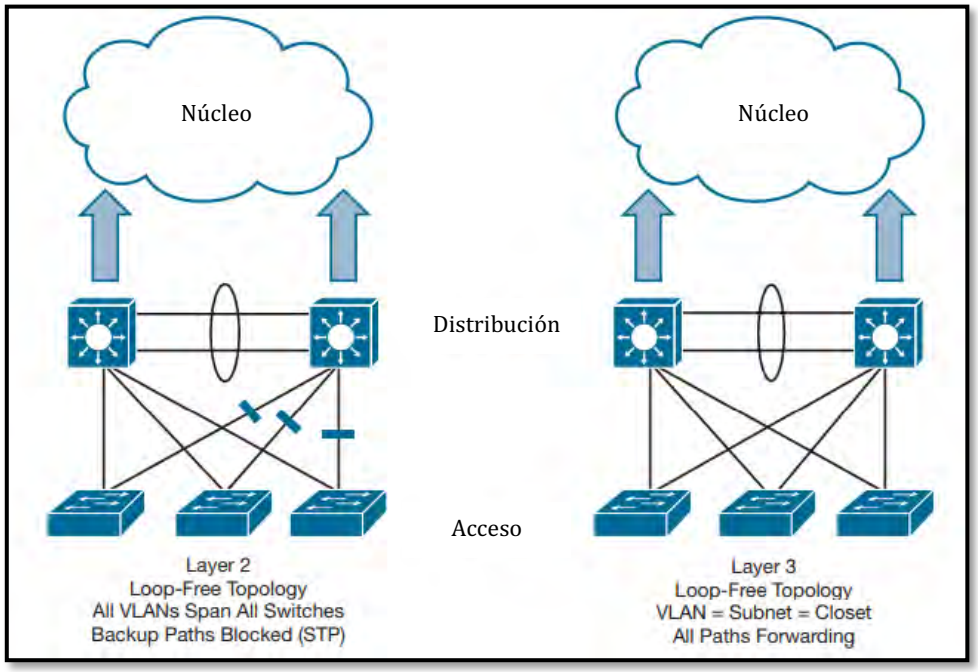


Figura 3-15 Diseño L2 de red clásica empresarial vs. Diseño L3 de red clásica empresarial Recuperado de (Gooley, Hasan, & Vemula, 2021)

Debido a que los datos ya no son estáticos y estos se generan en cualquier lugar de la red gracias a IoT y a BYOX, los diseños clásicos mostrados en las *Fig. 3-14* y *Fig. 3-15* no son los óptimos y las tendencias en IT marcan el camino hacia la definición de Multidominios, generando seguridad, simplicidad y un mejor control para la administración.

Un punto clave de entornos multidominio es contar con una política que pase por todos los sectores generadores de datos, tratando de otorgar consistencia y tráfico determinista.

La generación de multidominios es el paso inicial a la creación del denominado SD-Access o SD-Branch, términos acuñados por diversos fabricantes, en la que la política establecida es compatible con el lado SD-WAN. Una esquematización de ello se observa en la *Fig. 3-16*.

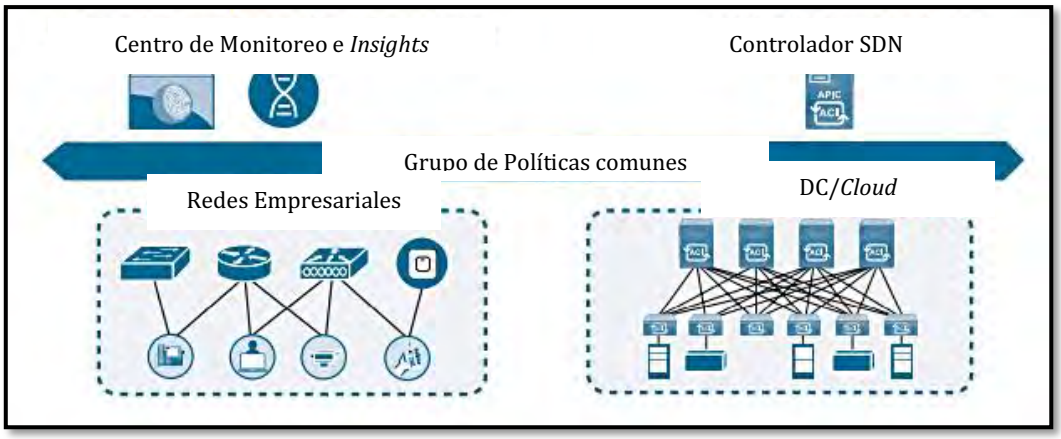


Figura 3-16 Creación de Multidominios en la red empresarial – SD-Access/SD-Branch Basado en (Gooley, Hasan, & Vemula, 2021)

La inclusión de entornos tipo *Cloud* también complica el panorama en una red empresarial, pues aparecen conceptos como *Shadow IT*, es decir, líneas del negocio (LoB) que circulan a través de diversos proveedores de *Cloud* para entregar el servicio sin control directo de los departamentos de TI, causando dificultades a nivel de privacidad y seguridad en general.

Solamente redes de empresariales de próxima generación serán capaces de abordar estas dificultades y entregar adecuadamente los servicios y una excelente experiencia de usuario uniendo el lado LAN con el WAN, tal como se aprecia en la *Fig. 3-17*.

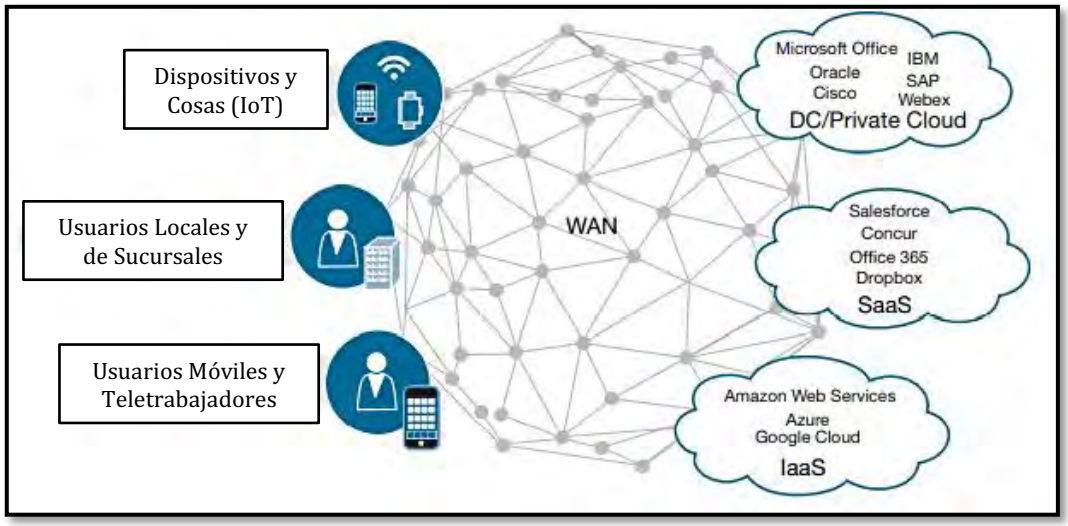


Figura 3-17 Red de Campus de próxima generación: SD-Access + SD-WAN + Multidominios + MultiCloud Basado en (Gooley, Hasan, & Vemula, 2021)

3.2.2 SD-Access/SD-Branch: Conceptualización y Arquitectura (*Fabric* de SD-Access mediante LISP)

SD-Access/SD-Branch surgió como resultado de la modernización de la red de Campus hacia una red de próxima generación en la que modelos y tecnologías tradicionales no permiten entregar el servicio de la manera más adecuada.

SD-Access se implementa bajo la premisa de la creación de redes con el paradigma *Underlay-Overlay*, dando a la red empresarial una red lógica robusta, segura, simple y de fácil monitoreo y gestión.

La red *Underlay* está compuesta por los mismos equipos de la red tradicional: *switches, routers, firewalls,* equipos de seguridad como IDS/IPS, etc. La principal función de la red *underlay* es la de conectar los nodos y generar comunicación entre ellos.

El *fabric* de SD-Access es la red *Overlay,* la cual separa el Plano de Envío del Plano de Control de forma lógica, simplificando de esa manera a la red subyacente (*underlay*). Se integra además un plano denominado Plano de Políticas/Orquestación. La *Fig. 3-18* muestra el concepto del paradigma *Underlay-Overlay*.

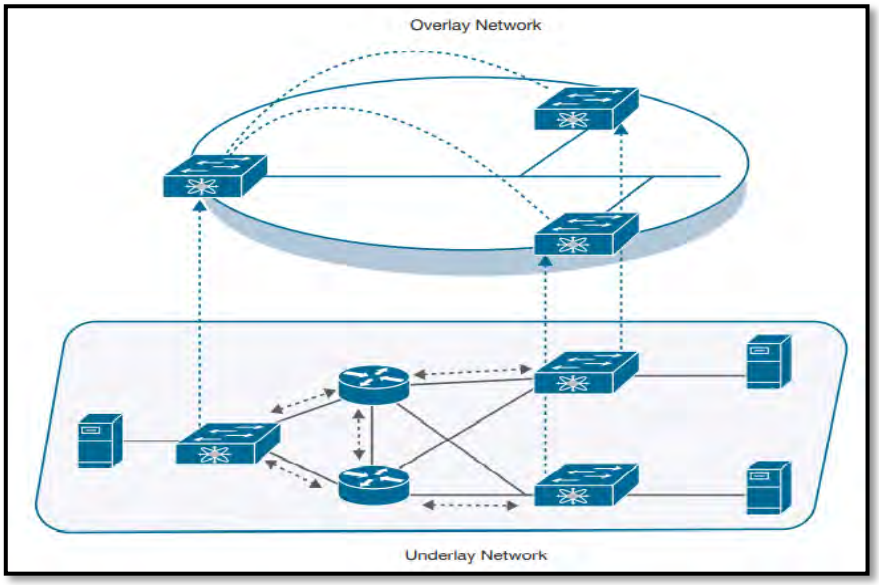


Figura 3-18 Paradigma Underlay-Overlay Basado en (Gooley, Hasan, & Vemula, 2021)

Los protocolos que se pueden usar en SD-Access y construir el *Fabric* necesario son:

- Plano de Control basado en el protocolo LISP (*Locator ID Separation Protocol*).
- Plano de Datos basado en VXLAN (*Virtual Extensible LAN*).
- Plano de Políticas y Orquestación mediante Controlador SDN y/o de seguridad.

LISP es considerado por (Cisco Systems, 2018) como una arquitectura de red y protocolo que implementa el uso de identificadores o *namespaces* en lugar de una sola dirección IP para identificar un nodo en la infraestructura, desarrollando así un sistema de mapeo bajo demanda.

Un nodo en LISP podrá tener:

- *Endpoint-ID* (EIDs), los cuales se asignan a dispositivos finales.

- *Routing o Record Locators* (RLOCs), asignados a dispositivos intermediarios, principalmente *routers* y generar un sistema de ruteo global.

Al contar con esos identificadores, se puede administrar mejor las infraestructuras que tienen gran cantidad de usuarios y mantener coherencia en su direccionamiento IP, ya que los EIDs tendrán una determinada dirección IP y los RLOCs otra dirección IP, es decir, LISP separa la identidad de la localización de un equipo, evitando así enrutamientos subóptimos.

La *Fig. 3-19* muestra una esquematización de LISP:

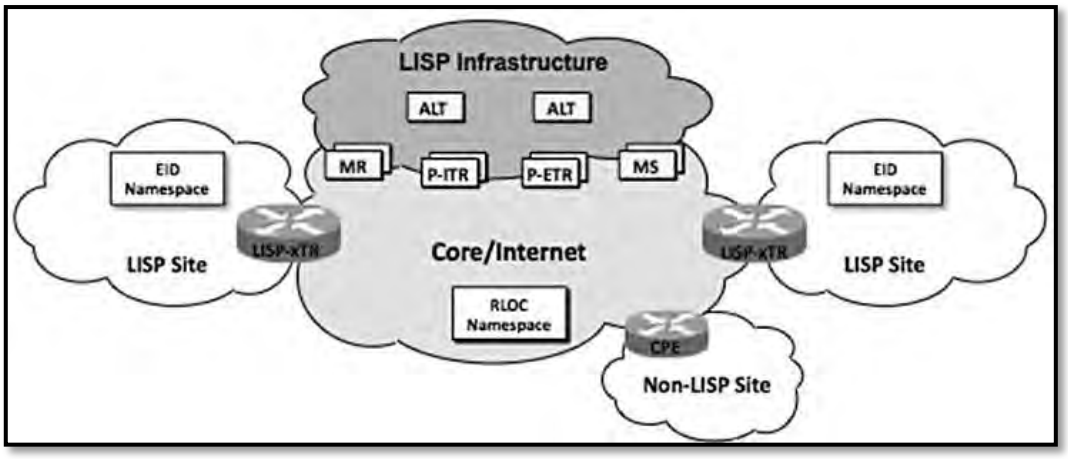


Figura 3-19 Esquematización de LISP
Recuperado de (Cisco Systems, 2018)

El funcionamiento de LISP permite que al conectarse un usuario a la infraestructura, el dispositivo de acceso registra su EID: Registra su dirección IP y/o dirección MAC, así como su ubicación a través de su interfaz *Loopback* como RLOC hacia un servidor de mapeo que funciona de forma similar a un servidor DNS que posee todos los EIDs y RLOCs de la red, de esta manera, cuando se requiere que el tráfico se dirija a un destino en particular, el dispositivo de origen solicita el RLOC del destino al servidor de mapeo, disminuyendo la cantidad de entradas en la tabla de enrutamiento de los equipos.

En otras palabras, LISP funciona bajo un paradigma de solicitud-respuesta, en el que los equipos no mantienen el estado de las redes, sino que le preguntan a un servidor de mapeo por la información necesaria, generando un Plano de Control manejable e ideal para el entorno SD-Access/SD-Branch. Este concepto es muy similar a entornos tipo iWAN con el protocolo NHRP.

La segmentación en entornos multidominio con gran cantidad de usuarios es un requerimiento necesario. Tradicionalmente esto se logra con VLANs, ACLs e incluso con VRFs (*Virtual Routing and Forwarding*), si bien estos métodos logran segmentar la red, no son lo suficientemente escalables ni tampoco seguros, en especial en entornos con múltiples *Clouds*, por ello, es mejor identificar al usuario y no seguirlo únicamente por su dirección IP. Mecanismos como *TrustSec*[49] segmentan la red en base a etiquetas denominadas *Scalable Group Tags* o SGTs, con ello, las políticas se aplican a los SGTs en lugar de a una dirección IP y así traducir una intención de negocio a una política de red.

[49] Cisco TrustSec: Cisco TrustSec Software-Defined Segmentation - Cisco

Se realizó una PoC de LISP en ***4.2 Emulación de LISP como Fabric de SD-Access*** y de esa manera, comprobar la factibilidad de uso y configuración de este protocolo.

VXLAN, definido en el RFC 7348[50] como una técnica de encapsulación y creación de túneles IP/UDP (Salazar Ch & Naranjo, 2017), tiene la función de ser el Plano de Datos de SD-Access gracias a sus características intrínsecas como creación de VNIDs, soporte en L2 y L3 y sencillez de configuración (Ver ***Fuente: Autor***

Configuración de VXLAN, parte del ***Anexo D: Ansible para entornos NetDevOps en infraestructuras de Red***).

Pero, para que VXLAN se integre plenamente con LISP y formen el *fabric* de SD-Access, la especificación original de VXLAN se convierte en VXLAN-GPO, actualmente como IETF *Draft* denominado VXLAN *Group Policy Option*, para incluir un campo para la etiqueta SGT de LISP.

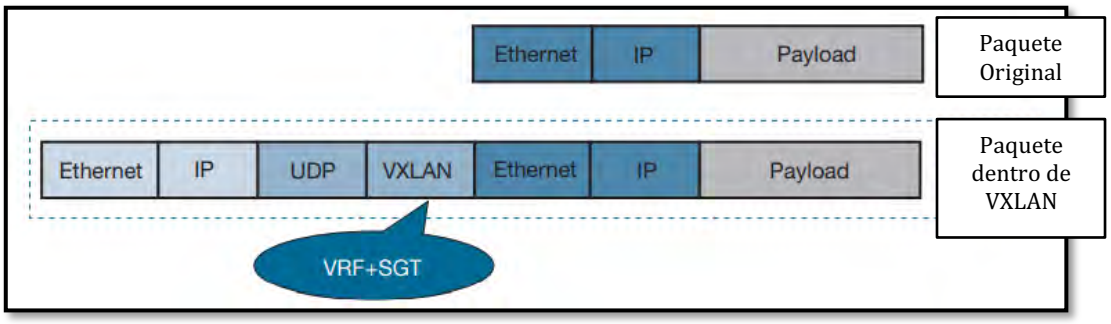


Figura 3-20 Encapsulación de VXLAN-GPO
Recuperado de (Gooley, Hasan, & Vemula, 2021)

3.2.3 SD-Access: Roles

Los componentes principales de SD-Access se muestran en la *Fig. 3-21*.

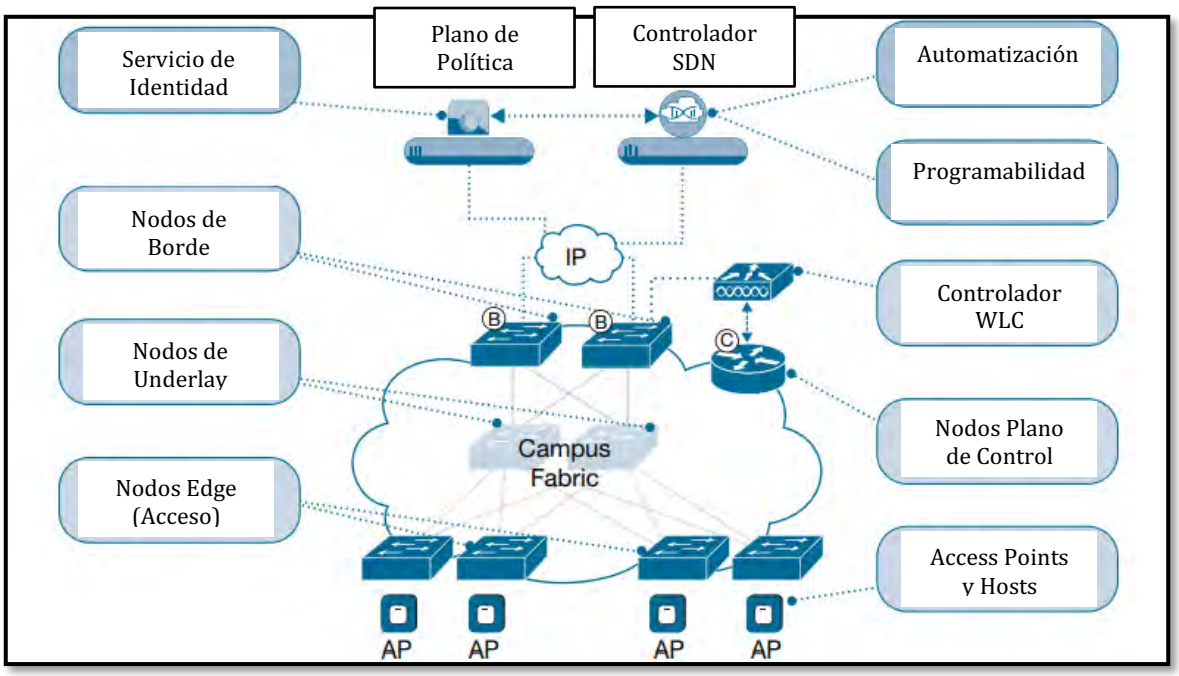


Figura 3-21 Componentes y Roles de SD-Access
Recuperado de (Gooley, Hasan, & Vemula, 2021)

[50] RFC 7348 VXLAN: https://tools.ietf.org/html/rfc7348

Componentes de SD-Access:

- Controlador SDN: Controlador que provee automatización basada en la intención para los usuarios, así como analítica para verificar flujos de datos, telemetría e *insights*. Puede implementarse con un controlador abierto como ODL o propietario como el *Cisco DNA Center.* Es factible implementar técnicas de programabilidad bajo *NetDevOps* de igual manera.
- Plano de Políticas: Equipo que dinámicamente agrupa los usuarios finales con su SGT para la aplicación de las políticas basadas en la intención. Puede implementarse mediante Cisco ISE, Citrix Gateway, FortiNAC, AWS Resource Access Manager o cualquier equipo que realiza control de acceso, postura, portal cautivo y co-locación.
- Nodo de Plano de Control: Es el equipo encargado de resolver y registrar a los usuarios finales con el fin de asignarles su EID (*Endpoint ID*). Tiene mucha importancia dentro del *Fabric* de SD-Access, pues permite la movilidad de los *hosts*. En la *Fig. 3-21* está marcado con la letra C.
- Nodo de Borde: Equipo que conecta la red externa (fuera de SD-Access) con la red interna a nivel L3. En la *Fig. 3-21* están marcados con la letra B.
- Nodo *Edge*: Equipo considerado de acceso. Es el encargado de concentrar y dar acceso a los usuarios finales. Debido a que son el acceso, tienen la responsabilidad de brindar autenticación y autorización (Conceptos de AAA – Autenticación, Autorización y Auditoría). Normalmente, en entornos de mucha movilidad, se emplean *Anycast* L3 *Gateways*, para tener un router que funcione como puerta de enlace común.
- Controlador WLC (Opcional): Permite la inclusión de APs ligeros a la red, formando un túnel CAPWAP-VXLAN con los nodos *Edge*.

3.2.4 SD-Data Center: Conceptualización e Importancia

En base a (IBM, 2021), SD-DC es un centro de datos donde los recursos y servicios empresariales son aprovisionados, monitoreados y administrados mediante técnicas y procesos definidos por *software*.

Por lo visto hasta el momento, una de las razones para los cambios de paradigmas en redes hacia entornos SDN, es la agilidad de implementación con seguridad y una de las maneras de lograrlo, es con SD-DC en el área de procesamiento de datos corporativos.

IBM también menciona que el uso de *Cloud,* virtualización de servicios y virtualización de la red son parte de la infraestructura empresarial moderna, por lo que aplicar los conceptos de SDN al mundo del DC es una obligación de los departamentos de IT.

Según (SDxCentral, 2015), al virtualizar el DC, todos sus recursos tanto de almacenamiento como de procesamiento pueden abstraerse en una forma de *software*, es así como la ventaja real para los usuarios es que toda la infraestructura para el tratamiento de los datos que circulan por la red se abstraiga en un *Cloud*, reduciendo así costos en CaPEX y OpEX.

La integración entre SD-Access (también llamado SD-LAN o SD-Branch), con SD-DC y SD-WAN es primordial para tener redes de próxima generación que apalanquen los objetivos empresariales del siglo XXI, tal como las vemos en la *Fig. 3-22.*

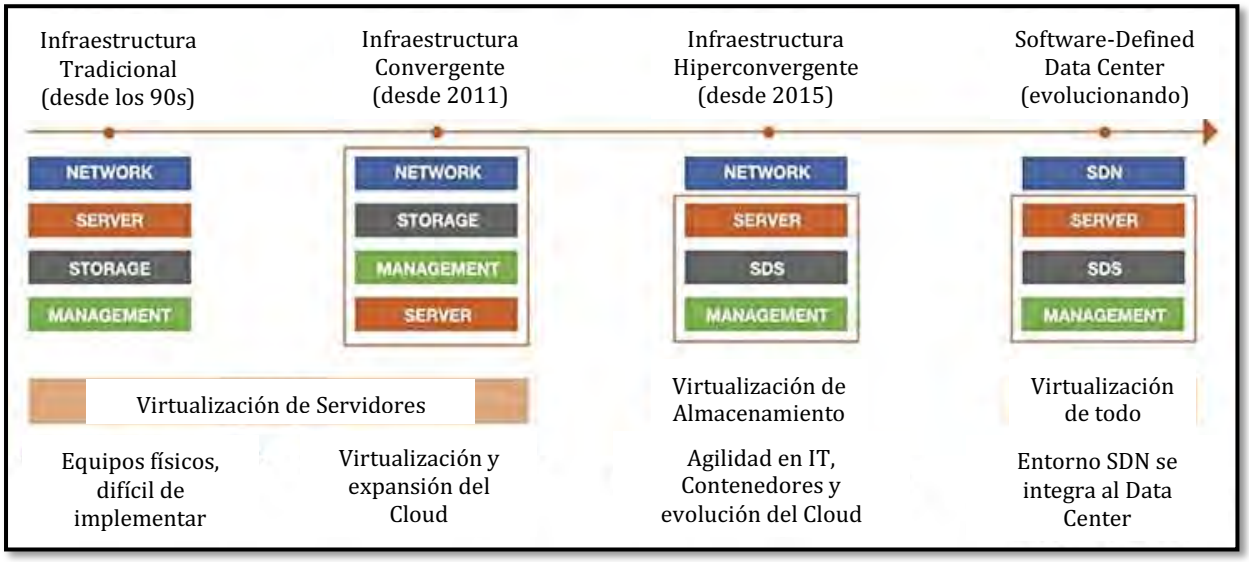


Figura 3-22 Evolución del Data Center
Recuperado de (Data Center Frontier, 2018)

3.3 Software-Defined WAN: Propuesta de rediseño del transporte de información y control de camino en redes Underlay y Overlay

El rumbo hacia Redes de Nueva Generación representa una evolución trascendental en términos de protocolos, estandarización y de entrenamiento del talento humano para cubrir las necesidades de los usuarios modernos, pero conseguir esos objetivos planteados en SLAs corporativos tomaría un costo elevado en cuanto a inversión, además de mucho tiempo en el diseño e implementación si se realiza con tecnologías tradicionales.

Por estos motivos, surgió el concepto de una WAN híbrida, la cual cuenta con enlaces adicionales/*backups* que no son MPLS y así contar con escalabilidad y resiliencia en la red. Estos enlaces redundantes o usados para ciertas aplicaciones del negocio toman el nombre de "**Independencia de Transporte**", ya que pueden ser creadas con diversas tecnologías tales como enlaces de Internet de banda ancha, L2VPNs o incluso enlaces 4G-LTE/5G, generando así libertad en la elección del *Underlay*.

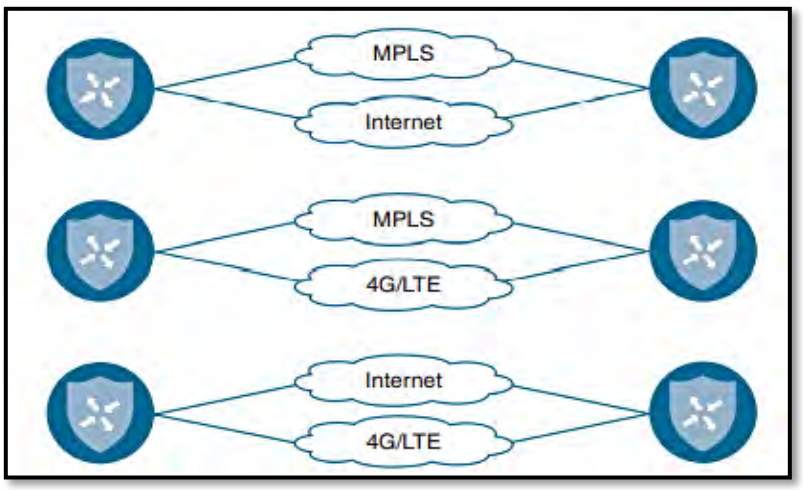


Figura 3-23 La WAN Híbrida – Independencia de Transporte
Recuperado de (Gooley, Yanch, Schuemann, & Curran, 2021)

Es obvio mencionar que estas redes empresariales requieren del establecimiento de políticas de enrutamiento y de seguridad con el fin de interconectar sus sedes. Estas políticas serían muy complejas de establecer con el paradigma de *networking* tradicional, incluso con técnicas como PBR (*Policy-Based Routing*), por ello, se debe repensar a la WAN pero con el paradigma SDN en mente, naciendo de esa manera SD-WAN tal como se observa en la *Fig. 3-24*.

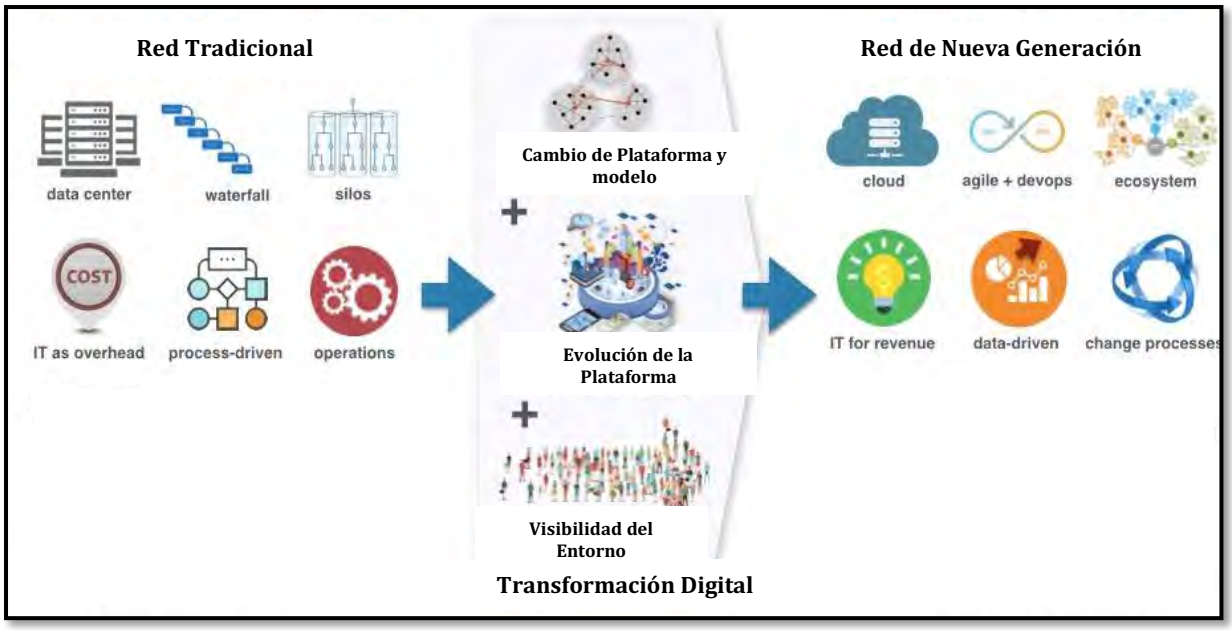


Figura 3-24 Redes de Nueva Generación
Recuperado de (Hinchcliffe, 2018)

Llegar al concepto de SD-WAN no fue repentino, es más, según (Salazar-Chacón & Reinoso, 2021), se tuvo un proceso gradual, comenzando con entornos conectados mediante DMVPNs-IPSec, luego con iWAN y finalmente SD-WAN.

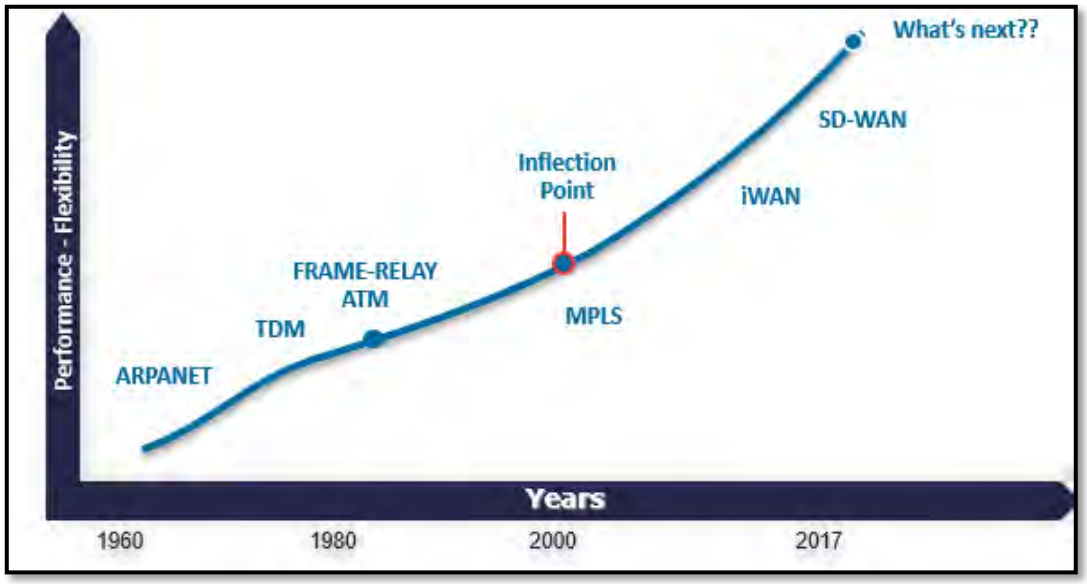


Figura 3-25 Evolución de la WAN con miras a SD-WAN
Recuperado de (Salazar & Solano, Fundamentos de Cisco Intelligent WAN, 2018)

3.3.1 SD-WAN: Rediseñando la WAN

Las WANs en la actualidad se han rediseñado para dar soporte a los requerimientos de conectividad a larga distancia de usuarios y empresas. Según un análisis de campo y por experiencia propia, se puede decir que las áreas clave de mejora son:

- Conectividad segura y flexible.
- Pensando en entornos multidominio y/o *multicloud.*
- Calidad de experiencia (QoE) al máximo nivel, más aún, en momentos donde la telemática ha llegado a campos de la educación, salud y economía.
- Operaciones ágiles.

La seguridad debe ser una parte integral de toda red, por ello, en tiempos modernos la segmentación y establecimiento de políticas claras es fundamental, es así como los conceptos de SDN analizados en ***2. Fundamentos de las Redes Definidas por Software*** se aplican en las redes WAN, separando el plano de control, datos y administración en toda la infraestructura utilizando el modelo denominado *Zero-Trust*[51], el cual significa que todo proceso debe ser autenticado y autorizado, tal como se aprecia en la *Fig. 3-26*.

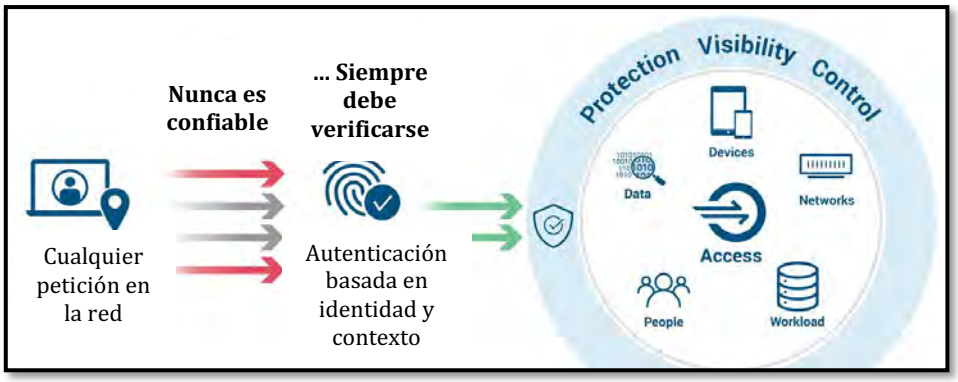


Figura 3-26 Modelo de Seguridad Zero-Trust Basado en (Cloudfare, 2021)

De igual manera, la operación y administración de una red WAN debe estar orientada al servicio, coordinando el enrutamiento y la orquestación de manera simple y rápida, debe ser un entorno basado en políticas y SLAs, donde APIs dan dan acceso al Controlador, gestionando la red de forma proactiva.

[51] Modelo Zero-Trust: https://www.welivesecurity.com/la-es/2020/09/14/zero-trust-que-es-modelo-seguridad-crecio-adopcion/

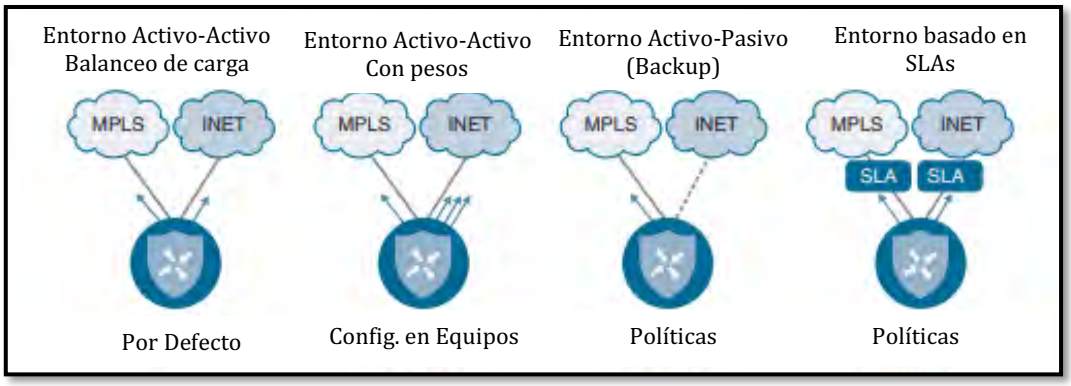


Figura 3-27 SD-WAN basado en políticas
Basado en (Gooley, Yanch, Schuemann, & Curran, 2021)

Un ejemplo de SLA y gestión de tráfico se observa en la *Fig. 3-28*, en la que la WAN permite mantener parámetros como *jitter*, pérdida de paquetes y latencia en valores establecidos en la política de enrutamiento empresarial, utilizando *multipaths* e independencia de transporte.

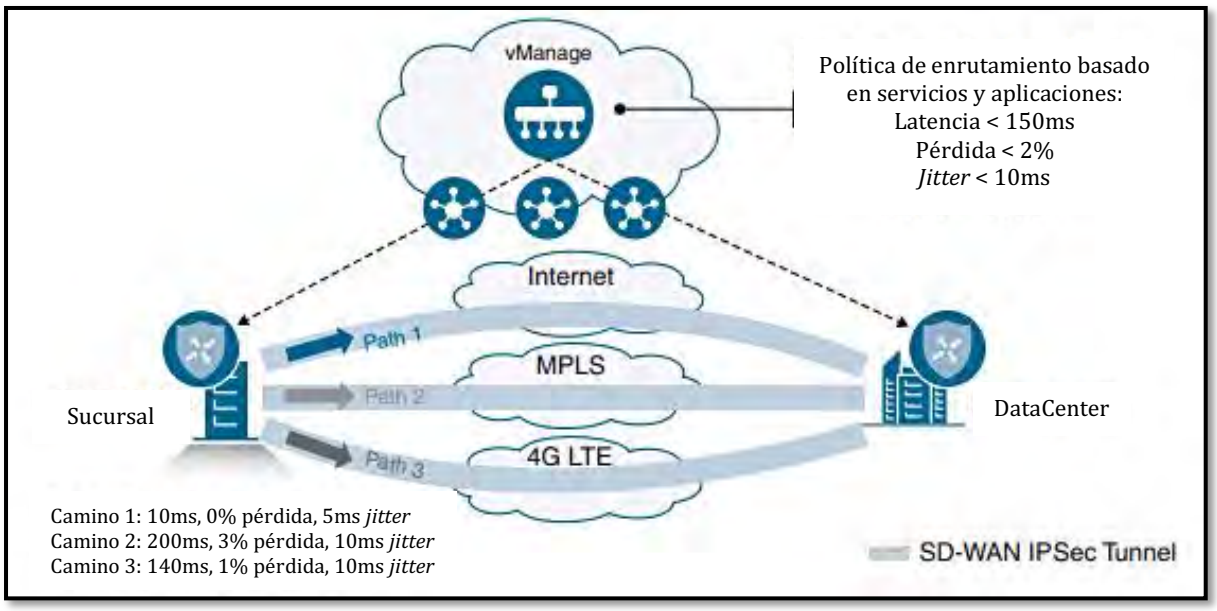


Figura 3-28 Ejemplo de SD-WAN basado en políticas: Viptela
Basado en (Gooley, Yanch, Schuemann, & Curran, 2021)

En cuanto a seguridad, la segmentación y encriptación de datos en tránsito a través de túneles IPSec bajo demanda son las mejores opciones, pues al no mantener una VPN fija, se evita la posibilidad de que intrusos ingresen u observen los datos y siendo estas VPNs bajo demanda, es posible segregar el tráfico según los datos que estén circulando. Este entorno fue concebido gracias a iWAN (*Intelligent WAN*), por muchos considerado el antecesor de SD-WAN.

Un *webinar* internacional presentado por el Gustavo Salazar (Salazar & Solano, 2018), autor de la tesis, sobre iWAN puede observarse en el siguiente enlace:

https://community.cisco.com/t5/videos-routing-y-switching/webcast-video-fundamentos-de-cisco-intelligent-wan/ba-p/3399530

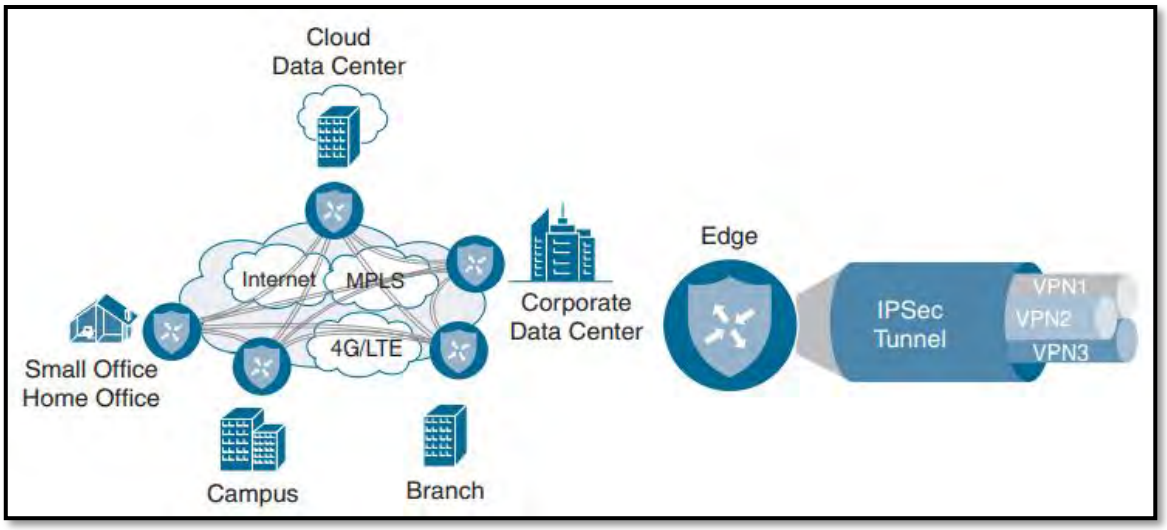


Figura 3-29 SD-WAN: Túneles IPSec bajo demanda - Segmentación
Recuperado de (Gooley, Yanch, Schuemann, & Curran, 2021)

3.3.2 SD-WAN: Pronto ROI, Fabricantes y Mercado Mundial

Las infraestructuras de red optan por consolidarse como una solución totalmente centralizada, dando flexibilidad no solo a la red, sino al negocio, incluso permitiendo que sea éste quien tome control de las políticas a implementar, moviéndose así a un modelo más OpEX que un modelo CapEX, en especial al haber fabricantes que pueden rentar la infraestructura a una empresa, por ello, uno de los primeros pasos al considerar una migración de una red tradicional WAN a SD-WAN es verificar el retorno de inversión (ROI) que produciría.

En base al análisis de costos realizado por (Gooley, Yanch, Schuemann, & Curran, 2021), se tiene un ahorro mensual cercano al 60% al comparar una solución dual de MPLS (WAN dedicada) que usando un entorno de independencia de transporte mediante acceso a Internet o 4G/5G.

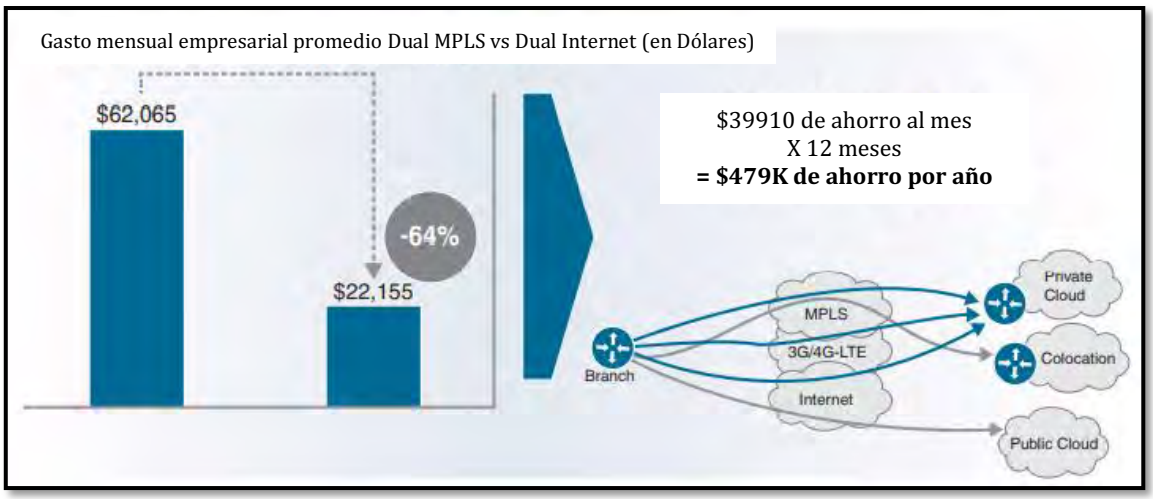


Figura 3-30 SD-WAN: Ahorro y pronto ROI
Recuperado de (Gooley, Yanch, Schuemann, & Curran, 2021)

Por todos estos motivos, SD-WAN es una tecnología que está revolucionando la red empresarial.

Fabricantes como Fortinet, Cisco, VMWare Velocloud, Citrix, Versa Networks y CloudGenix (PaloAlto) son los más representativos en el mercado SD-WAN mundial, la mayoría de ellos ofreciendo simplicidad, optimización de tráfico y uso del paradigma SDN.

Con el fin de comparar a estos fabricantes, se decidió usar los siguientes parámetros:

- ASIC de próxima generación: Es aquella ASIC diseñada para tareas especializadas del entorno SD-WAN, generando gran escalabilidad y eficiencia.
- Integración con equipamiento de Seguridad avanzada: Una adecuada solución SD-WAN debe nativamente incorporar equipos como NGFW (*Next-Generation Firewalls*)
- Buen TCO por Mbps protegido: Medida clara del TCO (*Total Cost of Ownership*) en términos de Mbps protegidos.
- Disponibilidad Multiplataforma e Interoperabilidad: Equipos SD-WAN deben incluir tanto factores de forma físicos como virtuales que permitan la implementación de esta tecnología.
- Selección Dinámica de caminos: SD-WAN mejora la resiliencia de la red a través de selección dinámica de caminos mediante independencia de transporte.
- SD-Access segura: La convergencia entre la WAN y el acceso a la red reduce riesgos, incrementa la agilidad y permite una mejor administración.

En base al estudio realizado por (Fortinet, 2021), así como de una investigación de campo, se llegó a la siguiente tabla comparativa:

Tabla 3-3 Comparación de diversos fabricantes de soluciones SD-WAN

	Fortinet	**Cisco Viptela**	**HPE Silver Peak**	**VMware VeloCloud**	**Palo Alto Prisma SD-WAN**	**Versa Networks**
NG-ASIC	SÍ	SÍ	NO	NO	NO	NO
Integración con Equipos de seguridad avanzada	SÍ	SÍ	NO	NO	NO	SÍ
TCO por Mbps protegido	$4	Desconocido	$37	$28	Desconocido	$10
Disponibilidad Multiplataforma e Interoperabilidad	SÍ	SÍ	SÍ	SÍ	SÍ	SÍ
Selección Dinámica de Caminos	SÍ	SÍ	SÍ	SÍ	NO	NO
SD-Access/SD-Branch Segura	SÍ	SÍ	NO	NO	NO	NO

Fuente: (Fortinet, 2021)

Sin importar el tipo de fabricante, todos buscan la mejora de la WAN en cuanto a productividad se refiere, así como dotar de seguridad a la infraestructura de extremo a extremo.

La llegada de la pandemia COVID-19 trajo muchas dificultades a todo el mundo, pero fue posible mantener los sistemas críticos como la educación, salud y sector energético a través del uso de aplicaciones basadas en la nube y otras tecnologías que dieron soporte a este tipo de servicios para el desenvolvimiento desde la casa durante los confinamientos. Esta coyuntura permitió el despliegue de SD-WAN con más rapidez, sin embargo, debido al cierre de fábricas de elementos y piezas, el desarrollo y evolución de SD-WAN tuvo una barrera que lo afectó a corto plazo, en especial en la creación de dispositivos, a pesar de ello, SD-WAN creció positivamente, ya que la implementación de redes 5G es un hecho y 5G requiere de infraestructuras de red tipo SD-WAN para lograr una adecuada infraestructura moderna capaz de soportar flujos masivos de datos en tiempo real.

El mercado mundial de SD-WAN tiene mucho por crecer, es más, según (Global Market Insights, 2020), se espera una tasa de crecimiento anual compuesto (CAGR – *Compound Annual Growth Rate*) que bordea el 60% entre el año 2020 y 2026, además de una inversión que pasará de 1 Billón de dólares en el 2019 a 30 Billones para el 2026.

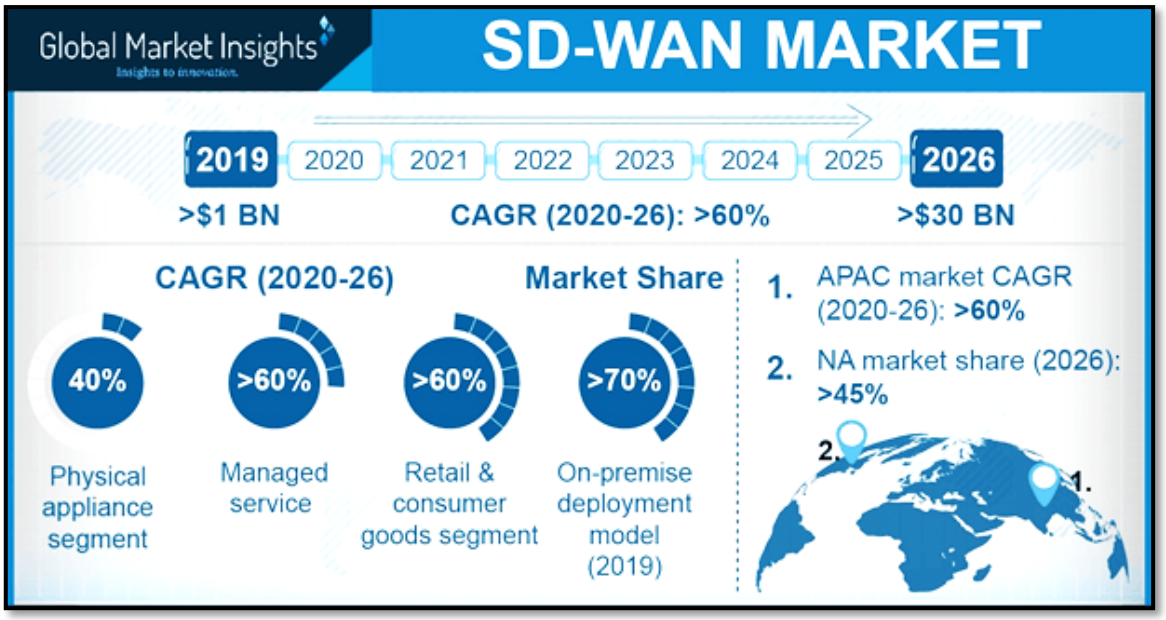


Figura 3-31 Mercado y crecimiento mundial de SD-WAN
Recuperado de (Global Market Insights, 2020)

Una de las regiones que más necesidad de SD-WAN con dispositivos virtuales tendrá es Asia. La *Fig. 3-32* muestra esa tendencia para China.

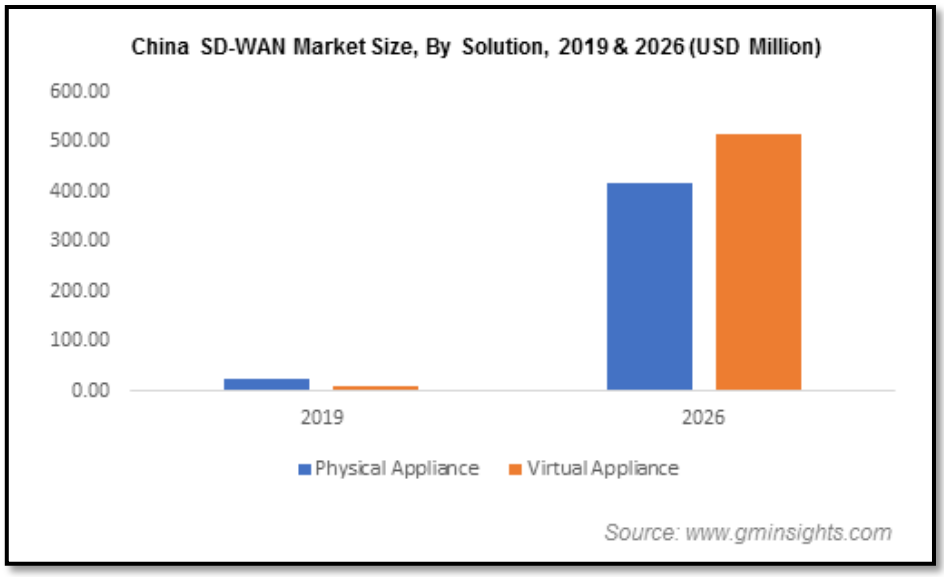


Figura 3-32 Mercado Chino para SD-WAN – proyecciones al 2026
Recuperado de (Global Market Insights, 2020)

En la región de América del Sur, el crecimiento es similar al global en cuanto a adopción SD-WAN, sin embargo, no se confía en entornos basados completamente en *Cloud* o IaaS (*Infrastructure-as-a-Service*), por ello, se tienen implementaciones en premisas casi en su mayoría. Brasil es un caso que muestra esta tendencia latinoamericana (*Fig, 3-33*).

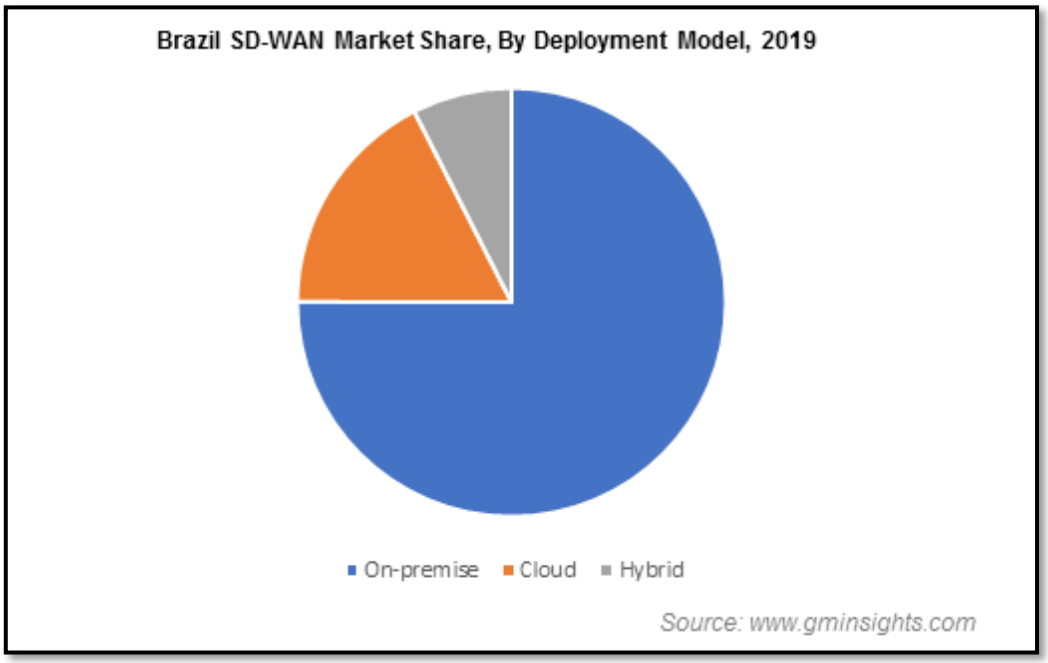


Figura 3-33 Mercado Brasileño para SD-WAN – Implementaciones en premisa/cloud/híbridas
Recuperado de (Global Market Insights, 2020)

Estas tendencias comprueban que SD-WAN es la tecnología a usar para interconexión de sedes empresariales, así como para sostener servicios necesarios para el desarrollo de una sociedad, incluyendo a soluciones de conectividad inalámbrica 4G/5G.

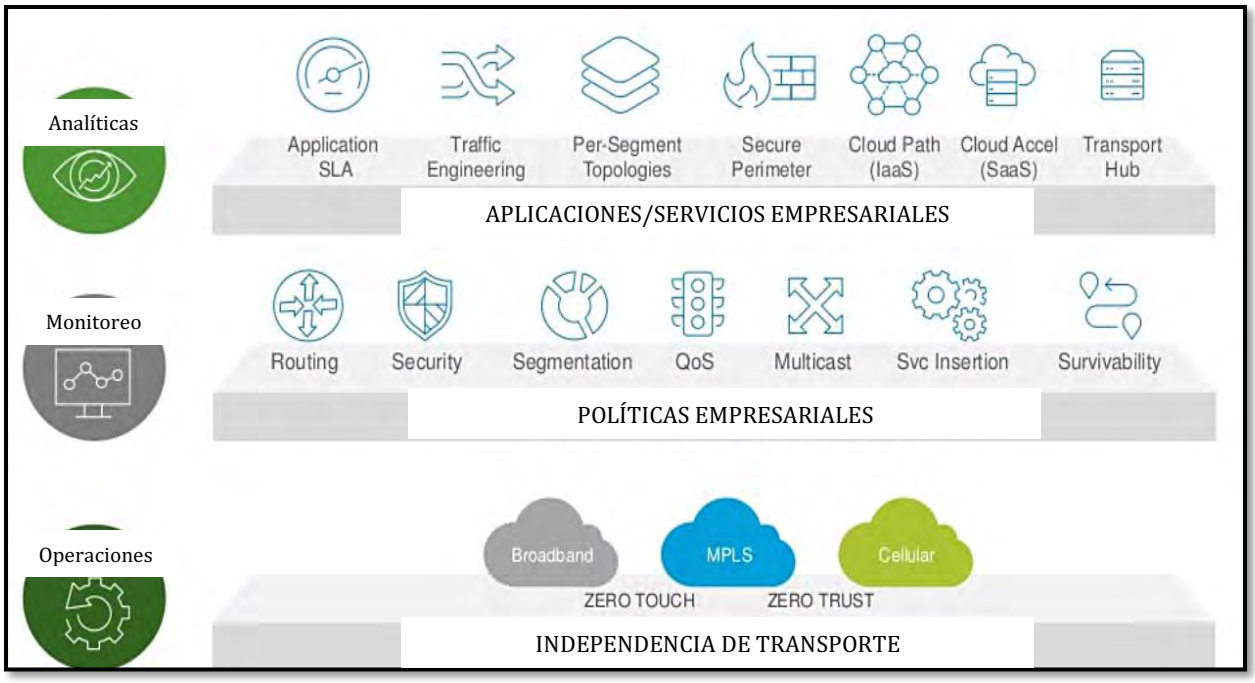


Figura 3-34 Esquematización de una WAN moderna: SD-WAN Basado en (Barton, 2017)

Según el Cuadrante de *Gartner*, las empresas líderes en implementaciones SD-WAN (*WAN Edge* Infraestructure 2020) son VMWare, Fortinet, Versa Networks, Cisco (Viptela), Silver Peak y Palo Alto.

3.3.3 SD-WAN: Componentes de la Implementación tipo Viptela

En redes tradicionales, todos los planos estructurales (control, datos y administración) de los equipos de red de diversas funciones y factores de forma se implementan individual e independientemente en cada dispositivo: *routers, switches,* equipos de seguridad, entre otros.

Para configurar ese tipo de infraestructura existe CLI (*Command Line Inteface*) para ingresar comandos que programan el CPU para activar/desactivar interfaces en función de lo que se desea en la red.

A medida que las redes se vuelven más y más grandes, se requiere de más intervención por parte de quien las administre, lo que puede generar más puntos de falla debido a la complejidad que se genera.

Otro factor por considerar es la constante actualización de las tablas de enrutamiento en *routers* en caso de alguna modificación en la red, impactando de esa forma el desempeño y escalabilidad a medida que las infraestructuras crecen y escalan.

Por ello, las soluciones SD-WAN se basan en infraestructuras distribuidas bajo el paradigma SDN tal como se aprecia en la *Fig. 3-35*, la cual muestra un esquema de SD-WAN tipo Viptela.

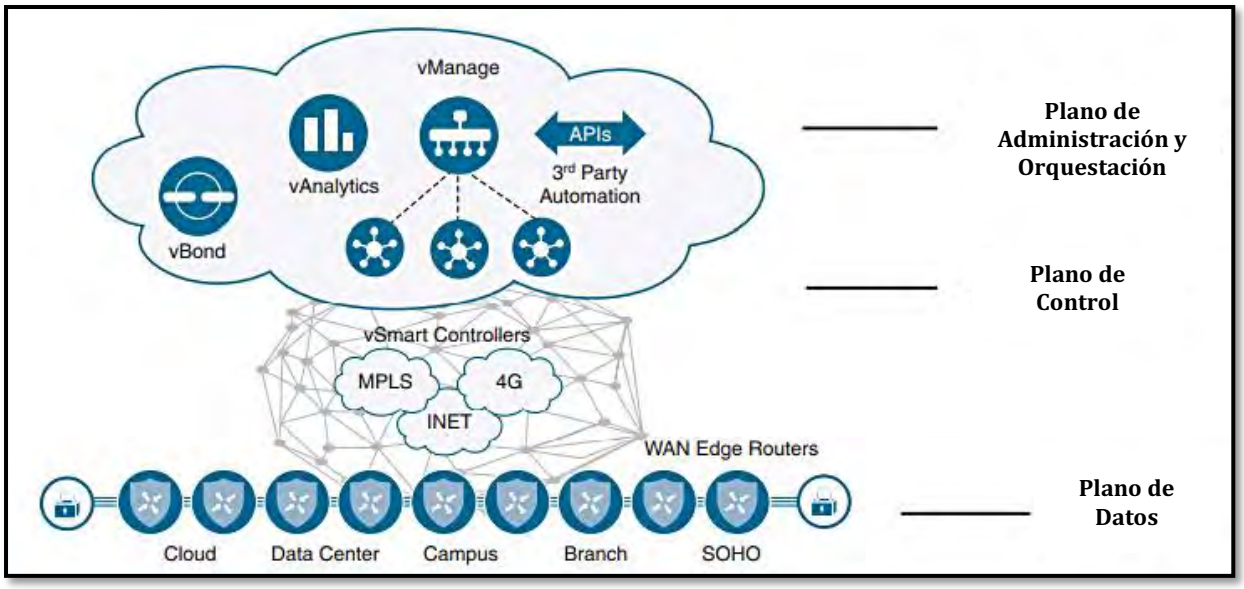


Figura 3-35 Esquema de SD-WAN Viptela
Recuperado de (Gooley, Yanch, Schuemann, & Curran, 2021)

Al contar con una arquitectura distribuida, el Plano de Control es el encargado de conocer todas las rutas de la red, liberando de la necesidad que cada equipo calcule su tabla de enrutamiento, pues este proceso solo se realiza una vez en el Controlador, consiguiendo así una vista completa de la topología, control centralizado y disminución de sobrecarga administrativa.

Plano de Datos de SD-WAN Viptela

En el ***Cap. 2 Fundamentos de las Redes Definidas por Software***, se explicó que el Plano de Datos hace referencia a los elementos físicos que permiten el envío de tráfico en una red, entre ellos, las interfaces, medios y conectores.

Para SD-WAN Viptela, el plano de datos hace referencia a los bordes de la WAN (*WAN Edges*), pudiendo implementarse mediante **vEdge *routers*** o Cisco IOS-XE SD-WAN *Routers*.

El plano de datos para la solución SD-WAN Viptela permite soportar el *Overlay* generado, así como apegándose a la definición tradicional de SDN, es el plano que envía tráfico de red. Cada *router* que forme parte de SD-WAN establecerá conexiones en este plano mediante túneles seguros bajo demanda de tipo IPSec.

La segmentación intrínseca del plano de Datos en SD-WAN Viptela se debe a la implementación del RFC 4023[52] (MPLS *Encapsulation in IP or GRE* o también llamado MPLS-in-IP), lo que implica la separación en varias instancias independientes en el Plano de Datos según los requerimientos del negocio. Las VPNs/túneles generados están completamente aislados uno de otro, a menos que una política permita su comunicación. Todas estas VPNs se encriptan a través de un encabezado IPSec, concepto análogo a las VRFs instanciadas en entornos MPLS-LDP para separar tráfico y virtualizar *routers*. La *Fig. 3-36* muestra ello:

[52] MPLS-in-IP/GRE: https://tools.ietf.org/html/rfc4023

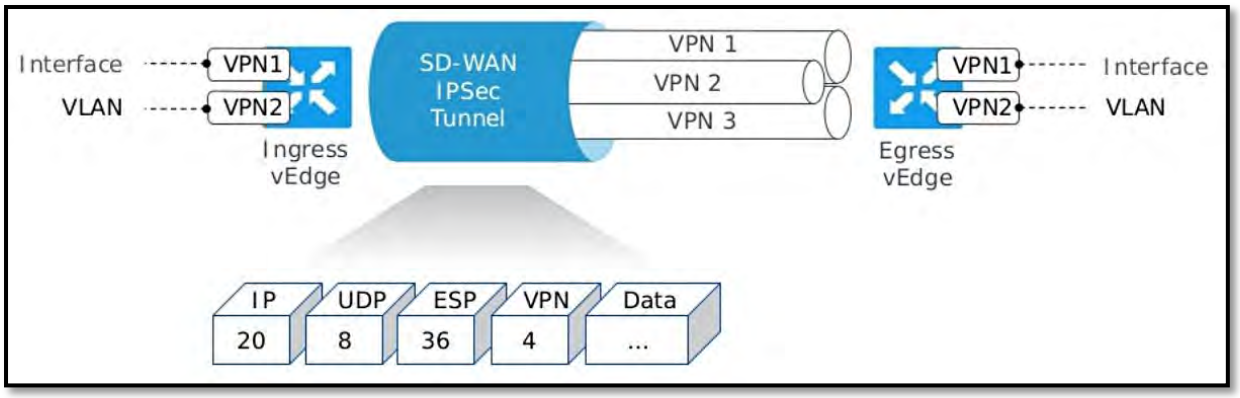


Figura 3-36 Segmentación en SD-WAN Viptela: RFC4023 e IPSec
Recuperado de (Barton, 2017)

En esta solución se utiliza *Bidirectional Forwarding Detection* (BDF) dentro del túnel IPSec para verificar parámetros como pérdida, *jitter* y retraso al aplicar alguna política.

Según (Gooley, Yanch, Schuemann, & Curran, 2021), existen tres tipos de VPN en SD-WAN Viptela para lograr segmentación coherente (*Fig. 3-37*):

- VPN de Servicio: VPN empleada para el envío de tráfico de usuario. Estas VPNs establecen el *Overlay* y terminan en el lado de la LAN. El ID que poseen va de 1 a 511.
- VPN de Transporte: Permiten la generación del *Underlay* o establecimiento de la infraestructura física. Normalmente se refiere como VPN 0 y se la denomina *WAN VPN*.
- VPN de Administración: Interfaz para configuración fuera de banda (OOB – *Out-of-Band*). Tiene un valor de VPN de 512.

En cuanto al control de acceso e ingreso de *WAN Edges* a la infraestructura SD-WAN, se emplean mecanismos tipo *Zero-Touch Provisioning* (ZTP), DNS Security, protección para dispositivos finales, NGFW, *Zero-Trust*, etc.

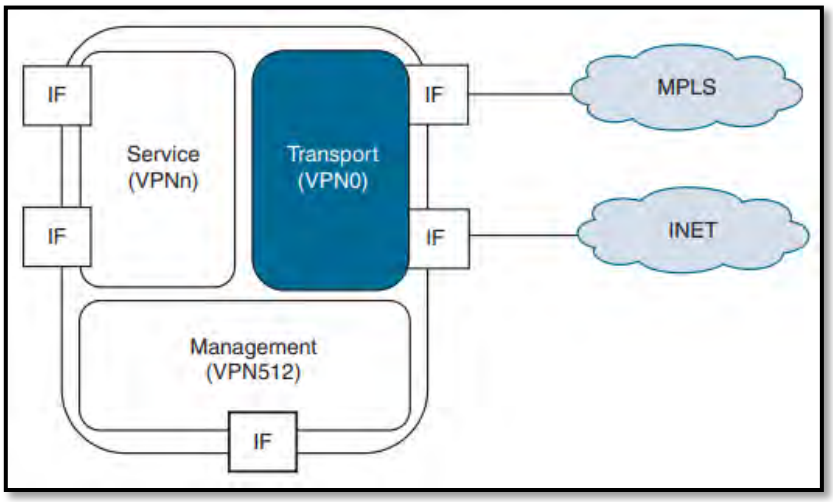


Figura 3-37 Tipos de VPNs en SD-WAN Viptela
Recuperado de (Gooley, Yanch, Schuemann, & Curran, 2021)

Plano de Administración de SD-WAN Viptela

La solución SD-WAN Viptela introduce el concepto de ***vManage***, el cual es un tipo de NMS (*Network Management System*) para administrar la infraestructura SD-WAN.

La función de *vManage* es la de ser el centro de monitoreo, gestión, aprovisionamiento y creación de políticas de SD-WAN, por lo que es posible tener una comunicación vía API usando RestCONF o NetCONF (ver ***Anexo C: NETCONF en la Práctica*).**

El plano de administración de SD-WAN Viptela es muy escalable, proveyendo redundancia de gestión en *Clústers* de *vManage*, soportando hasta seis mil (6000) *WAN Edges*.

Cada *WAN Edge* forma un único e individual plano de administración con *vManage*.

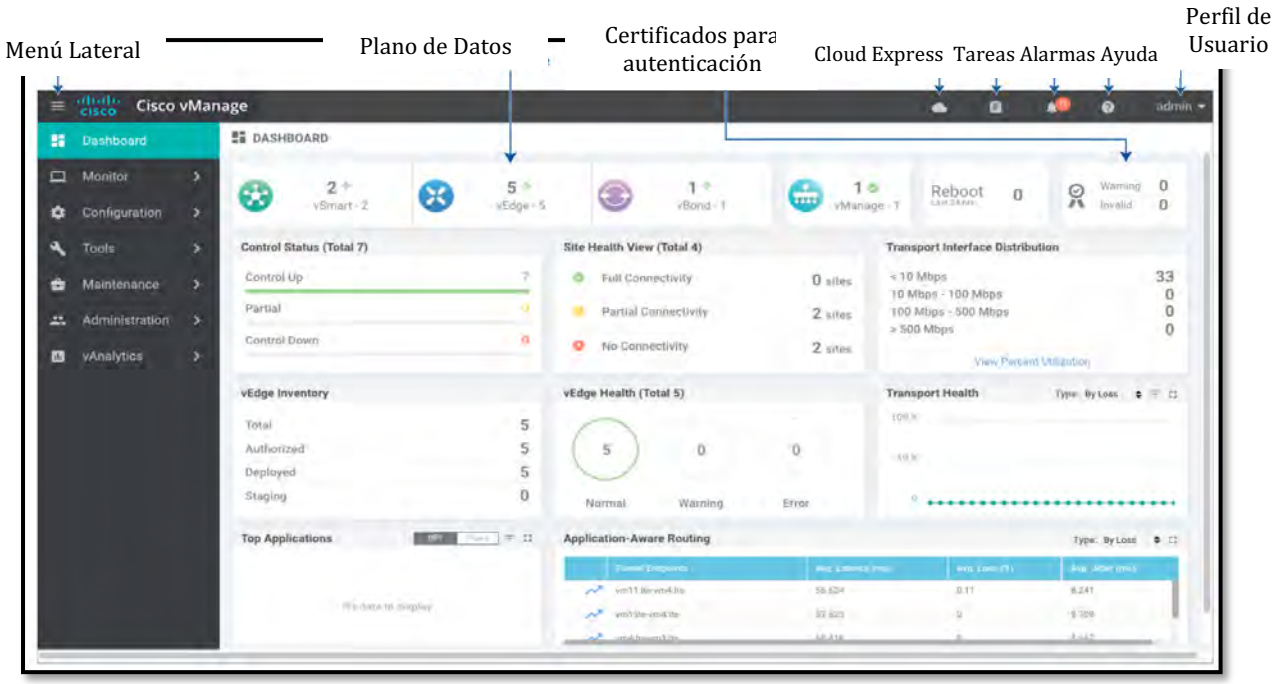


Figura 3-38 Cisco vManage Dashboard – Plano de administración de SD-WAN Viptela Recuperado de (Cisco Systems)

Etapas de monitoreo y solución de inconvenientes deben ejecutarse en el Plano de Monitoreo de SD-WAN, ya que esta solución se incluye también el denominado ***vAnalytics*** con el fin de analizar el tráfico de forma predictiva mediante técnicas de *Machine Learning*. Muchos fabricantes implementan este concepto con licencias adicionales en la solución que no suele implementarse por defecto.

Plano de Control de SD-WAN Viptela

El dispositivo que se encarga de proveer las funciones del plano de control para SD-WAN Viptela se denomina ***vSmart***, el cual se considera como el "Cerebro" de esta solución novedosa.

Un *vSmart* tiene la responsabilidad de la implementación de políticas del plano de control basadas en la creación de VPNs dinámicas, así como manejar la parte de seguridad y encriptación del *Fabric* de SD-WAN.

Los entornos basados en paradigmas SDN se han creado para dotar de escalabilidad y a su vez, sean implementaciones sencillas de poner en marcha. El entorno de SD-WAN es muy flexible,

permitiendo WANs muy grandes gracias a la separación del Plano de Control, del de Datos y de Administración.

En SD-WAN, solamente los *vSmarts* aprenden sobre el enrutamiento necesario para enviar tráfico en la red, siendo estos equipos quienes calculan y determinan la tabla de enrutamiento y luego la distribuyen a los *WAN Edges*; disminuyendo la complejidad y dando flexibilidad al entorno de enrutamiento, ya que es posible escalar a niveles masivos.

El protocolo empleado para compartir la información de enrutamiento por parte de los *vSmart* se denominó ***Overlay Management Protocol*** **(OMP),** encargándose también del enrutamiento, por lo que también es considerado por varias investigaciones como un protocolo de enrutamiento de nueva generación, el cual además de servir para encontrar el mejor camino a un destino, de igual manera maneja las actualizaciones y comunicaciones en el *Overlay* de SD-WAN Viptela. Debido a la importancia que tiene, las actualizaciones enviadas por OMP entre *vSmart* y *WAN Edges* se envían en un túnel seguro, ya sea con IPSec o DTLS/TLS.

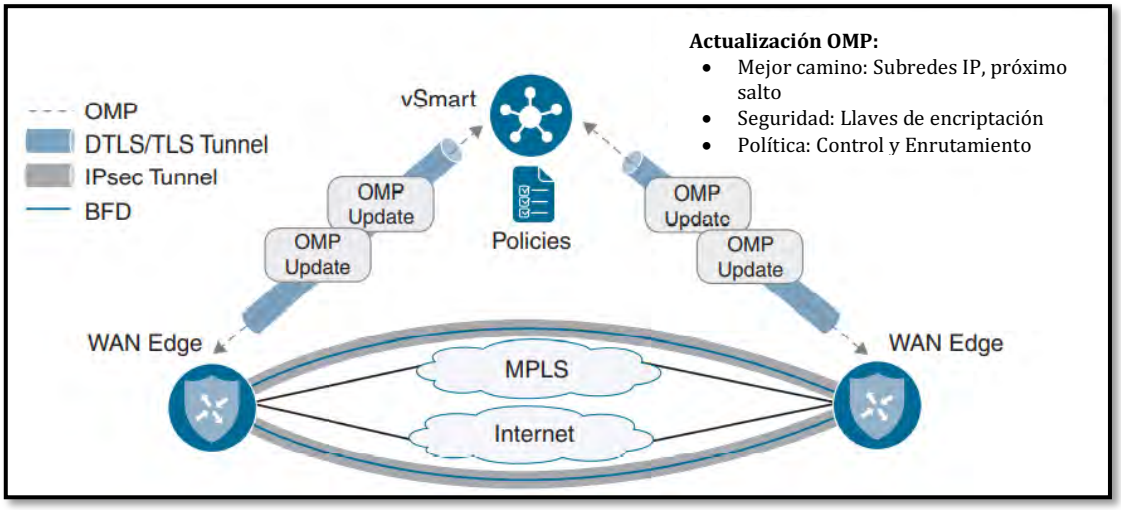


Figura 3-39 Comunicación Plano de Control y Plano de Datos en SD-WAN: OMP
Basado en (Gooley, Yanch, Schuemann, & Curran, 2021)

Si bien la imagen mostrada en la *Fig. 3-39* suele ser la más común, es posible que la política sea generada en el plano de administración (*vManage*), en ese caso, la política se distribuye a los *vSmart* mediante NetCONF y luego los *vSmart* distribuyen dicha política a los *WAN Edges* mediante OMP. Un *WAN Edge* puede conectarse hasta con tres *vSmart*.

En entornos Cisco SD-WAN, OMP está habilitado por defecto tanto en los *vSmart* como en los *WAN Edge* tipo *vEdges*. Tan pronto como esos equipos son autenticados y construyen el túnel seguro con DTLS/TLS (al aceptarse los certificados de autenticación), el protocolo OMP establece una paridad y comienza el intercambio de información de enrutamiento.

Para OMP según (Cisco Systems, 2019), existen tres tipos de rutas:

- Rutas OMP (*vRoutes*): Rutas aprendidas por OMP en el SD-WAN *fabric* para permitir la comunicación de las sedes detrás de un *WAN Edge*. Los prefijos de red que un *WAN Edge* desea comunicar, es enviado a los *vSmart*, de forma similar a una actualización de enrutamiento tradicional. OMP puede anunciar rutas directamente conectadas, rutas estáticas y rutas redistribuidas de protocolos IGP como OSPF, EIGRP y BGP.

Los atributos que en Rutas OMP se pueden encontrar están:

- ✓ TLOC: Identificador del próximo salto de la Ruta OMP
- ✓ Origin: Identifica el origen de la ruta, por ejemplo, si proviene de un IGP.
- ✓ Originator: Identifica la IP del sistema de quien anuncia la ruta, es decir del vecino en OMP
- ✓ OMP *Preference*: Atributo que puede ser usado para modificar la selección de la mejor ruta por parte de OMP. Opera de forma similar a *Local Preference* en BGP.
- ✓ *Service*: Equipo de red, como un FW, a ser identificado dentro de SD-WAN.
- ✓ *Site-ID*: Símil a un ASN (*Autonomous System Number*) en BGP. Se suele usar para orquestación de tráfico. Todos los sitios deben tener un *Site-ID* único para prevenir bucles de enrutamiento.
- ✓ Tag: Tag para políticas de enrutamiento y selección de caminos.
- ✓ VPN: Identificador de la VPN desde la cual la ruta es anunciada. Sirve en los procesos de segmentación

- Rutas *Transport Location* (TLOC): Son identificadores que unen una *vRoute* a una ubicación o sede física. TLOC es una dirección IP que es conocida y alcanzable desde la red *Underlay*. Las rutas OMP resuelven su próximo salto a una ruta TLOC. Este atributo es muy similar al atributo *Next_Hop* de BGP.

  Dentro de las rutas TLOC, se encuentran los siguientes atributos:

  - ✓ Dirección IP del sistema: Es un símil al *Router-ID* en OSPF. Es una dirección IP que no necesita estar en la tabla de enrutamiento, pero debe ser única dentro de todos los *WAN Edges*, por lo que es una manera de identificar unívocamente a los *WAN Edges.*

    Existen dos tipos de IPs del sistema:

    - Dirección Privada TLOC: Este atributo contiene la dirección IP privada derivada de una interfaz física de un *WAN Edge.*
    - Dirección Pública TLOC: Debido a que los *WAN Edge* construyen sus conexiones en el plano de control se notifica vía STUN (*Session Traversal Utilities for NAT*) definido en el RFC 5389, pues normalmente en un entorno IPv4 estarán las sedes empresariales detrás de NAT. Este atributo contiene una dirección IP pública y enrutable en Internet.
  - ✓ Carrier: El tipo de conexión, sea pública o privada.
  - ✓ Color: Forma de marcar una conexión WAN específica en el plano de datos o plano físico, interfaz que puede modificar una política de enrutamiento. Entre las opciones habilitadas al momento de escribir la presente tesis están: *biz-internet, bronze, custom1-3, default, gold, green, lte, metro-ethernet, mpls, private1-6, public-internet, red, 3g, silver.*
  - ✓ Tipo de Encapsulación: Identifica el tipo de encapsulación usada en el túnel del plano de datos: IPSec o GRE.
  - ✓ *Preference* y *Weight*: Atributos para seleccionar el mejor camino según políticas de enrutamiento. Su valor por defecto es 0.

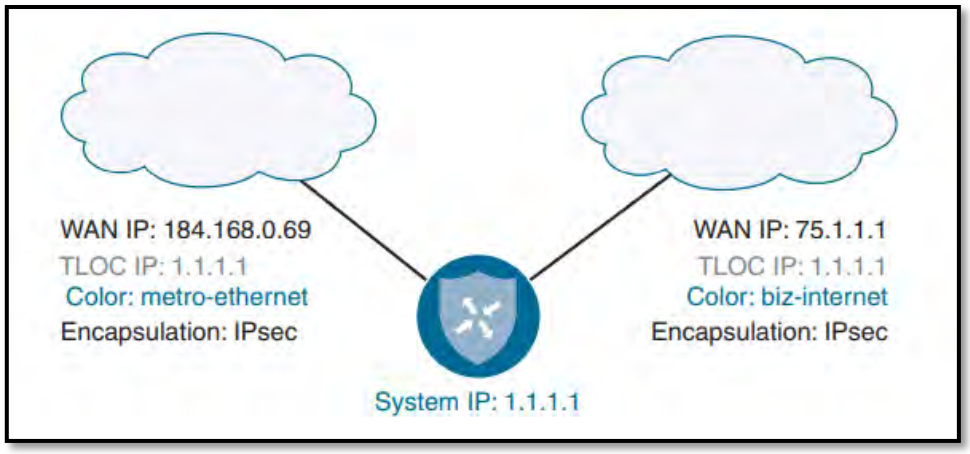


Figura 3-40 Rutas OMP TLOC en SD-WAN Viptela
Recuperado de (Gooley, Yanch, Schuemann, & Curran, 2021)

- Rutas de Servicio: Identifica un dispositivo de red en particular al *Overlay* de SD-WAN. Dentro de esos dispositivos están *firewalls*, IPS o cualquier otro dispositivo que pueda procesar tráfico de red y por algún motivo, sea necesario que el tráfico atraviese ese dispositivo. Esto se conoce como *Service-Chaining*.

La *Fig. 3-41* muestra un ejemplo con las distintas Rutas en OMP en un entorno SD-WAN que interconecta sedes por detrás de los *WAN Edges*.

En ***4.2 PoC de SD-WAN en EVE-ng*** se realiza una Prueba de Concepto sobre Viptela, demostrando su factibilidad de uso e implementación, llevando la parte conceptual a la práctica.

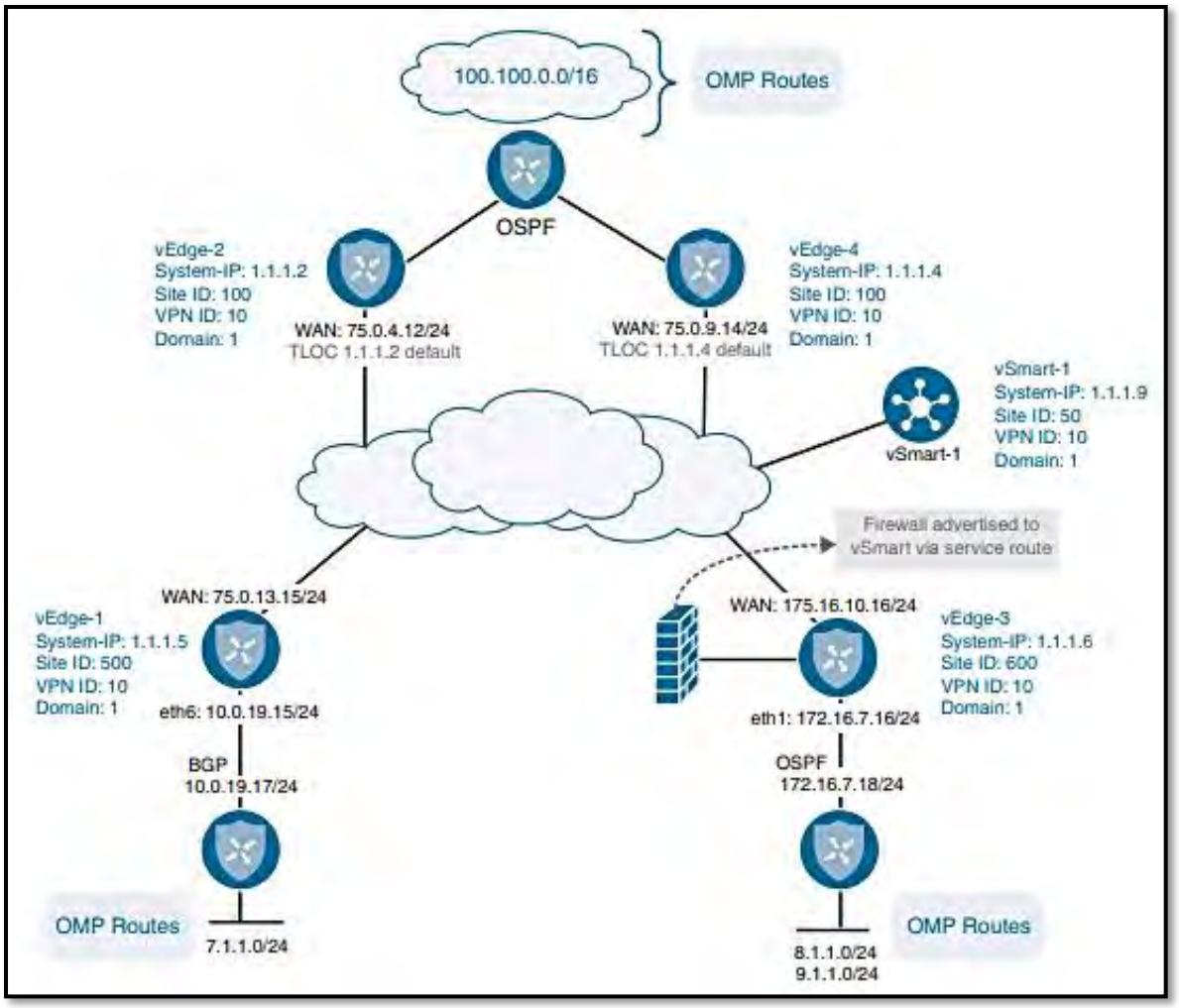


Figura 3-41 Rutas OMP en SD-WAN: vRoutes, TLOC y Rutas de servicio
Recuperado de (Gooley, Yanch, Schuemann, & Curran, 2021)

3.4 Redes Híbridas SDN: Relación del *Networking* Tradicional con SD-WAN

El *networking* tradicional tuvo que romper con sus paradigmas de redes clásicas y entrar al ecosistema de SDN y la programabilidad.

Un ejemplo de la convivencia entre lo tradicional y la nueva generación de redes se pudo apreciar en la *Fig. 3-41* que muestra una topología de SD-WAN Viptela, donde conceptos tradicionales de enrutamiento y conmutación se mezclan con conceptos de Plano de Control, Datos y administración desagregados, dotando así de flexibilidad, total visibilidad y rapidez de implementación, sin dejar de lado la simplicidad.

Es más, en entornos SD-WAN, las mejores características de ciertos protocolos tradicionales como BGP, han servido de punto de partida para el desarrollo de nuevos dispositivos, roles y reglas de comunicación, los *vSmart* del plano de Control son una muestra de ello.

Los *vSmart* funcionan de forma similar al concepto de *route-reflector* de BGP, ya que son el punto de recepción de información de enrutamiento proveniente de los *WAN Edges* y pueden aplicar políticas antes de anunciar esos cambios en la red. No hay que olvidar que las políticas se definen y crean en el plano de Administración (*vManage*) y los *vSmart* aplicar dichas políticas en el *fabric* de SD-WAN.

En tecnologías WAN antecesoras a SD-WAN (MPLS-L3VPN, DMVPN o iWAN), asegurar la red requería de procesos intensos para los CPUs de cada dispositivo como el manejo de las llaves de encriptación quizá empleando *frameworks* tipo ISAKMP/IKE en la conocida fase 1 de IPSec (Fase 1 de IKE). Todo ese procesamiento, es ahora responsabilidad de los *vSmart,* incluso mejorando el mecanismo y manejo de llaves en un entorno centralizado.

Quizá una de las preguntas más obvias en entornos SD-WAN, es qué ocurre si por algún motivo, se pierde conexión con el Plano de Control. La respuesta es simple, el tráfico sigue fluyendo por el Plano de Datos con el último estado de la tabla de enrutamiento, en el caso de SD-WAN Viptela, por 12 horas luego de la pérdida de conectividad entre *WAN Edge* y *vSmart*. Por motivos de redundancia, se sugiere tener pares de *vSmart* dispersos en varios lugares de la infraestructura, formando un *full-mesh* de sesiones OMP entre ellos para mantener sincronización.

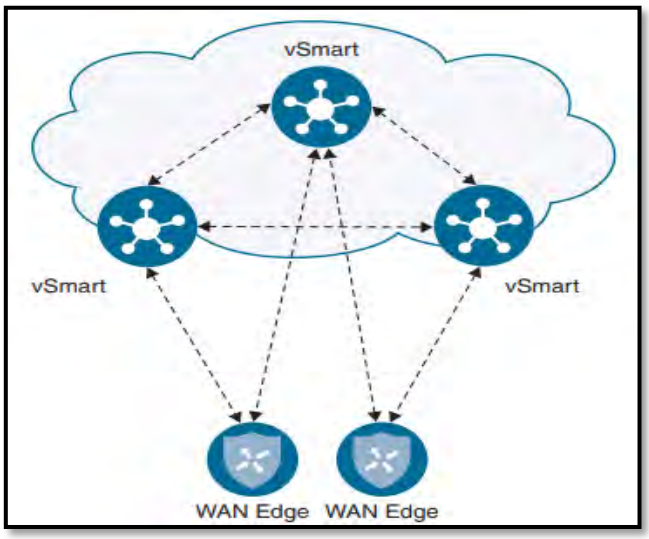


Figura 3-42 Full-Mesh de Sesiones OMP entre vSmarts redundantes
Recuperado de (Gooley, Yanch, Schuemann, & Curran, 2021)

3.4.1 Plano de Orquestación en SD-WAN: Plano faltante de la Red Tradicional

Es quizá el plano más importante del entorno SD-WAN moderno, pues cumple funciones que permiten integrar todos los componentes de SD-WAN, es el punto de autenticación para el ingreso de equipos al *fabric* SD-WAN y permite la mejora continua en base al contexto empresarial.

Para SD-WAN Viptela, el equipo encargado del Plano de Orquestación se llama ***vBond*.**

Cuando un *WAN Edge* desea ingresar a la red SD-WAN, lo único que conoce es la dirección IP del *vBond*, recibiendo esa información ya sea por procesos *Zero-Touch*, *Plug-and-Play*, Configuración *Bootstrap* o por configuración manual.

Los *WAN Edge* intentarán formar una conexión temporal con el *vBond*, pues una vez que la conectividad está activa con el Plano de Control y Administración, *vSmart* y *vManage* respectivamente, la conexión con el *vBond* desaparece.

Sin embargo, en el momento que se intenta formar esa conexión temporal, los *WAN Edges* entran en un proceso de autenticación en el que si es exitoso, se forma un túnel DTLS. Luego, el *vBond* distribuye la información de conectividad para los *vSmart* y *vManage* a los *WAN Edges.*

Una de las mejores funcionalidades presente en el plano de orquestación, es la capacidad que tienen los *vBond* de aceptar peticiones de ingreso al *fabric* SD-WAN incluso si los *WAN Edges* están detrás de un equipo que haga NAT. En ese caso, los *vBond* operan como un Servidor STUN (NAT *Traversal*) y los *WAN Edges* como clientes STUN, donde los *vBond* deben tener direccionamiento público.

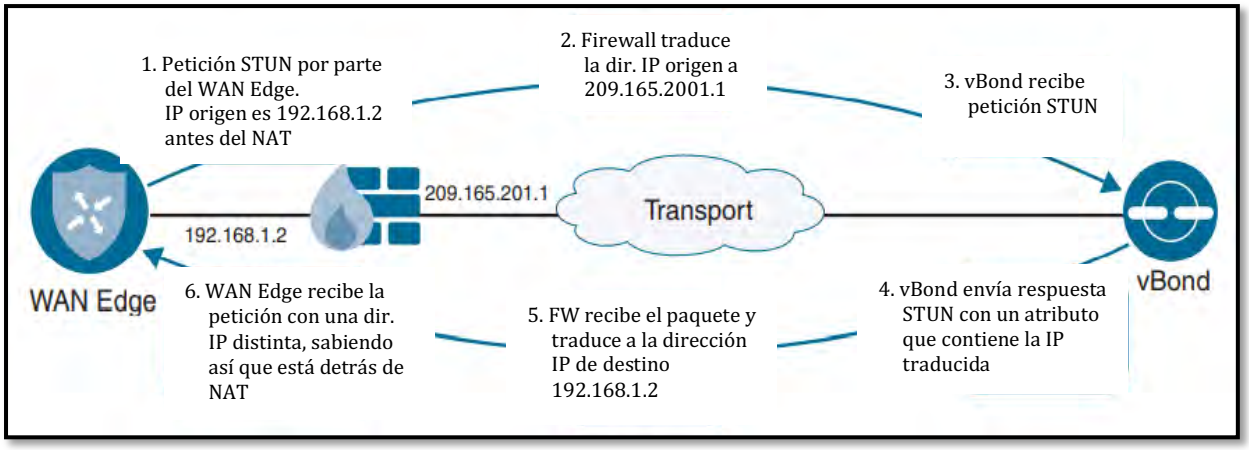


Figura 3-43 Proceso de ingreso de un WAN Edge detrás de NAT a un vBond (Proceso STUN)
Basado en (Gooley, Yanch, Schuemann, & Curran, 2021)

3.4.2 Generación de Túnel DTLS para el Plano de Control en SD-WAN

En *networking* tradicional, el plano de control únicamente se enfoca en cómo enviar tráfico en la red mediante protocolos de enrutamiento y eligiendo la mejor ruta a un determinado destino. Si se desea dar seguridad a los protocolos de enrutamiento, es un proceso manual e intensivo en cuanto a consumo de recursos, que incluso puede dejar sin funcionamiento a la red por un tiempo determinado, esto ya que la seguridad en redes se consideró un proceso añadido más que una característica intrínseca, siendo una debilidad de las redes tradicionales.

Por otro lado, en SD-WAN, la seguridad es una característica fundamental. Los túneles de Planos de Control están encriptados y autenticados mediante DTLS/TLS.

Los túneles DTLS/TLS tipo una vía, son mantenidos por todos los componentes de SD-WAN: *vBond, vSmart, WAN Edges* y *vManage*) a través de certificados SSL. En la negociación del establecimiento de los túneles, cada dispositivo validará que el certificado recibido esté firmado por una entidad certificadora raíz confiable (el *vManage* puede tomar esa función).

La comunicación DTLS ocurre sobre el puerto UDP 12346, mientras TLS emplea TCP, por defecto se elige DTLS. La *Fig. 3-44* muestra el proceso de autenticación con DTLS.

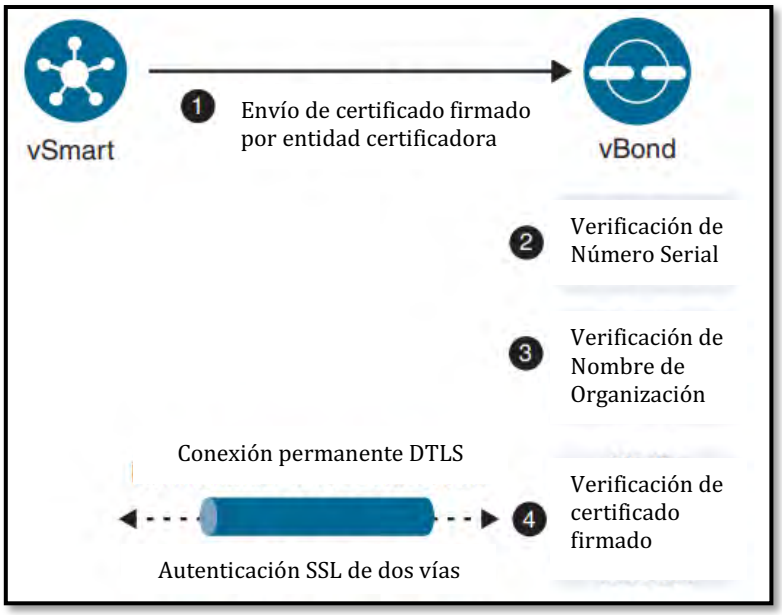


Figura 3-44 Proceso de autenticación DTLS entre vSmart - vBond Basado en (Gooley, Yanch, Schuemann, & Curran, 2021)

Una vez que los túneles DTLS del plano de control están activos, otros protocolos como OMP y NetCONF pueden también usarlos.

3.4.3 Selección de Mejor Camino en SD-WAN

Algo que poseen en común las redes tradicionales con SD-WAN Viptela, es el proceso de selección del mejor camino, pues es similar al proceso que se lleva a cabo con BGP, pero que para SD-WAN se realiza con OMP.

OMP debe elegir el mejor camino a todos los destinos disponibles, pero evitando bucles de enrutamiento. Se debe mencionar que un *WAN Edge* instalará en su tabla de enrutamiento una ruta OMP solo si su TLOC o próximo salto es válido. Se considera un TLOC válido si existe una sesión bidireccional asociada a ese TLOC.

A medida que los *WAN Edges* anuncian rutas OMP a los *vSmart*, estos últimos realizan el proceso de selección del mejor camino y lo anuncian a los *WAN Edges*. Esta selección puede ser influenciada por una política de enrutamiento.

La selección del mejor camino ocurre con este proceso:

1. Existencia de una ruta OMP válida, es decir si existe un TLOC válido. Este punto es similar a BGP, donde el atributo *Next_Hop* debe ser válido.
2. Ruta OMP localmente originada: Desde un *WAN Edge*, se prefiere una ruta originada localmente que una aprendida desde el *vSmart*.
3. Más baja Distancia Administrativa: Si múltiples rutas para el mismo destino se reciben, se selecciona la que tenga menos AD. OMP tiene una AD de 250 en *vEdges* y 251 para equipos con Cisco XE-SDWAN.
4. Mayor preferencia OMP
5. Mayor preferencia TLOC

6. Preferir el origen de la ruta en base a este orde: Directamente conectadas, estáticas, eBGP, EIGRP, OSPF intra-área, OSPF inter-área, OSPF externo, EIGRP externo, iBGP, desconocido
7. Menor métrica de origen
8. Mayor *System-ID*
9. Mayor dirección privada TLOC.

vSmarts puede anunciar hasta 16 rutas del mismo costo, por defecto, 4.

En los próximos capítulos de la presente tesis, se plantearán entornos de prueba de concepto y factibilidad (PoCs) de los protocolos y tecnologías más importantes del contexto de nueva generación de redes y con ello poner en práctica la teoría analizada, investigada y planteada dentro del *Networking* moderno:

- *Segment-Routing* en un *Fabric* de MPLS
- LISP como base del paradigma *Underlay-Overlay* de SD-Access/SD-Branch
- VXLAN en un entorno de Programabilidad (Parte del ***Anexo D: Ansible para entornos NetDevOps en infraestructuras de Red***)
- PoC de SDN empleando *OpenFlow* como protocolo *Southbound* bajo controladores *OpenDayLight* y ONOS.
- PoC de *Next-Generation* SDN.
- Emulación de SD-WAN Viptela.
- Implementación con equipos físicos de una red prototipo SDN.

Capítulo 4

4. Simulación/Emulación de Redes SDN Híbridas

Los protocolos y tecnologías analizadas en los capítulos anteriores serán puestos a prueba en entornos de emulación de infraestructuras de alto nivel utilizando herramientas de *Hardware* y *Software* de última generación, sustentadas por publicaciones científicas desarrolladas mientras se escribía la presente tesis.

Las fases PoC, así como la verificación de la implementación en equipos reales SDN, dotan de un valor muy relevante a esta investigación, dando un gran paso para la puesta en marcha de este tipo de tecnologías y protocolos en redes empresariales reales, permitiendo así que la Tesis sea un proceso circular investigativo, donde se ha iniciado con la observación de los hechos para fundamentar la hipótesis planteada, se diseñarán experimentos tanto en emulaciones como en equipos, logrando sistematicidad y a su vez objetividad.

4.1 Emulación de Segment-Routing con MPLS de Plano de Datos

Usando como base el artículo científico de Gustavo Salazar, autor de la tesis, publicado en el congreso internacional IEEE UEMCON 2018 desarrollado en la Universidad de Columbia, Nueva York, ganador como mejor paper, titulado *SDN-Ready WAN Networks: Segment-Routing in MPLS-Based Environments* (Salazar Ch., Naranjo, & Marrone, 2018), se planteó la prueba de concepto en EVE-ng del punto *4.1.1. Topología y Emulación de SR-MPLS* (Para ver el proceso de instalación de EVE-ng ver ***Anexo B: Guía de Instalación GNS3-VM, EVE-ng y Mininet***).

4.1.1 Topología y Emulación de SR-MPLS

Tal como se analizó en ***3.1.2 Segment-Routing: Plano de Datos y Plano de Control***, SR puede ser implementando a través de MPLS-LDP como plano de datos, es decir, usando ese concepto, la lista de segmentos es encapsulada en una etiqueta de MPLS, dando lugar a un *fabric* que ofrece transporte de datos sin requerir a protocolos adicionales o mecanismos de señalización como LDP o RSVP para obtener beneficios de ingeniería de tráfico, tal como se indica en (Salazar Ch., Naranjo, & Marrone, 2018) y en (Salazar-Chacón & Reinoso García, 2021).

Para el PoC de SR se plantea una conectividad Matriz-Sucursal mediante un ISP, donde la comunicación entre PE-CE se realiza a través de OSPF.

Entre PEs, se tiene un *underlay* MPLS, estableciendo un túnel L3-VPNv4 con conexión iBGP.

Se usará IS-IS como el IGP dentro de la nube del proveedor. IS-IS necesita una extensión dentro del *address-family* para IPv4 con el fin de soportar SR.

El hecho de que este tipo de SR tenga como *underlay* a MPLS, permite una interoperabilidad mediante un equipo denominado SR *Mapping-Server* en caso de que un nodo sin capacidades de comprender SR exista en el camino.

La topología SR-MPLS emulada se indica en la *Fig. 4-1*, la cual consta de dos tipos de equipos con los siguientes NOS:

- IOL-IOS AdvEnterpriseK9-15.4.1T para los CEs
- Cisco XR-K9 6.0.1 para la Nube del proveedor

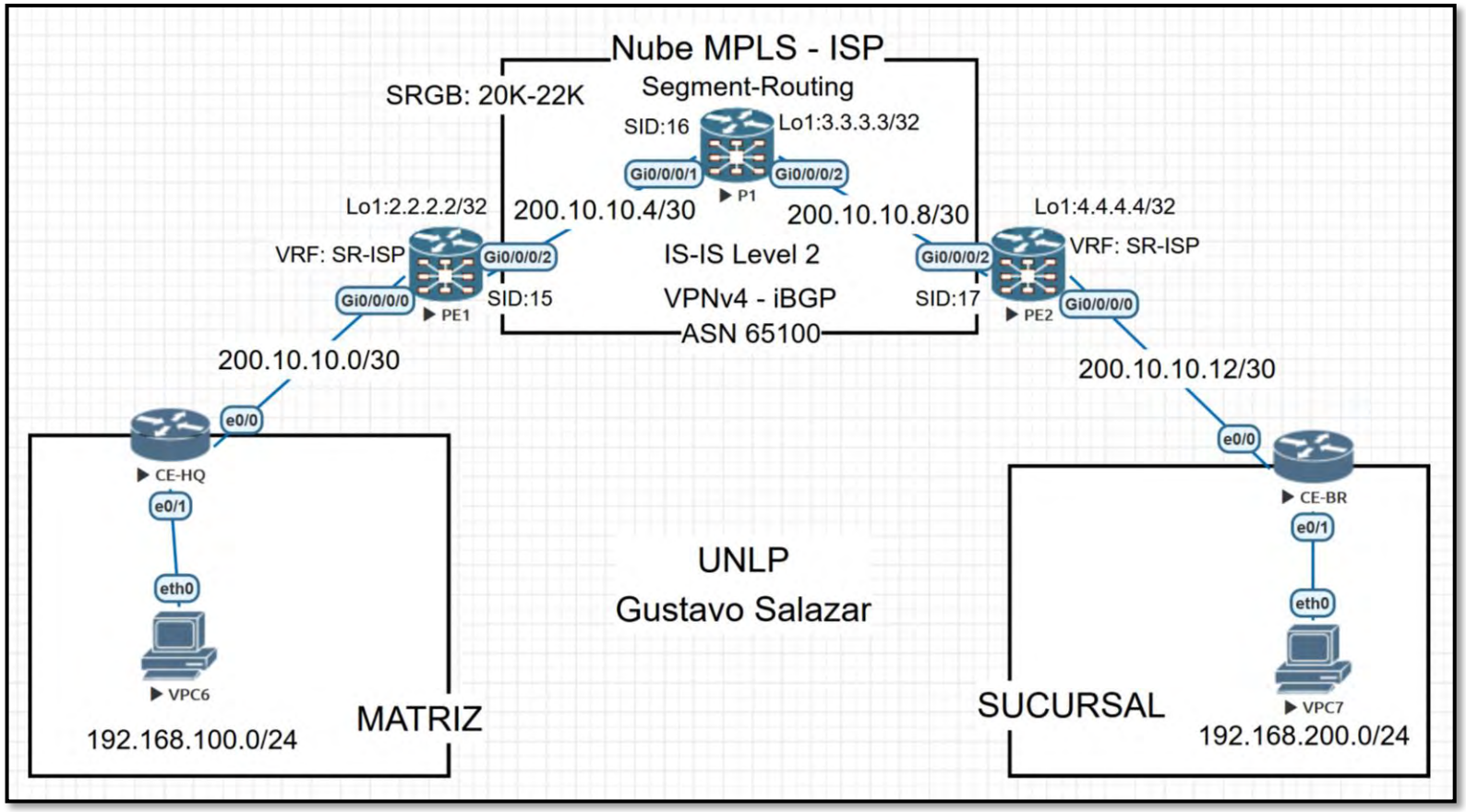


Figura 4-1 Topología PoC – Segment-Routing
Fuente: *Autor*

Conectividad CE-PE

La conectividad CE-PE se da a través de OSPFv2 de área única. Se utilizó una VRF de nombre SR-ISP del lado del PE con el fin de segmentar el tráfico entre diversos clientes.

En el *script* se observa la configuración del IGP mencionado dentro de una VRF en PE2 y la creación de la interfaz *Loopback* como identificador para IS-IS y *Segment-Routing*

```
RP/0/0/CPU0:ios(config)#hostname PE2
RP/0/0/CPU0:ios(config)#
RP/0/0/CPU0:ios(config)#vrf SR-ISP
RP/0/0/CPU0:ios(config-vrf)#address-family ipv4 unicast
RP/0/0/CPU0:ios(config-vrf-af)#import route-target
RP/0/0/CPU0:ios(config-vrf-import-rt)#65100:1
RP/0/0/CPU0:ios(config-vrf-import-rt)#exit
RP/0/0/CPU0:ios(config-vrf-af)#export route-target
RP/0/0/CPU0:ios(config-vrf-export-rt)#65100:1
RP/0/0/CPU0:ios(config-vrf-export-rt)#exit
RP/0/0/CPU0:ios(config-vrf-af)#exit
```

Script 20 Configuración de VRF en PE2 (IOS-XR) para Segmentación de Tráfico en PoC de Segment-Routing
Fuente: *Autor*

```
RP/0/0/CPU0:ios(config)#int lo1
RP/0/0/CPU0:ios(config-if)#ipv4 add 3.3.3.3 255.255.255.255
RP/0/0/CPU0:ios(config-if)#
RP/0/0/CPU0:ios(config-if)#exit
RP/0/0/CPU0:ios(config)#int g0/0/0/0
RP/0/0/CPU0:ios(config-if)#vrf SR-ISP
RP/0/0/CPU0:ios(config-if)#ipv4 add 200.10.10.13 255.255.255.252
RP/0/0/CPU0:ios(config-if)#no shut
RP/0/0/CPU0:ios(config-if)#exit
```

Script 21 Configuración de interfaces en PE2 para PoC de Segment-Routing
Fuente: *Autor*

```
RP/0/0/CPU0:ios(config)#router ospf 3
RP/0/0/CPU0:ios(config-ospf)#vrf SR-ISP
RP/0/0/CPU0:ios(config-ospf-vrf)#area 0
RP/0/0/CPU0:ios(config-ospf-vrf-ar)#interface g0/0/0/0
RP/0/0/CPU0:ios(config-ospf-vrf-ar-if)#exit
RP/0/0/CPU0:ios(config-ospf-vrf-ar)#exit
RP/0/0/CPU0:ios(config-ospf-vrf)#exit
RP/0/0/CPU0:ios(config-ospf)#exit
RP/0/0/CPU0:ios(config)#
RP/0/0/CPU0:ios(config)#commit
Mon Apr 19 18:20:58.045 UTC
```

Script 22 Configuración de OSPFv2 en PE2 para PoC de Segment-Routing (Conexión PE-CE)
Fuente: *Autor*

Si la configuración es correcta y una vez que se ha configurado el CE, se podrá verificar la adyacencia CE-PE.

```
RP/0/0/CPU0:PE2#show ospf vrf SR-ISP neigh
Mon Apr 19 18:47:41.015 UTC

* Indicates MADJ interface
# Indicates Neighbor awaiting BFD session up

Neighbors for OSPF 3, VRF SR-ISP

Neighbor ID     Pri   State           Dead Time   Address         Interface
4.4.4.4         1     FULL/DR         00:00:36    200.10.10.14    GigabitEthernet0/0/0/0
    Neighbor is up for 00:26:31

Total neighbor count: 1
```

Script 23 Verificación de vecindad OSPFv2 entre PE2-CE2 para PoC de Segment-Routing (Conexión PE-CE)
Fuente: *Autor*

Configuración Nube ISP (*Segment-Routing* con plano de datos MPLS)

Dentro de la Nube del proveedor, es necesaria la configuración de un IGP Estado de Enlace capaz de soportar SR, tal como se indicó en ***3.1 Segment-Routing (SR) y Grupo de trabajo SPRING en Redes Tradicionales y SDN, una alternativa a MPLS.***

Para el caso de este PoC, se empleará IS-IS con un **SRGB de 20000 a 22000**. De igual manera, se configuró una relación iBGP entre PEs para la comunicación a través de **L3VPNv4** y circulen dentro de dicho túnel, los prefijos de OSPF de la Matriz y Sucursal. Se estableció también un *Prefix-SID* de índice 15, 16 y 17 para PE1, P y PE2 respectivamente.

```
RP/0/0/CPU0:PE1(config)#
RP/0/0/CPU0:PE1(config)#router isis 1
RP/0/0/CPU0:PE1(config-isis)#is-type level-2-only
RP/0/0/CPU0:PE1(config-isis)#net 49.0001.0000.0015.00
RP/0/0/CPU0:PE1(config-isis)#segment-routing global-block 20000 22000
RP/0/0/CPU0:PE1(config-isis)#address-family ipv4 unicast
RP/0/0/CPU0:PE1(config-isis-af)#metric-style wide
RP/0/0/CPU0:PE1(config-isis-af)#segment-routing mpls sr-prefer
RP/0/0/CPU0:PE1(config-isis-af)#exit
RP/0/0/CPU0:PE1(config-isis)#int lo1
RP/0/0/CPU0:PE1(config-isis-if)#address-family ipv4 unicast
RP/0/0/CPU0:PE1(config-isis-if-af)#prefix-sid index 15
RP/0/0/CPU0:PE1(config-isis-if-af)#exit
RP/0/0/CPU0:PE1(config-isis-if)#exit
RP/0/0/CPU0:PE1(config-isis)#interface gi0/0/0/2
RP/0/0/CPU0:PE1(config-isis-if)#address-family ipv4 unicast
RP/0/0/CPU0:PE1(config-isis-if-af)#exit
RP/0/0/CPU0:PE1(config-isis-if)#exit
RP/0/0/CPU0:PE1(config-isis)#exit
RP/0/0/CPU0:PE1(config)#commit
```

Script 24 Configuración de IS-IS y SR en la Nube del proveedor para PoC de Segment-Routing (Conexión PE-P-PE)
Fuente: *Autor*

La tabla de enrutamiento debe mostrar conocimiento de todas las interfaces *Loopback* dentro de la nube del proveedor aprendidas por IS-IS, tal como se observa en el *Script 25* para PE1.

```
RP/0/0/CPU0:PE1#show ip route
Mon Apr 19 20:12:55.475 UTC

Codes: C - connected, S - static, R - RIP, B - BGP, (>) - Diversion path
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, su - IS-IS summary null, * - candidate default
       U - per-user static route, o - ODR, L - local, G  - DAGR, l - LISP
       A - access/subscriber, a - Application route
       M - mobile route, r - RPL, (!) - FRR Backup path

Gateway of last resort is not set

L    2.2.2.2/32 is directly connected, 02:09:10, Loopback1
i L2 3.3.3.3/32 [115/20] via 200.10.10.6, 00:11:07, GigabitEthernet0/0/0/2
i L2 4.4.4.4/32 [115/30] via 200.10.10.6, 00:00:24, GigabitEthernet0/0/0/2
C    200.10.10.4/30 is directly connected, 00:16:41, GigabitEthernet0/0/0/2
L    200.10.10.5/32 is directly connected, 00:16:41, GigabitEthernet0/0/0/2
i L2 200.10.10.8/30 [115/20] via 200.10.10.6, 00:11:07, GigabitEthernet0/0/0/2
```

Script 25 Verificación de aprendizaje de ruta por IS-IS en la Nube del proveedor
Fuente: *Autor*

El último paso es el establecimiento de iBGP entre PEs (L3VPNv4) y redistribución de ruta aprendidas por OSPF en la VRF SR-ISP. El *Script 26* muestra esa configuración en PE1.

```
RP/0/0/CPU0:PE1(config)#router bgp 65100
RP/0/0/CPU0:PE1(config-bgp)#bgp router-id 2.2.2.2
RP/0/0/CPU0:PE1(config-bgp)#address-family vpnv4 unicast
RP/0/0/CPU0:PE1(config-bgp-af)#neighbor 4.4.4.4
RP/0/0/CPU0:PE1(config-bgp-nbr)#remote-as 65100
RP/0/0/CPU0:PE1(config-bgp-nbr)#update-source lo1
RP/0/0/CPU0:PE1(config-bgp-nbr)#exit
RP/0/0/CPU0:PE1(config-bgp)#vrf SR-ISP
RP/0/0/CPU0:PE1(config-bgp-vrf)#rd 100:100
RP/0/0/CPU0:PE1(config-bgp-vrf)#address-fam ipv4 unicast
RP/0/0/CPU0:PE1(config-bgp-vrf-af)#redistribute ospf 2
RP/0/0/CPU0:PE1(config-bgp-vrf-af)#exit
RP/0/0/CPU0:PE1(config-bgp-vrf)#exit
RP/0/0/CPU0:PE1(config-bgp)#exit
RP/0/0/CPU0:PE1(config)#commit
```

Script 26 Configuración de iBGP – L3VPNv4 en la Nube del proveedor (Conexión PE-PE)
Fuente: *Autor*

Con el fin de que el proceso de enrutamiento fluya, es necesario se aplique una política que permita el paso de tráfico y se formen las adyacencias en la VPNv4 (esto debido al comportamiento del NOS IOS-XR). También, es necesaria la redistribución de rutas de BGP a OSPF para que los CEs tengan conocimiento de las rutas de cada sede. Esta configuración debe darse en los PEs (*Script 27 y Script 28*).

```
RP/0/0/CPU0:PE1(config)#route-policy PASS-ALL
RP/0/0/CPU0:PE1(config-rpl)#pass
RP/0/0/CPU0:PE1(config-rpl)#end-policy
```

Script 27 Configuración de Política de Enrutamiento en IOS-XR para iBGP L3VPNv4
Fuente: *Autor*

```
RP/0/0/CPU0:PE1(config)#router ospf 2
RP/0/0/CPU0:PE1(config-ospf)#vrf SR-ISP
RP/0/0/CPU0:PE1(config-ospf-vrf)#redistribute bgp 65100
RP/0/0/CPU0:PE1(config-ospf-vrf)#
RP/0/0/CPU0:PE1(config-ospf-vrf)#commit
```

Script 28 Redistribución de rutas aprendidas en L3VPNv4 hacia OSPF
Fuente: *Autor*

4.1.2 Resultados de la Emulación de SR

Si todo el proceso de configuración fue realizado correctamente, se puede alcanzar la conectividad de extremo a extremo, es decir, entre Matriz y Sucursal del PoC propuesto en la *Fig. 4-1.*

La configuración de los vPC es como la que se muestra en la *Fig. 4-2*. vPC1 representa un host de la Matriz con dirección IPv4 192.168.100.10/24 y puerta de enlace 192.168.100.1; mientras la vPC2 representa un host de la sucursal con dirección IPv4 192.168.200.10/24 y puerta de enlace 192.168.200.1.

```
VPCS is free software, distributed under the terms of the "BSD" licence.
Source code and license can be found at vpcs.sf.net.
For more information, please visit wiki.freecode.com.cn.
Modified version supporting unetlab by unetlab team

Press '?' to get help.

VPCS> ip 192.168.100.10/24 192.168.100.1
Checking for duplicate address...
PC1 : 192.168.100.10 255.255.255.0 gateway 192.168.100.1
```

Figura 4-2 Configuración de vPC1 para PoC de Segment Routing.
Fuente: *Autor*

La *Fig.4-3* muestra la conectividad de extremo a extremo (ping entre vPC1y vPC2), comprobando de esa manera la factibilidad de uso de *Segment-Routing* en una infraestructura de proveedor sin contar con otros protocolos de etiquetado o señalización adicional (como LDP o RSVP respectivamente), eso se demuestra mediante las *Fig. 4-4 y Fig. 4-5,* visualizando un *traceroute* entre la puerta de enlace de la Matriz (CE-HQ) y el *host* de la sucursal con el marcaje y las operaciones de SR: *Continue* o cambio de etiqueta y *Next* o retirada de etiqueta, así como el uso de un comando en IOS-XR desde PE1 para verificar el contenido de CEF (*Cisco Express Forwarding*) y sus NLRI (*Network Layer Routing Information*) junto a sus respectivos *labels* en SR; mientras en la *Fig. 4-6* el uso de un comando en IOS-XR evidenciando el no empleo de LDP en la nube del proveedor.

```
VPCS> show ip

NAME        : VPCS[1]
IP/MASK     : 192.168.100.10/24
GATEWAY     : 192.168.100.1
DNS         :
MAC         : 00:50:79:66:68:06
LPORT       : 20000
RHOST:PORT  : 127.0.0.1:30000
MTU         : 1500

VPCS> ping 192.168.200.10

84 bytes from 192.168.200.10 icmp_seq=1 ttl=59 time=14.811 ms
84 bytes from 192.168.200.10 icmp_seq=2 ttl=59 time=15.572 ms
84 bytes from 192.168.200.10 icmp_seq=3 ttl=59 time=15.044 ms
84 bytes from 192.168.200.10 icmp_seq=4 ttl=59 time=15.648 ms
84 bytes from 192.168.200.10 icmp_seq=5 ttl=59 time=15.710 ms
```

Figura 4-3 Prueba de Conectividad entre vPC1 y vPC2 en el PoC de Segment Routing.
Fuente: *Autor*

```
CE-HQ#traceroute 192.168.200.10 source 192.168.100.1 numeric
Type escape sequence to abort.
Tracing the route to 192.168.200.10
VRF info: (vrf in name/id, vrf out name/id)
  1 200.10.10.2 3 msec 1 msec 2 msec
  2 200.10.10.6 [MPLS: Labels 20017/24002 Exp 0] 14 msec 14 msec 15 msec
  3 200.10.10.10 [MPLS: Label 24002 Exp 0] 15 msec 18 msec 16 msec
  4 200.10.10.14 16 msec 18 msec 17 msec
  5 192.168.200.10 20 msec 14 msec 15 msec
```

Figura 4-4 Traceroute entre CE-HQ y vPC2 en el PoC de Segment Routing (Marcaje de SR).
Fuente: *Autor*

```
RP/0/0/CPU0:PE1#show cef 4.4.4.4 | include labels
Mon Apr 19 21:25:50.265 UTC
     local label 20017      labels imposed {20017}
RP/0/0/CPU0:PE1#
RP/0/0/CPU0:PE1#show cef 3.3.3.3 | include labels
Mon Apr 19 21:26:01.214 UTC
     local label 20016      labels imposed {ImplNull}
```

Figura 4-5 Contenido de CEF (NLRI) en PE1 y sus respectivo labels SR en el PoC de Segment Routing
Fuente: *Autor*

```
RP/0/0/CPU0:PE1#show mpls int gi0/0/0/2 detail
Mon Apr 19 21:24:35.600 UTC
Interface GigabitEthernet0/0/0/2:
        LDP labelling not enabled
        LSP labelling not enabled
        MPLS ISIS enabled
        MPLS enabled
```

Figura 4-6 Comprobación del no uso de LDP en el PoC de Segment Routing
Fuente: *Autor*

Finalmente, en las *Fig. 4-7 y Fig. 4-8* se tiene una captura de tráfico mediante *Wireshark* de la comunicación entre vPC1 y vPC2 en la cual se comprueba que SR puede ser transportado en un Plano de Datos MPLS (MPLS *Label 20017*) tipo L3VPNv4 con iBGP como protocolo de enrutamiento entre PEs, facilitando la migración de un entorno MPLS-LDP a SR-MPLS, así como una interoperabilidad adecuada, sin embargo, se debe indicar que SR se concibió para ser transportado nativamente en IPv6, así como usar Controladores SDN en entornos masivos, normalmente usando PCEP tal como se indica en (Salazar Ch., Naranjo, & Marrone, 2018) y en (Salazar-Chacón & Reinoso García, 2021).

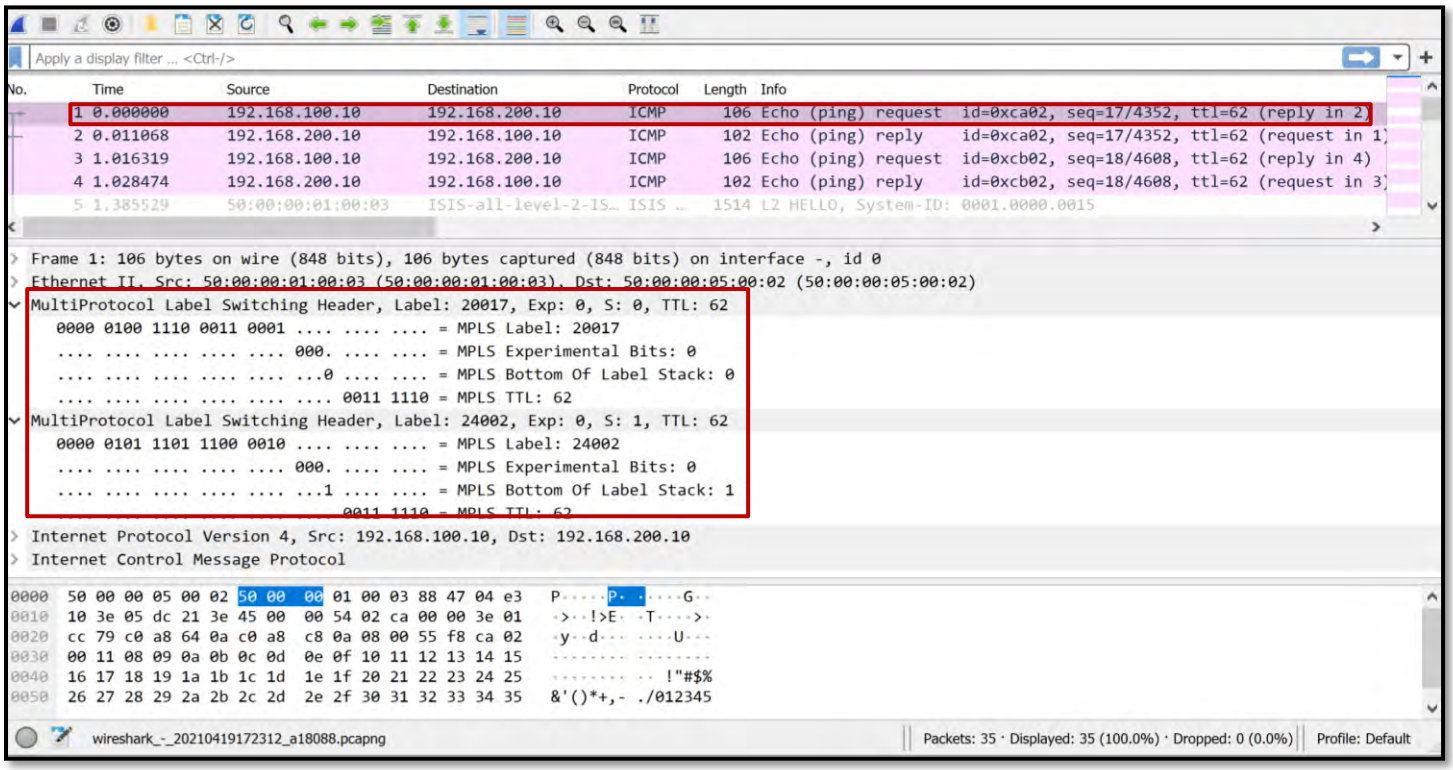


Figura 4-7 Captura de Tráfico en Wireshark - PoC de Segment Routing
Fuente: *Autor*

```
RP/0/0/CPU0:PE1#show mpls forwarding detail
Mon Apr 19 22:39:32.162 UTC
Local  Outgoing    Prefix            Outgoing     Next Hop        Bytes
Label  Label       or ID             Interface                    Switched
------ ----------- ----------------- ------------ --------------- ------------
20016  Pop         SR Pfx (idx 16)   Gi0/0/0/2    200.10.10.6     0
     Updated: Apr 19 20:01:48.000
     Version: 12, Priority: 1
     Label Stack (Top -> Bottom): { Imp-Null }
     NHID: 0x0, Encap-ID: N/A, Path idx: 0, Backup path idx: 0, Weight: 0
     MAC/Encaps: 14/14, MTU: 1500
     Packets Switched: 0

  Traffic-Matrix Packets/Bytes Switched: 0/0
20017  20017       SR Pfx (idx 17)   Gi0/0/0/2    200.10.10.6     109403
     Updated: Apr 19 20:12:30.856
     Version: 14, Priority: 1
     Label Stack (Top -> Bottom): { 20017 }
     NHID: 0x0, Encap-ID: N/A, Path idx: 0, Backup path idx: 0, Weight: 0
     MAC/Encaps: 14/18, MTU: 1500
     Packets Switched: 1285
```

Figura 4-8 MPLS como Plano de Datos de Segment-Routing
Fuente: *Autor*

4.2 Emulación de LISP como *Fabric* de SD-Access

SD-Access/SD-Branch, analizado en ***3.2 Software-Defined Access/Software-Defined Branch y Software-Defined Data*** Center**,** es una clara evolución de la red empresarial comenzando desde la red de acceso, Integrando redes cableadas, redes inalámbricas con WLC, entornos tipo BYOX e IoT, dando visibilidad, automatización y rapidez de conexión entre diferentes sitios empresariales a manera de un Campus Distribuido bajo el mismo entorno corporativo (Red de Tránsito SD-Access) o con integración con SD-WAN en caso que las sedes sean más lejanas, permitiendo una ubicuidad de la conectividad con seguridad en mente.

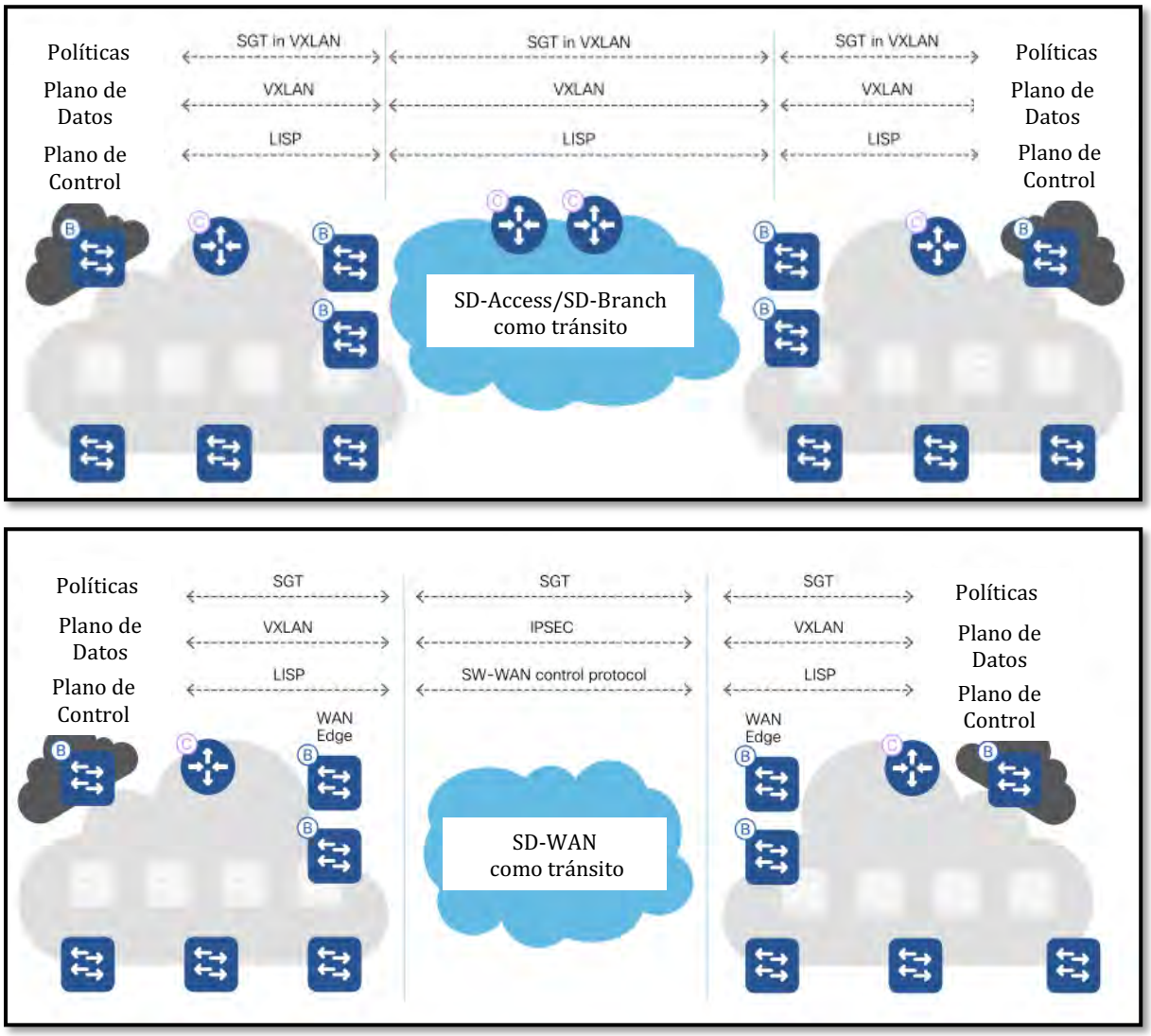


Figura 4-9 Planos de Control y Datos en SD-Access: SD-Access como Tránsito y SD-WAN como tránsito Basado en (Hill, et al., 2019)

En la *Fig. 4-9* se identifican los protocolos necesarios para traer esas ventajas a las redes de nueva generación, por ello, en esta sección se hará una Prueba de Concepto y factibilidad de LISP, protocolo principal para el *fabric* de SD-Access.

4.2.1 Topología y Emulación de LISP

La topología usada para el PoC de LISP se indica en la *Fig. 4-10*, la cual fue diseñada con el siguiente NOS para todos los nodos:

- CSR1000vng-UniversalK9.17.02.01

El objetivo de la prueba de concepto será usar LISP y unir dos sedes empresariales, bajo el sistema de tránsito SD-Access donde estará ubicado el *Map Server/Resolver* mediante el cual se relacionará los RLOCs que identifican cada sede y permitir la transferencia de información.

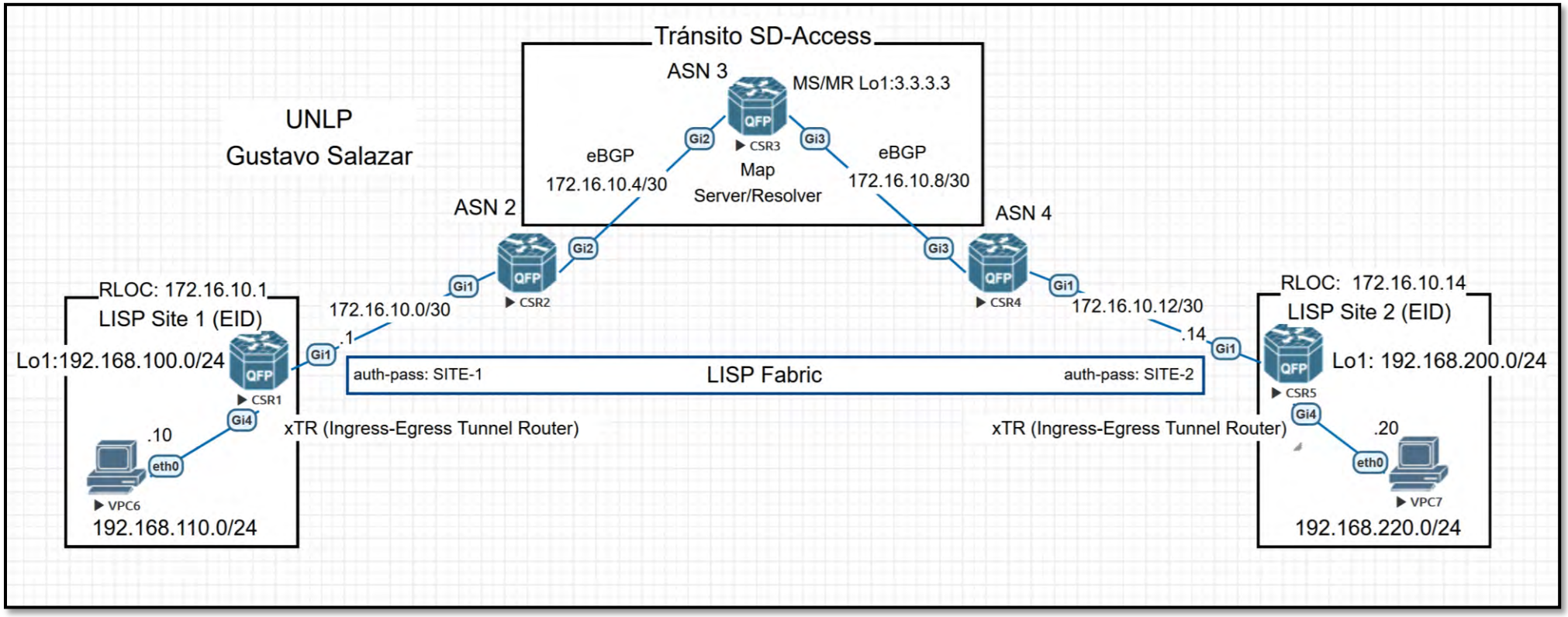


Figura 4-10 Topología PoC – LISP
Fuente: *Autor*

Recordemos que LISP (*Locator/ID Separation Protocol*) es una arquitectura de enrutamiento que cambia la semántica para el direccionamiento IP en una red corporativa, separando la identidad del dispositivo (llamado EID - *Endpoint Identifier)* de su ubicación (llamado RLOC – *Routing Locator*), trayendo con ello muchas ventajas en cuanto a desempeño y seguridad.

Una vez se ha configurado el direccionamiento IPv4 mostrado en la *Fig. 4-10*, se debe establecer el enrutamiento en la nube de tránsito (T1, T2 y T3), conocido como el RLOC *Namespace*. El *Script 29* muestra la configuración de eBGP en la nube para T1 (CSR2). Es necesario exista conectividad entre T1, T2 y T3, incluyendo los enlaces entre Sitio-1 (CSR1) y T1 (CSR2), así como entre Sitio-2 (CSR5) con T3 (CSR4):

```
T1(config)#router bgp 2
T1(config-router)#neigh 172.16.10.6 remote-as 3
T1(config-router)#network 172.16.10.0 mask 255.255.255.252
T1(config-router)#network 172.16.10.4 mask 255.255.255.252
T1(config-router)#
```

Script 29 Configuración de eBGP en T1 para la nube de tránsito SD-Access
Fuente: *Autor*

La nube de SD-Access formará vecindades eBGP dentro de dicha nube. La *Fig. 4-11* muestra la tabla BGP en T2 como parte del *Fabric* SD-Access.

```
T2#show ip bgp summary
BGP router identifier 3.3.3.3, local AS number 3
BGP table version is 13, main routing table version 13
5 network entries using 1240 bytes of memory
6 path entries using 816 bytes of memory
3/3 BGP path/bestpath attribute entries using 864 bytes of memory
2 BGP AS-PATH entries using 48 bytes of memory
0 BGP route-map cache entries using 0 bytes of memory
0 BGP filter-list cache entries using 0 bytes of memory
BGP using 2968 total bytes of memory
BGP activity 5/0 prefixes, 6/0 paths, scan interval 60 secs
5 networks peaked at 06:48:25 Apr 21 2021 UTC (00:06:55.182 ago)

Neighbor        V           AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
172.16.10.5     4            2      22      22       13    0    0 00:15:28        2
172.16.10.10    4            4      13      16       13    0    0 00:07:58        1
```

Figura 4-11 Tabla BGP – Nube de Tránsito LISP (eBGP)
Fuente: *Autor*

Las sedes o LISP EID *Namespaces* (para el PoC las redes 192.168.100.0/24 y 192.168.110.0/24 para el SITIO-1 y 192.168.200.0/24 y 192.168.220.0/24 para el SITIO-2) no se deben enviar al RLOC *Namespace*, este es un punto clave en la seguridad que entrega LISP, ya que la conectividad de extremo a extremo tiene lugar gracias a los pares del Túnel LISP: *Ingress/Egress Tunnel Router* (ITR/ETR). Para el PoC de la *Fig. 4-10* SITE-1(CSR1) y SITE-2 (CSR5) toman la función de xTR o equipo que funciona como ITR y ETR, equipos que generarán el proceso de encapsulación/desencapsulación LISP luego del proceso de solicitud-respuesta enviado hacia el *Map Server/Resolver* y establecer el *Fabric* LISP. Existen roles adicionales como los LISP Proxy ETR/ITR, los cuales permiten la comunicación a entornos sin soporte de LISP.

La función de un *Map Resolver* (MR) es la de aceptar solicitudes de un ITR, desencapsular dichos mensajes y luego enviarlos hacia el *Map Server* (MS) responsable del ETR.

Por motivos de escalabilidad y resiliencia, los equipos asignados como MR/MS, ya sea que funcionen independientemente o en un mismo equipo, deben tener redundancia con otros MR/MS a través de GRE o sesiones BGP. Estos escenarios se conocen como LISP *Alternative Logical Topology* (ALT), la cual no está presente en todas las implementaciones.

Hasta este punto, la conectividad entre T1 y T3 (conectividad en el RLOC *namespace*) se comprueba mediante un *ping* (conectividad en nube de tránsito mediante eBGP):

```
T1#ping 172.16.10.10
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.10.10, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/8/13 ms
T1#
T1#traceroute 172.16.10.10
Type escape sequence to abort.
Tracing the route to 172.16.10.10
VRF info: (vrf in name/id, vrf out name/id)
  1 172.16.10.6 6 msec 1 msec 2 msec
  2 172.16.10.10 [AS 3] 25 msec 16 msec *
```

Figura 4-12 Conectividad en la Nube de Tránsito LISP (eBGP) – Ping T1(CSR2) a T3 (CSR4)
Fuente: *Autor*

Conectividad mediante LISP *Fabric*

Con el fin de conectar el Sitio 1 (SITE-1) y Sitio 2 (SITE-2), se debe configurar LISP tanto en los extremos (xTRs), así como en la nube de tránsito. Los Sitios 1 y 2 deben apuntar a un *Map Server/Resolver* ubicado en la nube de tránsito y tener una ruta por defecto hacia la nube LISP.

El *Script 30* muestra el establecimiento de LISP en *Site-1* (CSR1):

```
SITE-1(config)#router lisp
SITE-1(config-router-lisp)#locator-set SITE-1
SITE-1(config-router-lisp-locator-set)#172.16.10.1 priority 0 weight 0
SITE-1(config-router-lisp-locator-set)#exit-locator-set
SITE-1(config-router-lisp)#
SITE-1(config-router-lisp)#instance-id 0
SITE-1(config-lisp-inst)#service ipv4
SITE-1(config-lisp-inst-srv-ipv4)#eid-table default
SITE-1(config-lisp-inst-srv-ipv4)#database-mapping 192.168.100.0/24 locator-set SITE-1
SITE-1(config-lisp-inst-srv-ipv4)#database-mapping 192.168.110.0/24 locator-set SITE-1
SITE-1(config-lisp-inst-srv-ipv4)#
SITE-1(config-lisp-inst-srv-ipv4)#itr map-resolver 3.3.3.3
SITE-1(config-lisp-inst-srv-ipv4)#itr
SITE-1(config-lisp-inst-srv-ipv4)#
*Apr 23 06:16:01.234: %LINEPROTO-5-UPDOWN: Line protocol on Interface LISP0, changed state to up
SITE-1(config-lisp-inst-srv-ipv4)#
SITE-1(config-lisp-inst-srv-ipv4)#etr map-server 3.3.3.3 key SITE-1
SITE-1(config-lisp-inst-srv-ipv4)#etr
SITE-1(config-lisp-inst-srv-ipv4)#exit-service-ipv4
SITE-1(config-lisp-inst)#exit-instance-id
SITE-1(config-router-lisp)#exit-router-lisp
SITE-1(config)#
SITE-1(config)#ip route 0.0.0.0 0.0.0.0 172.16.10.2
SITE-1(config)#
```

Definición del RLOC. *Priority* y *Weight* permiten seleccionar caminos y balancear carga.

Creación de EIDs: Designación de IPs de sedes

Identificación del *Map-Resolver* y creación de interfaz LISP0

Identificación del *Map-Server* junto a su llave de autenticación

Configuración de ruta estática por defecto apuntado a la nube LISP

Script 30 Configuración de LISP en xTR
Fuente: *Autor*

La configuración del *Script 30*, es la misma para el otro extremo del túnel LISP, obviamente siguiendo el direccionamiento correspondiente del *Site-2* (CSR5).

El siguiente paso es definir el Map Server/Resolver en la nube de tránsito LISP, para lo cual se debe ejecutar los comandos establecido en el *Script 31* para configurar el direccionamiento y autenticación para el registro del Sitio 1 y Sitio 2 en T2 (CSR3):

```
T2(config)#router lisp
T2(config-router-lisp)#service ipv4
T2(config-lisp-srv-ipv4)#map-server
T2(config-lisp-srv-ipv4)#map-resolver
T2(config-lisp-srv-ipv4)#exit-service-ipv4
T2(config-router-lisp)#
T2(config-router-lisp)#instance-id 0
T2(config-lisp-inst)#service ipv4
T2(config-lisp-inst-srv-ipv4)#exit-service-ipv4
T2(config-lisp-inst)#
T2(config-lisp-inst)#site SITE-1
T2(config-router-lisp-site)#authentication-key SITE-1
T2(config-router-lisp-site)#eid-record 192.168.100.0/24 accept-more-specifics
T2(config-router-lisp-site)#eid-record 192.168.110.0/24 accept-more-specifics
T2(config-router-lisp-site)#
T2(config-router-lisp-site)#exit-site
T2(config-router-lisp)#
T2(config-router-lisp)#site SITE-2
T2(config-router-lisp-site)#authentication-key SITE-2
T2(config-router-lisp-site)#eid-record 192.168.200.0/24 accept-more-specifics
T2(config-router-lisp-site)#eid-record 192.168.220.0/24 accept-more-specifics
T2(config-router-lisp-site)#exit-site
T2(config-router-lisp)#
T2(config-router-lisp)#do wr
Building configuration...
[OK]
```

Habilitación de funciones como *Map-Server* y *Map-Resolver* y asignación de *instance-id 0* bajo IPv4

Definición de Sitios LISP

Script 31 Configuración de LISP en xTR
Fuente: *Autor*

4.2.2 Resultados de la Emulación de LISP

Al terminar la configuración de los nodos con éxito, se puede alcanzar la conectividad de extremo a extremo, es decir, entre Sitio-1 y Sitio-2 del PoC propuesto en la *Fig. 4-10.*

Mediante el comando de la *Fig. 4-13* se puede verificar el establecimiento de sesión LISP entre los RLOCs de SITE-1 y SITE-2 con LISP *Map-Server.*

```
SITE-1#show lisp session

Sessions for VRF default, total: 1, established: 1
Peer                      State      Up/Down        In/Out    Users
3.3.3.3:4342              Up         00:43:01        7/5      3
SITE-1#
```

```
SITE-2#show lisp session

Sessions for VRF default, total: 1, established: 1
Peer                      State      Up/Down        In/Out    Users
3.3.3.3:4342              Up         00:45:49        7/5      3
```

Figura 4-13 Establecimiento de sesión LISP xTRs con Map Server/Resolver
Fuente: *Autor*

Desde el punto de vista del *Map Server/Resolver*, se debe verificar el establecimiento de la relación entre este equipo y los RLOCs hacia cada uno de los EID *namespaces* definidos para la conectividad entre los sitios. La *Fig.4-14* muestra el estado de *UP* y del registro de los equipos parte de la infraestructura LISP.

```
T2#show lisp site
LISP Site Registration Information
* = Some locators are down or unreachable
# = Some registrations are sourced by reliable transport

Site Name      Last      Up     Who Last             Inst     EID Prefix
               Register         Registered           ID
SITE-1         00:14:37  yes#   172.16.10.1:30514             192.168.100.0/24
               00:10:07  yes#   172.16.10.1:30514             192.168.110.0/24
SITE-2         00:18:33  yes#   172.16.10.14:51099            192.168.200.0/24
               00:08:54  yes#   172.16.10.14:51099            192.168.220.0/24
```

Figura 4-14 Registro por parte del Map Server/Resolver a los RLOCs (xTRs) y EIDs de cada sitio en PoC de LISP
Fuente: *Autor*

La prueba de conectividad de extremo a extremo se establece mediante un ping entre los Sitios y un *traceroute* entre el RLOC del Sitio-1 al Sitio-2:

```
VPCS> show ip

NAME        : VPCS[1]
IP/MASK     : 192.168.110.10/24
GATEWAY     : 192.168.110.1
DNS         :
MAC         : 00:50:79:66:68:06
LPORT       : 20000
RHOST:PORT  : 127.0.0.1:30000
MTU         : 1500

VPCS> ping 192.168.200.1

84 bytes from 192.168.200.1 icmp_seq=1 ttl=251 time=2.245 ms
84 bytes from 192.168.200.1 icmp_seq=2 ttl=251 time=1.913 ms
84 bytes from 192.168.200.1 icmp_seq=3 ttl=251 time=2.078 ms
84 bytes from 192.168.200.1 icmp_seq=4 ttl=251 time=1.971 ms
84 bytes from 192.168.200.1 icmp_seq=5 ttl=251 time=4.292 ms

VPCS> ping 192.168.220.20

84 bytes from 192.168.220.20 icmp_seq=1 ttl=59 time=2.467 ms
84 bytes from 192.168.220.20 icmp_seq=2 ttl=59 time=1.951 ms
84 bytes from 192.168.220.20 icmp_seq=3 ttl=59 time=2.261 ms
84 bytes from 192.168.220.20 icmp_seq=4 ttl=59 time=2.278 ms
```

Figura 4-15 Conectividad Extremo a Extremo: Ping entre Sitio-1 y Sitio-2 en el PoC de LISP
Fuente: *Autor*

```
SITE-1#traceroute 192.168.220.20 numeric
Type escape sequence to abort.
Tracing the route to 192.168.220.20
VRF info: (vrf in name/id, vrf out name/id)
  1 172.16.10.2 1 msec 2 msec 1 msec
  2 172.16.10.6 1 msec 2 msec 1 msec
  3 172.16.10.10 2 msec 4 msec 3 msec
  4 172.16.10.14 4 msec 4 msec 4 msec
  5 192.168.220.20 5 msec 2 msec 2 msec
SITE-1#
```

Figura 4-16 Traceroute entre RLOC de Sitio-1 (xTR) y Sitio-2 en el PoC de LISP
Fuente: *Autor*

Al existir la conectividad mostrada en las *Figs. 4-15* y *4-16*, es factible que los nodos de cada Sitio o xTRs tengan conocimiento sobre los EIDs del otro Sitio, así, por ejemplo, el nodo *Site-1* con RLOC 172.16.10.1 puede conocer mediante LISP los EIDs 192.168.200.0/24 y 192.168.220.0/24 a través del RLOC 172.16.10.14. De la misma manera, el *Site-2*. Estos EIDs no se envían en la nube de tránsito LISP (SD-Access), dando así seguridad y segmentación.

La *Fig. 4-17* muestra ese aprendizaje a través del *Fabric* LISP.

```
SITE-1#show ip lisp map-cache
LISP IPv4 Mapping Cache for EID-table default (IID 0), 3 entries

0.0.0.0/0, uptime: 00:00:00, expires: 00:00:59, via static-send-map-request
  Negative cache entry, action: send-map-request
192.168.200.0/24, uptime: 00:31:27, expires: 23:28:32, via map-reply, complete
  Locator       Uptime    State  Pri/Wgt     Encap-IID
  172.16.10.14  00:31:27  up       1/1         -
192.168.220.0/24, uptime: 00:08:29, expires: 23:51:31, via map-reply, complete
  Locator       Uptime    State  Pri/Wgt     Encap-IID
  172.16.10.14  00:08:29  up       0/0         -
```

```
SITE-2#show ip lisp map-cache
LISP IPv4 Mapping Cache for EID-table default (IID 0), 3 entries

0.0.0.0/0, uptime: 00:00:01, expires: 00:00:59, via static-send-map-request
  Negative cache entry, action: send-map-request
192.168.100.0/24, uptime: 00:21:30, expires: 23:38:29, via map-reply, complete
  Locator      Uptime    State  Pri/Wgt     Encap-IID
  172.16.10.1  00:21:30  up       0/0         -
192.168.110.0/24, uptime: 00:13:30, expires: 23:46:29, via map-reply, complete
  Locator      Uptime    State  Pri/Wgt     Encap-IID
  172.16.10.1  00:13:30  up       0/0         -
```

Figura 4-17 Traceroute entre RLOC de Sitio-1 y Sitio-2 en el PoC de LISP
Fuente: *Autor*

Se tomó capturas de tráfico con Wireshark para demostrar mediante este PoC la estructura de la encapsulación LISP, la cual está en el RFC experimental 6830[53]: *The Locator/ID Separation Protocol (LISP),* tal como se observa en las *Fig. 4-18* y *Fig. 4-19.*

[53] RFC 6830: https://tools.ietf.org/html/rfc6830

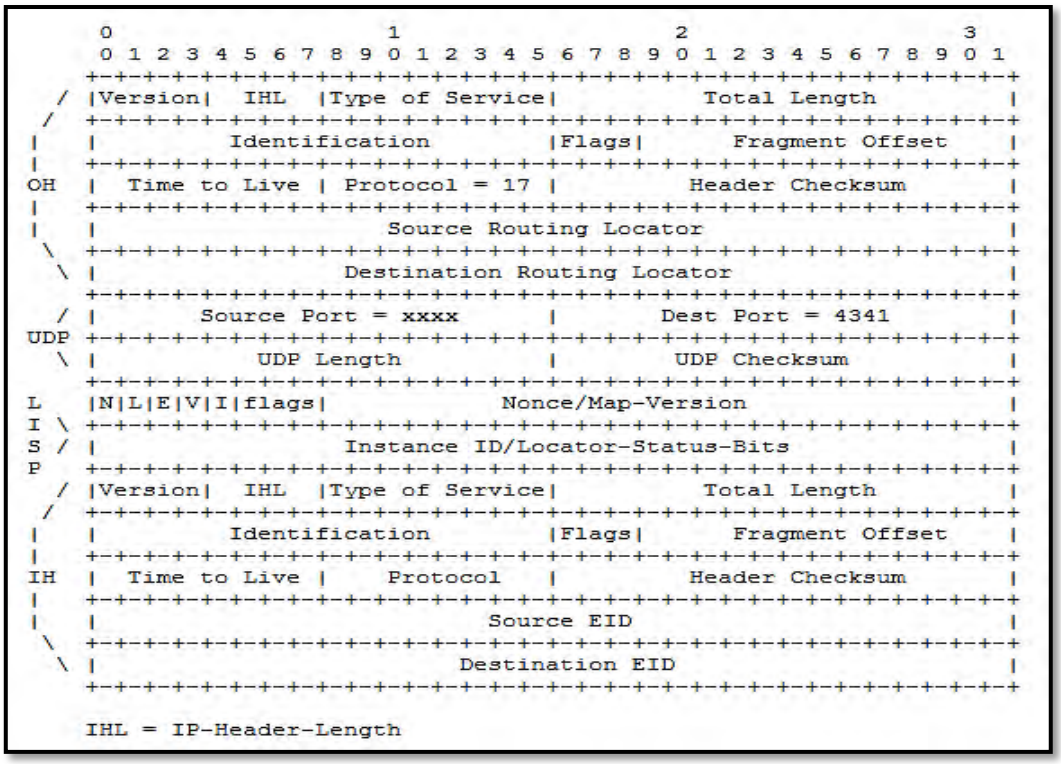


Figura 4-18 Esquematización del Encabezado LISP IPv4-in-IPv4
Recuperado de (Internet Engineering Task Force - IETF, 2013)

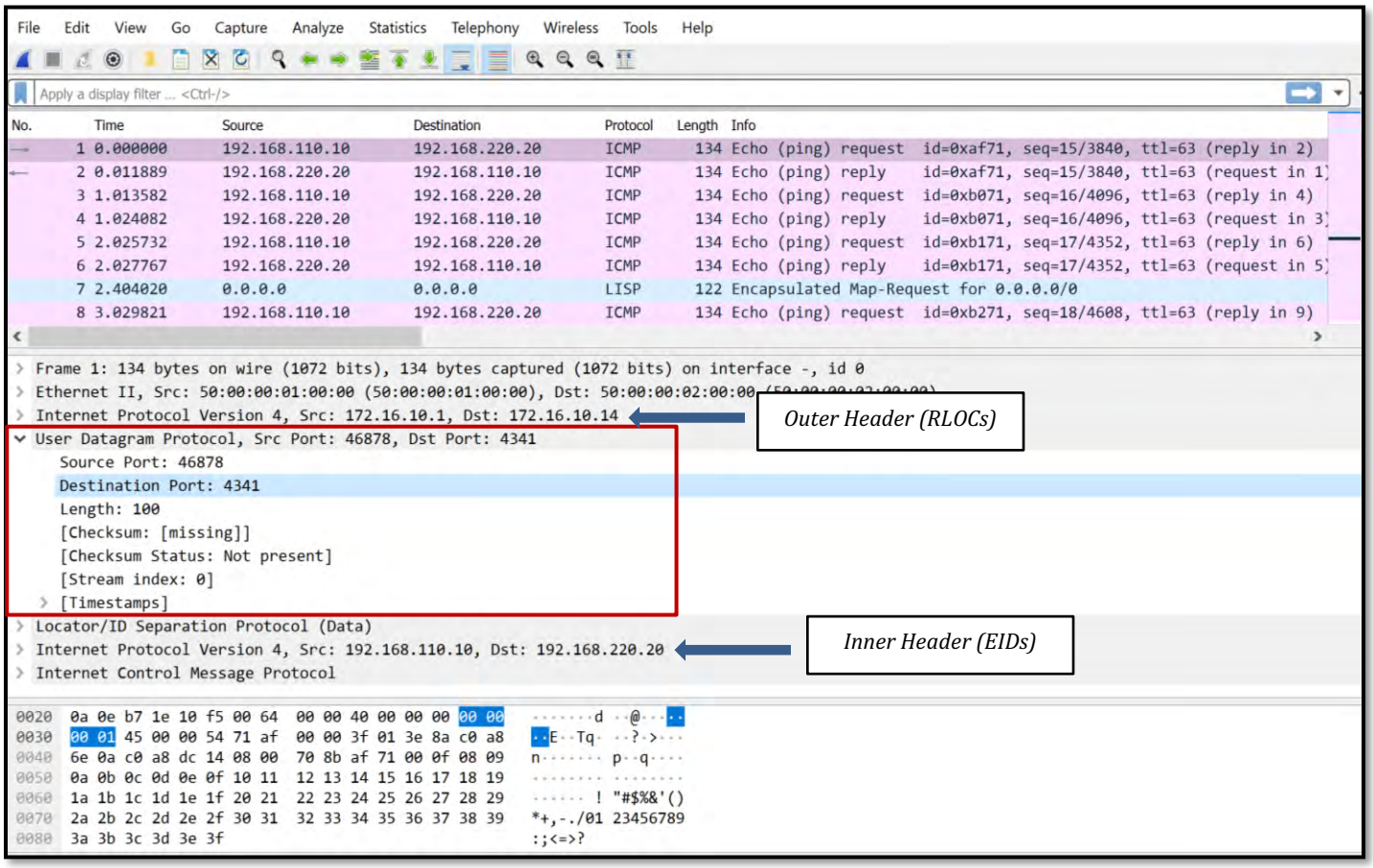


Figura 4-19 Captura de Tráfico en Wireshark – Encabezado de LISP
Fuente: *Autor*

Fue posible durante un ping continuo capturar un LISP *Map Request* desde el Sitio-1 (xTR-RLOC) y el LISP *Map ACK*

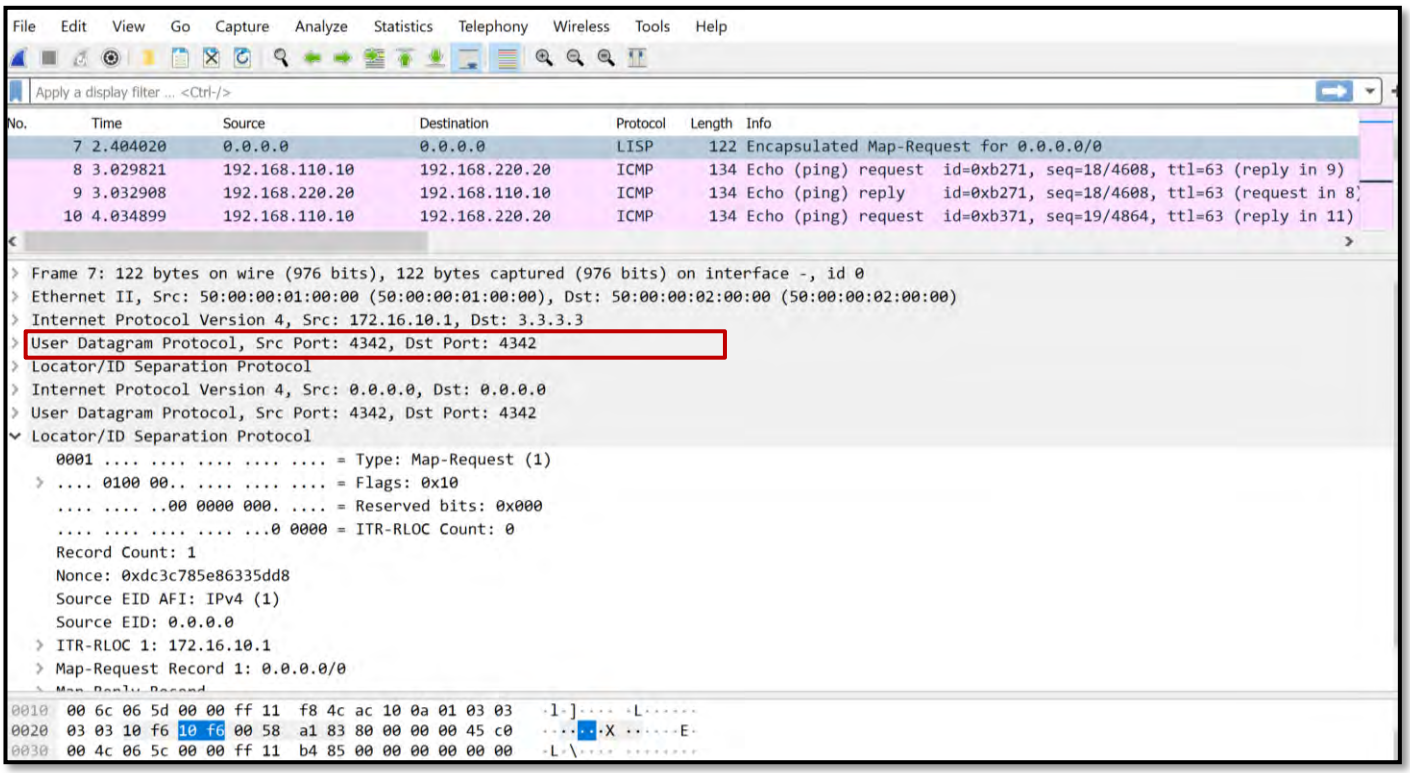


Figura 4-20 Captura de Tráfico en Wireshark – LISP Map Request
Fuente: *Autor*

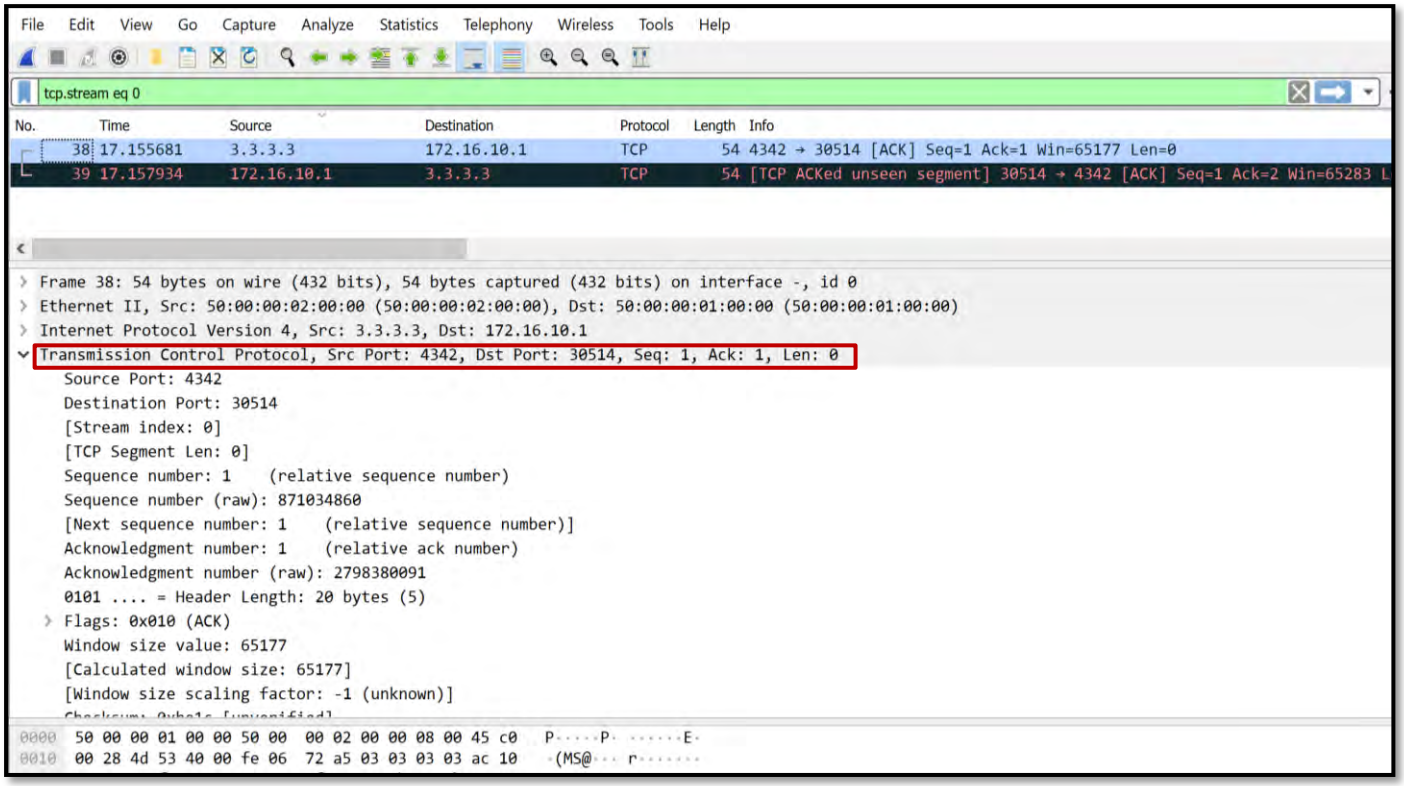


Figura 4-21 Captura de Tráfico en Wireshark – LISP Map ACK (TCP)
Fuente: *Autor*

Para finalizar con este PoC, se debe mencionar que investigaciones recientes proponen a LISP como un protocolo *Southbound* de SDN (Rodriguez-Natal, et al., 2015), es más, luego de concluir con éxito el PoC se puede decir que LISP posee una estructura alineada al contexto SDN (Separación de Plano de Control y Datos) mediante la relación xTR-RLOC y *Map Server/Resolver*, lo que lo hace un protocolo ideal para Redes de Nueva Generación.

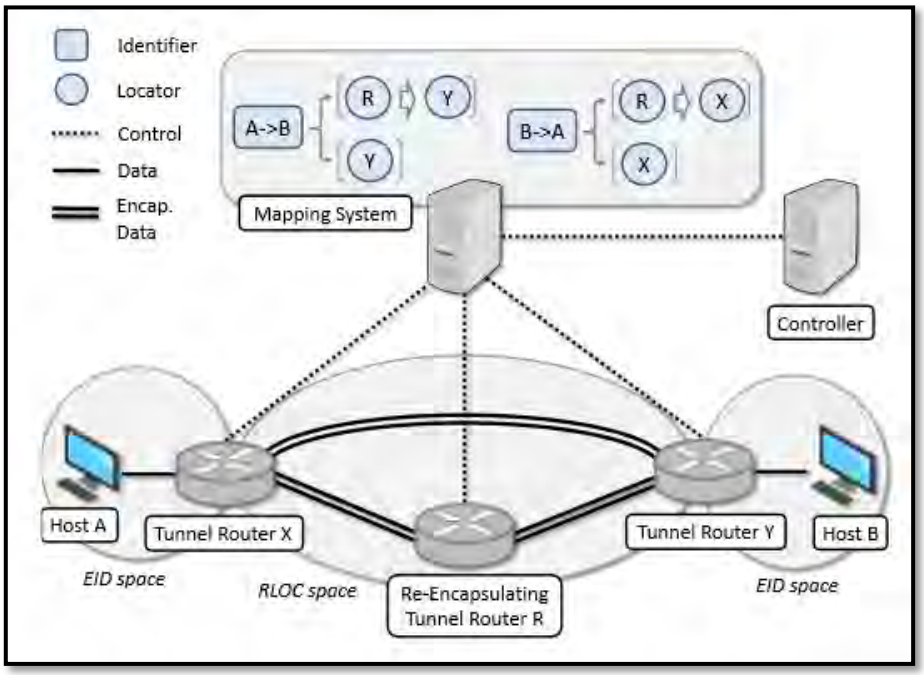
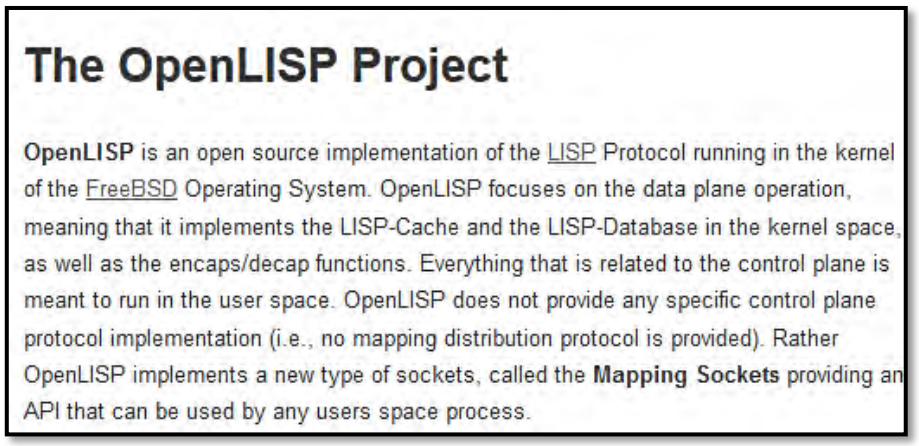


Figura 4-22 Esquematización de LISP como protocolo Southbound en contexto SDN
Recuperado de (Rodriguez-Natal, et al., 2015)

LISP está en pleno proceso de apertura, pues el proyecto *OpenLISP*[54], toma lo mejor de la implementación propietaria, junto con las especificaciones del RFC 6830 en la implementación del protocolo en un *kernel FreeBSD*-NOS manejando *sockets* y tuplas para dar lugar a la separación de entre EID y RLOC de un equipo.



The OpenLISP Project

OpenLISP is an open source implementation of the LISP Protocol running in the kernel of the FreeBSD Operating System. OpenLISP focuses on the data plane operation, meaning that it implements the LISP-Cache and the LISP-Database in the kernel space, as well as the encaps/decap functions. Everything that is related to the control plane is meant to run in the user space. OpenLISP does not provide any specific control plane protocol implementation (i.e., no mapping distribution protocol is provided). Rather OpenLISP implements a new type of sockets, called the **Mapping Sockets** providing an API that can be used by any users space process.

Figura 4-23 Proyecto OpenLISP
Recuperado de (OpenLisp, n.d.)

[54] OpenLISP: http://www.openlisp.org/

4.3 Emulación de Red OpenSDN Híbridas: SDN y Redes Tradicionales

Según lo mencionado en el ***Capítulo 2: Fundamentos de las Redes Definidas por Software***, existe una conceptualización generalizada de estas Redes de nueva generación dada por la ONF:

> "Se define a este paradigma como la separación del Plano de Control del Plano de Datos que tradicionalmente está en el mismo equipo de red, con el fin de centralizar el control, administración y monitoreo de varios dispositivos en un solo rol denominado Controlador, abstrayendo de esa forma los planos estructurales de una infraestructura." (Open Networking Foundation - ONF, 2020)

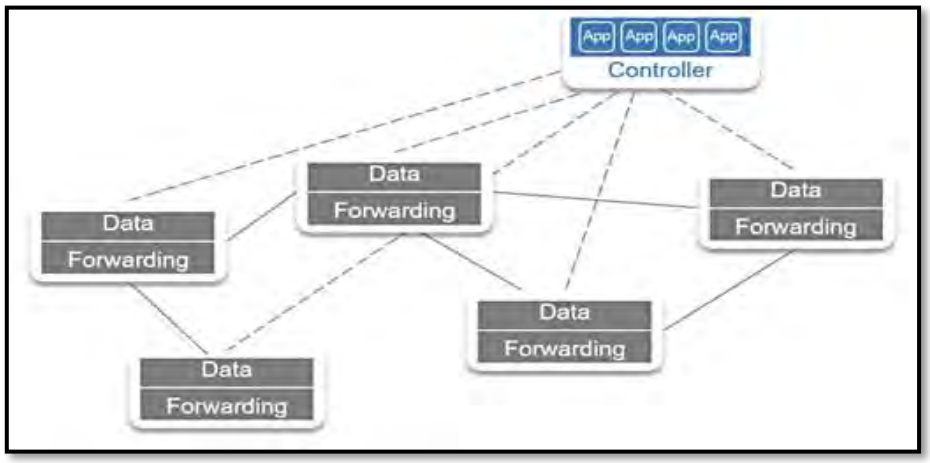


Figura 4-24 Esquematización de Administración/Control Centralizado en SDN
Recuperado de (Guillermo, 2017)

Sin embargo, implementar estos entornos en ambientes de producción real fue posible gracias al concepto de *Underlay-Overlay*, motivando así el uso de los distintos "sabores" de SDN: *SD-Access/SD-Branch, SD-DataCenter* y *SD-WAN* (Salazar-Chacón & Marrone, 2020), donde un Controlador es el encargado de definir las políticas de enrutamiento, gestión y administración necesarias para la Transformación Digital moderna. Dichas políticas son enviadas hacia el plano de datos usando protocolos *Southbound* como *OpenFlow*. Según (Salazar-Chacón & Marrone, 2020), cuando la red SDN emplea *OpenFlow*, se convierte en *OpenSDN*.

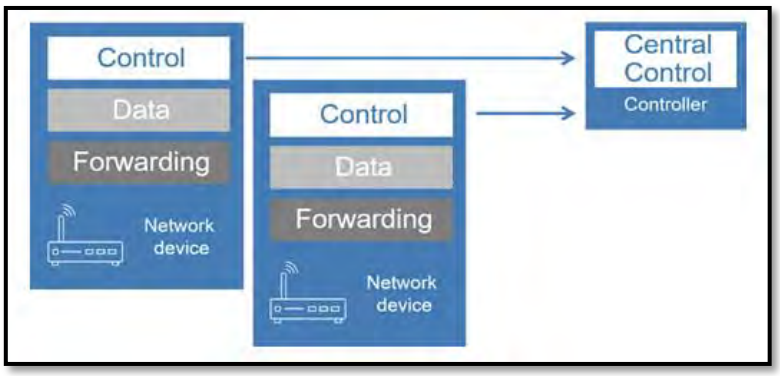


Figura 4-25 Esquematización de OpenSDN tradicional
Recuperado de (Guillermo, 2017)

El protocolo *OpenFlow* fue analizado en ***2.6 Funcionamiento de las Redes SDN basadas en OpenFlow (Open Networking Foundation)***, por lo que en este capítulo se pondrá en ejecución un PoC en Mininet y GNS3-VM para comprobar el funcionamiento y factibilidad de implementación utilizando tanto un Controlador *OpenDayLight* como ONOS (ver ***2.5 OpenDaylight Project y NFV***) para gestionar y controlar redes SDN y redes tradicionales.

Basada en la arquitectura de *OpenFlow* mostrada en la *Fig. 4-26*, Mininet permitirá emular dicha arquitectura con total precisión y fidelidad.

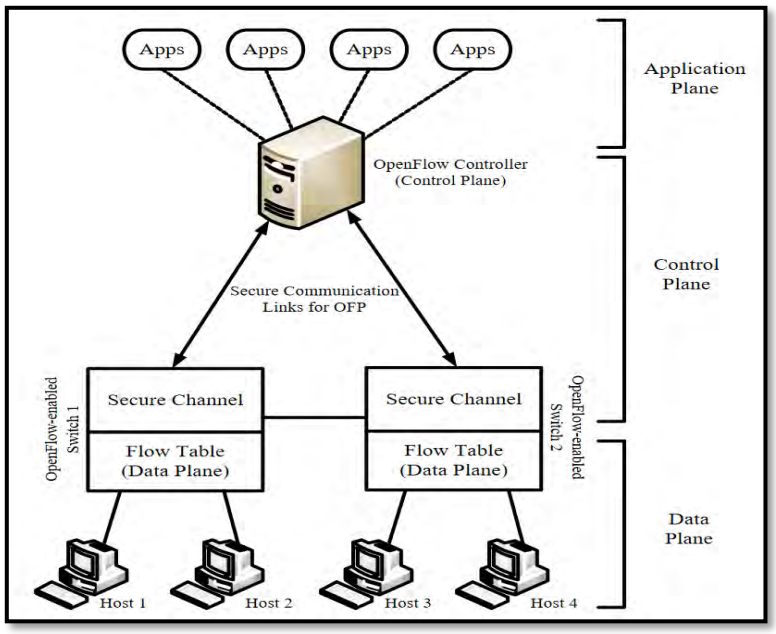


Figura 4-26 Arquitectura de OpenFlow emulada en Mininet
Recuperado de (Zoher Bholebawa & Dalal, 2016)

Para conocer la instalación de Mininet y GNS3-VM, puede leer el ***Anexo B: Guía de Instalación GNS3-VM, EVE-ng y Mininet*****,** sin embargo, para este PoC se actualizó nuevamente tanto la GUI de GNS3 como la VM a la versión 2.2.20, versión actual al momento de realizar esta prueba de concepto y factibilidad.

Figura 4-27 Actualización de GNS3-VM a la versión 2.2.20
Fuente: *Autor*

4.3.1 Instalación de Controladores SDN: ODL y ONOS

En cuanto a los controladores SDN, se emplearán tanto ODL como ONOS.

Para la instalación de ODL, se ingresará a la página oficial, lugar que nos indica que la versión estable de *OpenDayLight* es *Aluminium*. Se descarga dicha versión para este PoC.

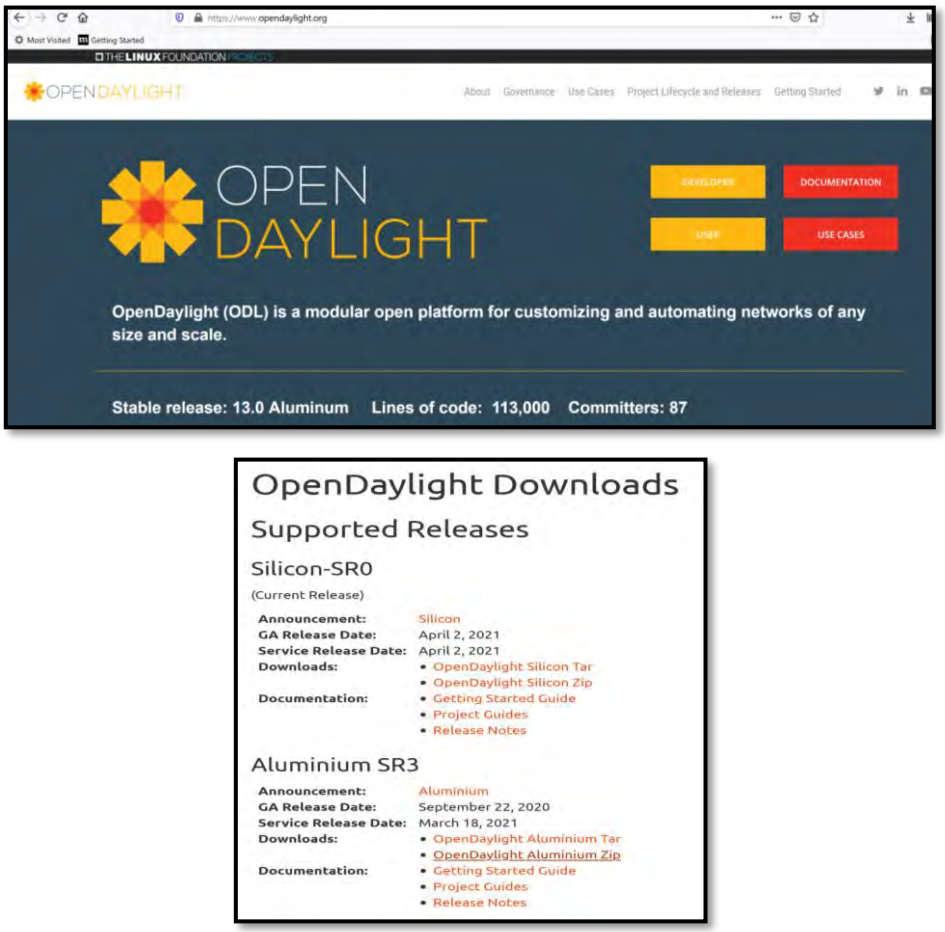


Figura 4-28 Open DayLight version estable 13.0 Aluminium
Fuente: (OpenDaylight - The Linux Foundation Projects, 2021)

La instalación de ODL tomará lugar en Ubuntu 18.04 previamente alojado en GNS3 bajo el siguiente procedimiento:

Crear un proyecto en GNS3-VM y ubicar en la zona de trabajo a Ubuntu junto con una nube NAT para que el equipo pueda tener salida a Internet (*Fig. 4-29*).

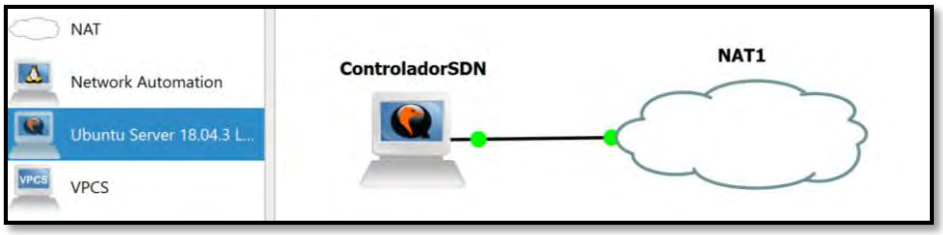


Figura 4-29 Servidor Ubuntu en GNS3-VM para instalación de ODL y ONOS
Fuente: *Autor*

Arrancar Ubuntu y en una terminal realizar un *update*

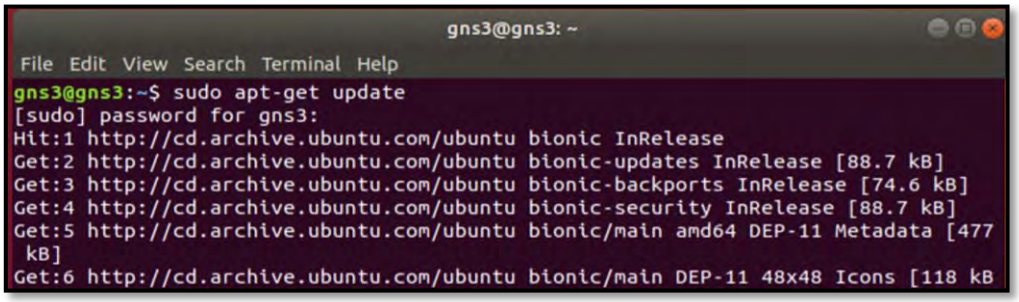


Figura 4-30 Update de Ubuntu previa instalación de ODL
Fuente: *Autor*

Luego, se instalará Java para el adecuado funcionamiento de ODL y exportar su ubicación

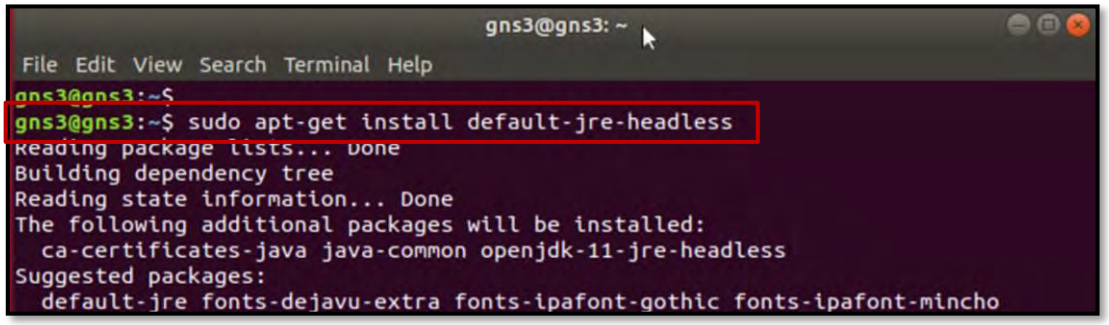


Figura 4-31 Instalación de Java en Ubuntu previa instalación de ODL
Fuente: *Autor*

Para obtener ODL de la página oficial (wget), además de definir la variable de entorno JAVA_HOME, se realiza lo siguiente:

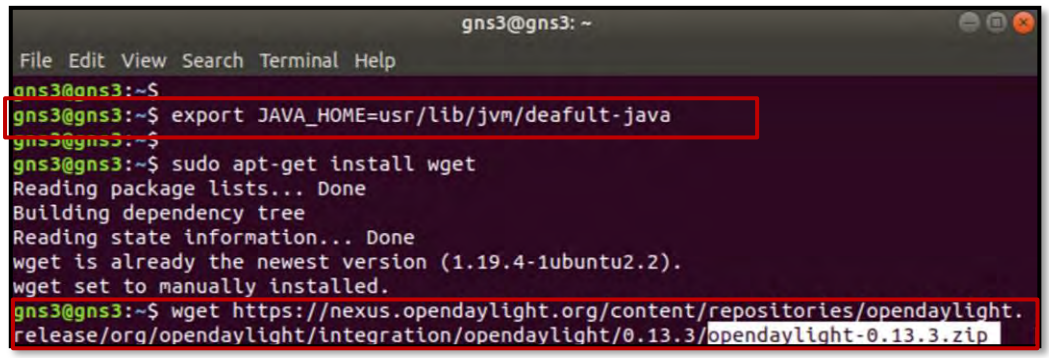


Figura 4-32 Descarga de ODL Aluminium
Fuente: *Autor*

Una vez descargado ODL, entrar a la carpeta previamente descomprimida con *unzip*:

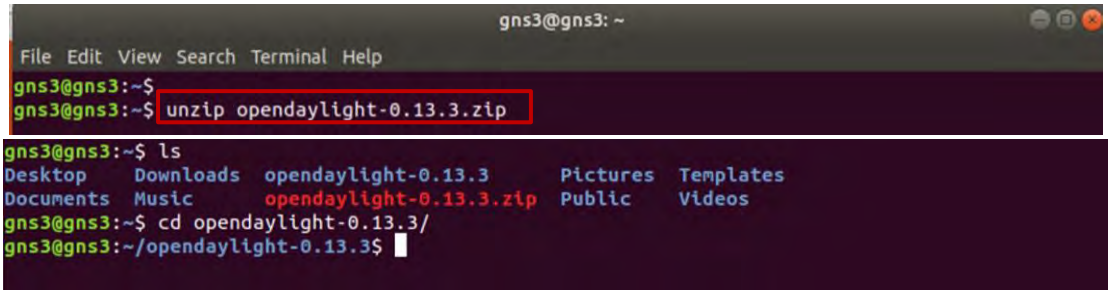


Figura 4-33 Descomprimir ODL Aluminium
Fuente: *Autor*

Ya que el *Front-End* de ODL funciona con *Apache Karaf*[55], este debe ser inicializado primero en cada ocasión que se desee arrancar el controlador SDN.

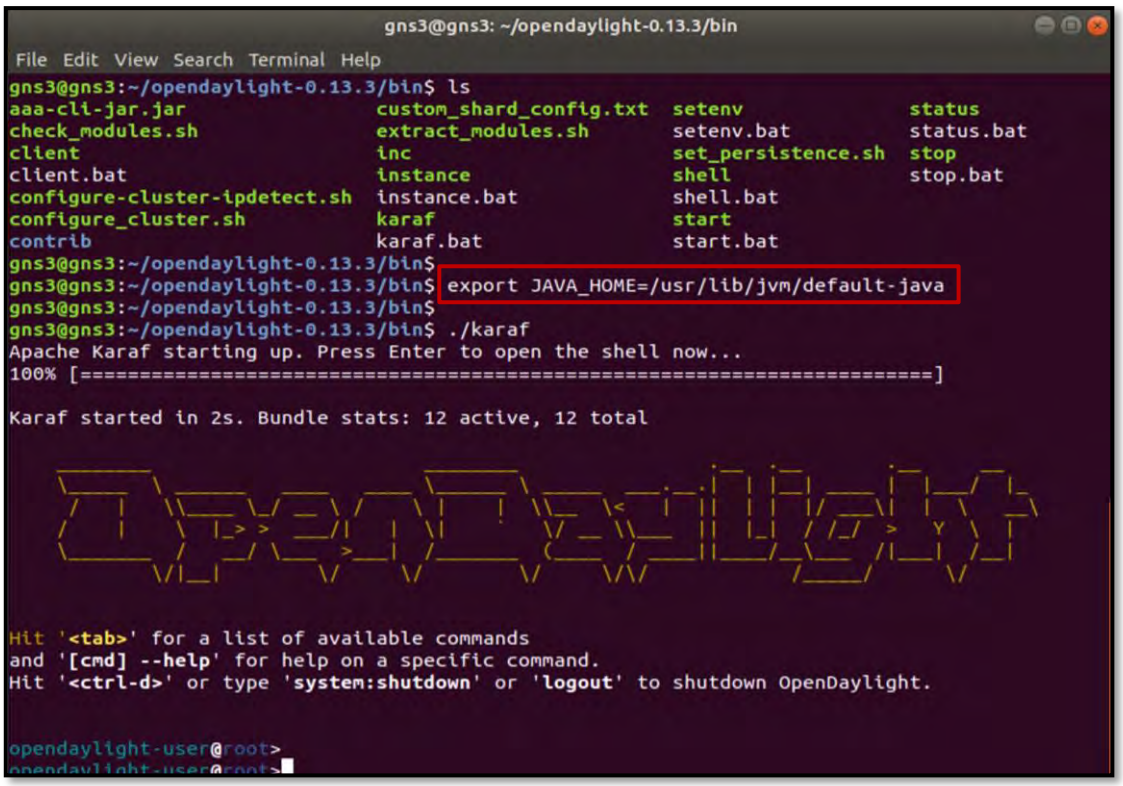


Figura 4-34 Inicialización de Apache Karaf - ODL Aluminium
Fuente: *Autor*

El paso antes de abrir el *Front-End* del controlador es activar ciertos *features* para que ODL pueda administrar el plano de datos, entre ellos DLUX (GUI de ODL), ODL-L2Switch, etc:

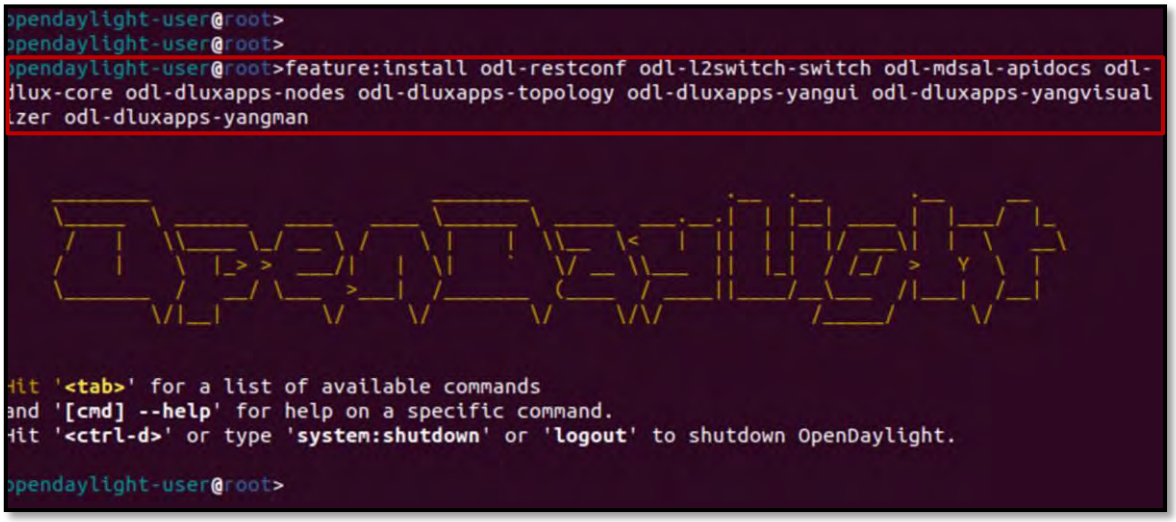


Figura 4-35 Activación de features de ODL para el plano de datos
Fuente: *Autor*

[55] Apache Karaf: https://karaf.apache.org/stories

En caso de que no sea posible activar ciertos *features,* en especial L2Switch y DLUX (*Front-End*), se debe instalar una versión anterior de ODL como la *Nitrogen* y que funcione junto a *Java Environment 8*. Lamentablemente este es un impedimento de sacar el máximo potencial de las últimas versiones de ODL, pues se implementan en JRE 11 o superior, además de perder el soporte por parte de la comunidad *OpenDayLight* de ciertos *features* de control de infraestructura.

Para ingresar al GUI, abra un navegador y digite la siguiente URL en el equipo donde se instaló ODL:

Localhost:8181/index.html

Ingrese con el *user/password* de **admin/admin** tal como en la *Fig. 4-36*

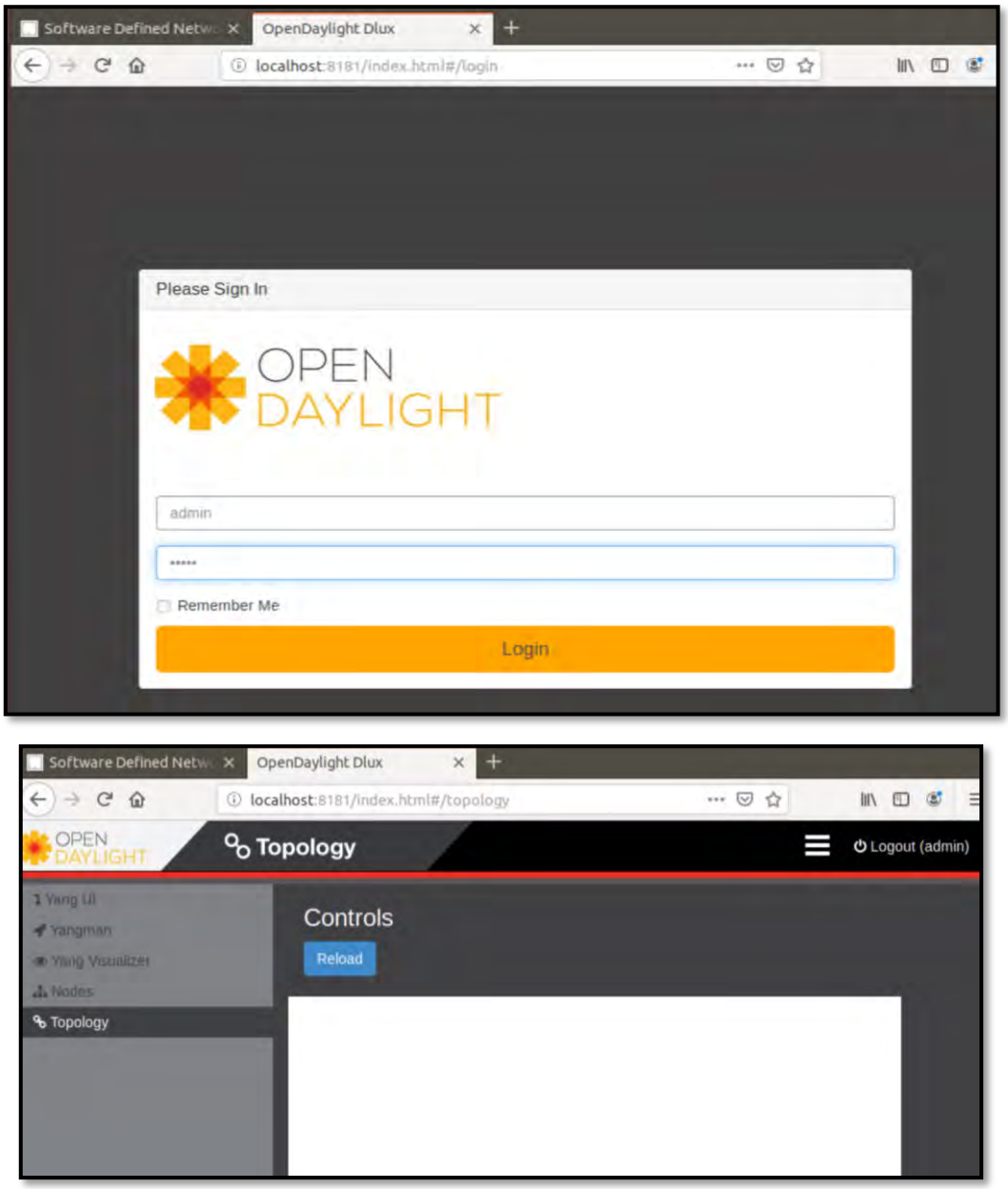


Figura 4-36 Front-End de ODL
Fuente: *Autor*

Por otro lado, para la instalación de ONOS, se requiere de ciertos paquetes y dependencias en un equipo con Linux, por lo que se necesita acceso a Internet a través de una nube NAT en GNS3 previa su actualización (*apt-get update*) como lo visto en la *Fig. 4-29* y *Fig. 4-30.*

Para la correcta instalación de ONOS se clonará un repositorio mediante Git a través de *Git Core*.

```
gns3@gns3:~$
gns3@gns3:~$ sudo apt-get install git-core
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'git' instead of 'git-core'
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk
  gitweb git-cvs git-mediawiki git-svn
The following packages will be upgraded:
  git
1 upgraded, 0 newly installed, 0 to remove and 457 not upgraded.
Need to get 3,916 kB of archives.
After this operation, 73.7 kB of additional disk space will be used.
Get:1 http://cd.archive.ubuntu.com/ubuntu bionic-updates/main amd64 git amd64 1
2.17.1-1ubuntu0.8 [3,916 kB]
25% [1 git 1,217 kB/3,916 kB 31%]
```

Figura 4-37 Instalación de Git Core - ONOS
Fuente: *Autor*

ONOS requiere de *Maven*[56] o *Buck*[57] para compilar, gestionar y añadir *Apps/Software* que permitan el adecuado funcionamiento de este controlador. Hay que recordar que ha sido diseñado en Apache Karaf en un ambiente JDK también.

Instalación de Maven para ONOS (En una carpeta de nombre *Applications*)

```
gns3@gns3:~$ cd ~
gns3@gns3:~$ mkdir Applications
gns3@gns3:~$ wget http://archive.apache.org/dist/maven/maven-3/3.3.9/binaries/ap
ache-maven-3.3.9-bin.tar.gz
--2021-05-31 16:44:16--  http://archive.apache.org/dist/maven/maven-3/3.3.9/bina
ries/apache-maven-3.3.9-bin.tar.gz
Resolving archive.apache.org (archive.apache.org)... 138.201.131.134, 2a01:4f8:1
72:2ec5::2
Connecting to archive.apache.org (archive.apache.org)|138.201.131.134|:80... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 8491533 (8.1M) [application/x-gzip]
Saving to: 'apache-maven-3.3.9-bin.tar.gz'

apache-maven-3.3.9- 100%[===================>]   8.10M  1.60MB/s    in 5.1s

2021-05-31 16:44:22 (1.60 MB/s) - 'apache-maven-3.3.9-bin.tar.gz' saved [8491533
/8491533]

gns3@gns3:~$ tar -zxvf apache-maven-3.3.9-bin.tar.gz -C ../Applications/
```

Figura 4-38 Instalación de Maven - ONOS
Fuente: *Autor*

[56] Maven: https://maven.apache.org/
[57] Buck: https://buck.build/ - Sistema para realizar *builds* de apps creado por Facebook.

En cuanto al ambiente JDK, se requiere de Java 8/11 y de establecer la variable de entorno JAVA_HOME tal como se muestra en la *Fig. 4-39.*

```
gns3@gns3:~$ sudo apt-get install software-properties-common -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  python3-software-properties software-properties-gtk
The following packages will be upgraded:
  python3-software-properties software-properties-common software-properties-gtk
3 upgraded, 0 newly installed, 0 to remove and 454 not upgraded.
Need to get 96.7 kB of archives.
After this operation, 0 B of additional disk space will be used.
Get:1 http://cd.archive.ubuntu.com/ubuntu bionic-updates/main amd64 software-propertie
s-common all 0.96.24.32.14 [10.1 kB]
```

```
gns3@gns3:~$ sudo add-apt-repository ppa:webupd8team/java -y
Get:1 http://ppa.launchpad.net/webupd8team/java/ubuntu bionic InRelease [15.4 kB]
Hit:2 http://cd.archive.ubuntu.com/ubuntu bionic InRelease
Hit:3 http://cd.archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:4 http://cd.archive.ubuntu.com/ubuntu bionic-backports InRelease
Hit:5 http://cd.archive.ubuntu.com/ubuntu bionic-security InRelease
Fetched 15.4 kB in 2s (6,458 B/s)
```

*Realizar un *apt-get update* luego del comando anterior

```
gns3@gns3:~$ sudo apt-get install openjdk-8-jdk -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ca-certificates-java fonts-dejavu-extra java-common libatk-wrapper-java
  libatk-wrapper-java-jni libgail-common libgail18 libgif7 libgtk2.0-0 libgtk2.0-bin
  libgtk2.0-common libice-dev libpthread-stubs0-dev libsm-dev libx11-6 libx11-dev
  libx11-doc libxau-dev libxau6 libxcb1-dev libxdmcp-dev libxt-dev
```

Figura 4-39 Instalación de JDK – ONOS
Fuente: *Autor*

Para verificar la correcta instalación de Java, usar el comando de la *Fig. 4-40*.

```
gns3@gns3:~$ java -version
openjdk version "1.8.0_292"
OpenJDK Runtime Environment (build 1.8.0_292-8u292-b10-0ubuntu1~18.04-b10)
OpenJDK 64-Bit Server VM (build 25.292-b10, mixed mode)
gns3@gns3:~$ update-alternatives --config java
There is only one alternative in link group java (providing /usr/bin/java): /usr/lib/jvm/
java-8-openjdk-amd64/jre/bin/java
Nothing to configure.
```

Figura 4-40 Comprobación de la versión instalada de JDK – ONOS
Fuente: *Autor*

Al completar la instalación de Java, es requerido establecer la variable de entorno JAVA_HOME.

```
gns3@gns3:~$ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
gns3@gns3:~$
gns3@gns3:~$ env | grep JAVA_HOME
JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
```

Figura 4-41 Estableciendo variable de entorno JAVA_HOME – ONOS
Fuente: *Autor*

Para la descarga de ONOS, se clonará el repositorio de dicho proyecto de *Onos Project*:

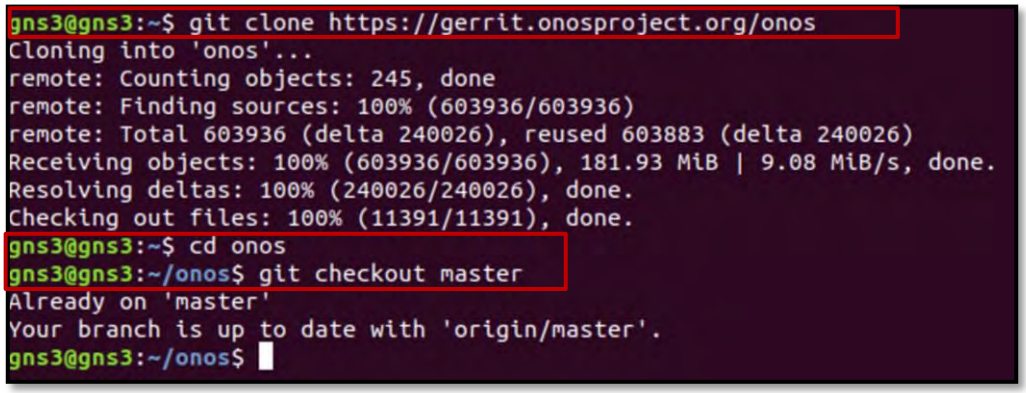


Figura 4-42 Clonación de Onos Project
Fuente: *Autor*

ONOS posee un archivo denominado *bash_profile* para configurar sus variables de entorno. En ese archivo se debe editar con **nano ~/.bashrc** e incluir al final la siguiente línea.

. ~/onos/tools/dev/bash_profile

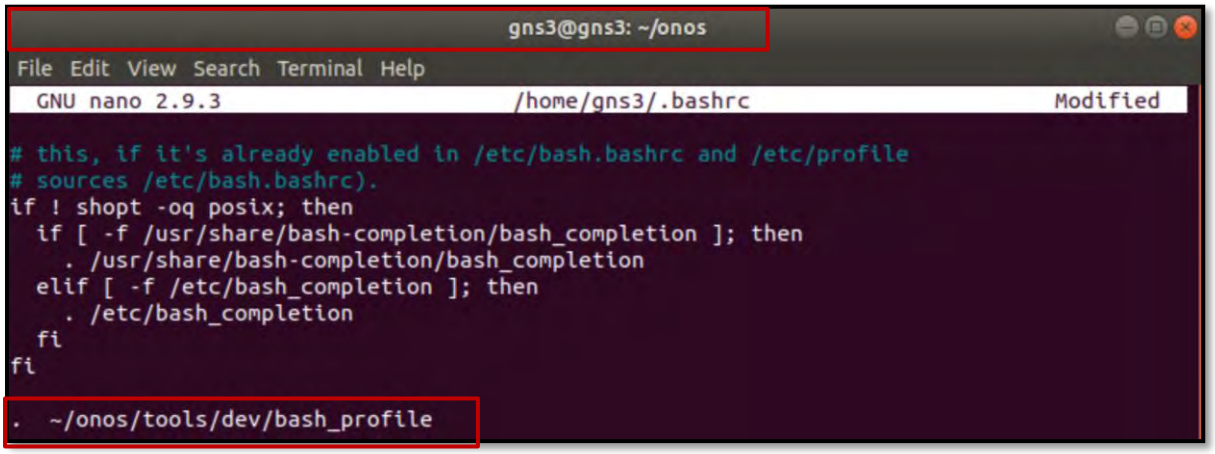


Figura 4-43 Edición de archivo bashrc - ONOS
Fuente: *Autor*

Una vez realizado ello, las variables de entorno de ONOS, Maven y Karaf deben estar definidas:

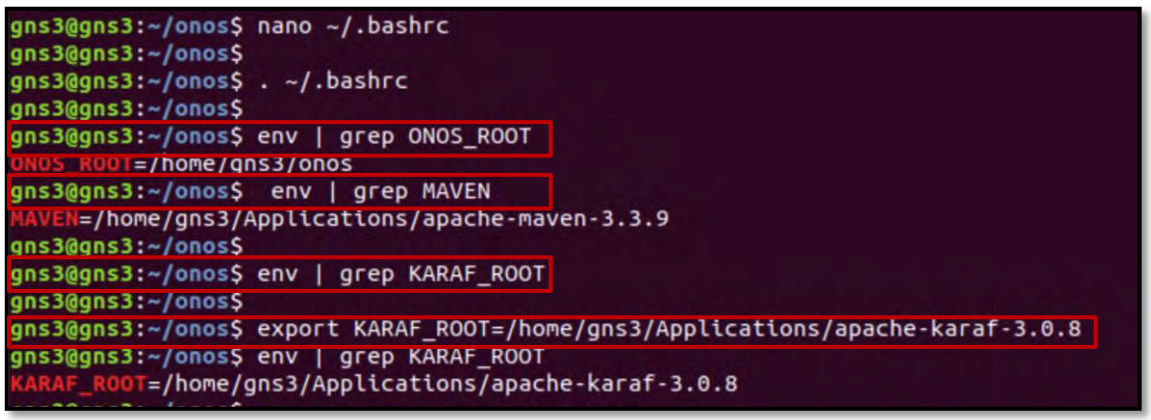


Figura 4-44 Variables de entorno de ONOS, Maven y Karaf
Fuente: *Autor*

En la etapa final de la configuración de ONOS se realiza un *Build* y *Deploy*.

Para ello, es necesario editar el *Features Repositories* con el proceso mostrado en la *Fig. 4-45*:

```
gns3@gns3:~/onos$ clear

gns3@gns3:~/onos$
gns3@gns3:~/onos$ cd ..
gns3@gns3:~$
gns3@gns3:~$ nano /home/gns3/Applications/apache-karaf-3.0.5/etc/org.apache.karaf.features.cfg
gns3@gns3:~$
gns3@gns3:~$
```

Añadir la siguiente línea con una coma antes en la ubicación mostrada:

mvn:org.onosproject/onos-features/1.10.0-SNAPSHOT/xml/features

```
gns3@gns3: ~
File  Edit  View  Search  Terminal  Help
     /home/gns3/Applications/apache-karaf-3.0.5/etc/org.apache.karaf.features.cfg      Modified

# If true, means stop bundles respecting the descend order of start level in a certain feature.
#
#respectStartLvlDuringFeatureUninstall=true

#
# Comma separated list of features repositories to register by default
#
$.2.6/xml/features,mvn:org.apache.karaf.features/spring/3.0.5/xml/features,mvn:org.onosproject/o$

#
# Comma separated list of features to install at startup
#
featuresBoot=config,standard,region,package,kar,ssh,management
```

Figura 4-45 Proceso previo a realizar el Build de ONOS
Fuente: *Autor*

ONOS está diseñado en *Bazel*[58], por lo que también debe ser instalado:

```
gns3@gns3:~$ sudo apt install apt-transport-https curl gnupg
[sudo] password for gns3:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  dirmngr gnupg-l10n gnupg-utils gpg gpg-agent gpg-wks-client gpg-wks-server
  gpgconf gpgsm gpgv libcurl4
Suggested packages:
  tor parcimonie xloadimage scdaemon
The following NEW packages will be installed:
  apt-transport-https
The following packages will be upgraded:
  curl dirmngr gnupg gnupg-l10n gnupg-utils gpg gpg-agent gpg-wks-client
  gpg-wks-server gpgconf gpgsm gpgv libcurl4
13 upgraded, 1 newly installed, 0 to remove and 439 not upgraded.
Need to get 2,530 kB of archives.
After this operation, 178 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

[58] Bazel: https://bazel.build/

```
gns3@gns3:~$ curl -fsSL https://bazel.build/bazel-release.pub.gpg | gpg --dearmo
r > bazel.gpg
gns3@gns3:~$
gns3@gns3:~$ sudo mv bazel.gpg /etc/apt/trusted.gpg.d/
gns3@gns3:~$
gns3@gns3:~$ echo "deb [arch=amd64] https://storage.googleapis.com/bazel-apt sta
ble jdk1.8" | sudo tee /etc/apt/sources.list.d/bazel.list
deb [arch=amd64] https://storage.googleapis.com/bazel-apt stable jdk1.8
gns3@gns3:~$
```

```
gns3@gns3:~$
gns3@gns3:~$ sudo apt update && sudo apt install bazel
Hit:1 http://cd.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 https://storage.googleapis.com/bazel-apt stable InRelease [2,256 B]
Get:3 http://cd.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Hit:4 http://ppa.launchpad.net/webupd8team/java/ubuntu bionic InRelease
Get:5 http://cd.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:6 http://cd.archive.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:7 https://storage.googleapis.com/bazel-apt stable/jdk1.8 amd64 Packages [6,0
77 B]
Fetched 260 kB in 2s (122 kB/s)
Reading package lists... 98%
```

Figura 4-46 Proceso de instalación de Bazel – ONOS
Fuente: *Autor*

Con el propósito de realizar el *Build* de ONOS, Maven requiere de la existencia de un archivo POM (*Project Object Model*), el cual es un archivo escrito en XML que indica el proceso general de instalación.

La ubicación del archivo POM de Maven es en el *path*:

/home/gns3/onos/tools/build

De ese *path* hay que copiar a:

/home/gns3/onos/

```
gns3@gns3:~/onos/tools/build$ cp pom.xml /home/gns3/onos
gns3@gns3:~/onos/tools/build$
```

Figura 4-47 Copia de archivo POM (Maven) – ONOS
Fuente: *Autor*

Realizar un *clean install* de ONOS:

```
gns3@gns3:~$ cd onos
gns3@gns3:~/onos$ ls
apps   CODE_OF_CONDUCT.md  Dockerfile  LICENSE.txt  pom.xml    README.md    utils
BUILD  core                docs        models       protocols  RELEASES.md  web
cli    deps                drivers     pipelines    providers  tools        WORKSPACE
gns3@gns3:~/onos$ mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building onos-base 2
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ onos-base ---
```

Figura 4-48 Clean Install – ONOS
Fuente: *Autor*

Cambie la dirección IP de ONOS a la asignada para el PoC (192.168.10.11/24), así como se debe agregar unos Apps para que se instalen una vez corra ONOS:

```
gns3@gns3:~/onos$ export ONOS_IP=192.168.10.11
gns3@gns3:~/onos$ export ONOS_NIC=192.168.10.*
gns3@gns3:~/onos$
gns3@gns3:~/onos$ export ONOS_APPS=drivers,openflow,proxyarp,mobility,fwd
gns3@gns3:~/onos$
```

Figura 4-49 Designación de IP a controlador – ONOS
Fuente: *Autor*

Arranque el servicio de ONOS (Preste atención a la versión de Bazel):

```
gns3@gns3:~/onos$
gns3@gns3:~/onos$ ok clean
ERROR: The project you're trying to build requires Bazel 3.7.2 (specified in /home/gns3/onos/.baz
elversion), but it wasn't found in /usr/bin.

You can install the required Bazel version via apt:
  sudo apt update && sudo apt install bazel-3.7.2

If this doesn't work, check Bazel's installation instructions for help:
  https://docs.bazel.build/versions/master/install-ubuntu.html
gns3@gns3:~/onos$ sudo apt update && sudo apt install bazel-3.7.2
Hit:1 https://storage.googleapis.com/bazel-apt stable InRelease
Hit:2 http://cd.archive.ubuntu.com/ubuntu bionic InRelease
Hit:3 http://ppa.launchpad.net/webupd8team/java/ubuntu bionic InRelease
Get:4 http://cd.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:5 http://cd.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:6 http://cd.archive.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Fetched 252 kB in 2s (116 kB/s)
Reading package lists... 6%
```

Figura 4-50 Instalación de la versión adecuada de Bazel – ONOS
Fuente: *Autor*

Corrigiendo el error de la versión de Bazel, tomará un tiempo el arranque inicial (***ok clean***).

Debe tener instalado *Python* de igual manera:

sudo apt-get install ssh git curl zip unzip python python3 bzip2
sudo apt-get install pkg-config g++ zlib1g-dev

```
gns3@gns3:~/onos$ ok clean
WARNING: Running Bazel server needs to be killed, because the startup options are different.
Starting local Bazel server and connecting to it...
INFO: Analyzed target //:onos-local (1665 packages loaded, 52539 targets configured).
INFO: Found 1 target...
INFO: From Building utils/misc/libonlab-misc-native-class.jar (203 source files):
utils/misc/src/main/java/org/onlab/util/RichComparable.java:25: warning: [ComparableType] Type o
 Comparable (T) is not the same as implementing class (RichComparable<T>).
public interface RichComparable<T> extends Comparable<T> {
       ^
    (see https://errorprone.info/bugpattern/ComparableType)
[814 / 1,318] .../app:onos-apps-openstacknetworking-app_swagger_java; 0s linux-sandbox
```

Figura 4-51 Arranque del controlador ONOS (ok clean) en carpeta ONOS
Fuente: *Autor*

Al correr ONOS, se podrá acceder a la configuración y a la GUI de este controlador.

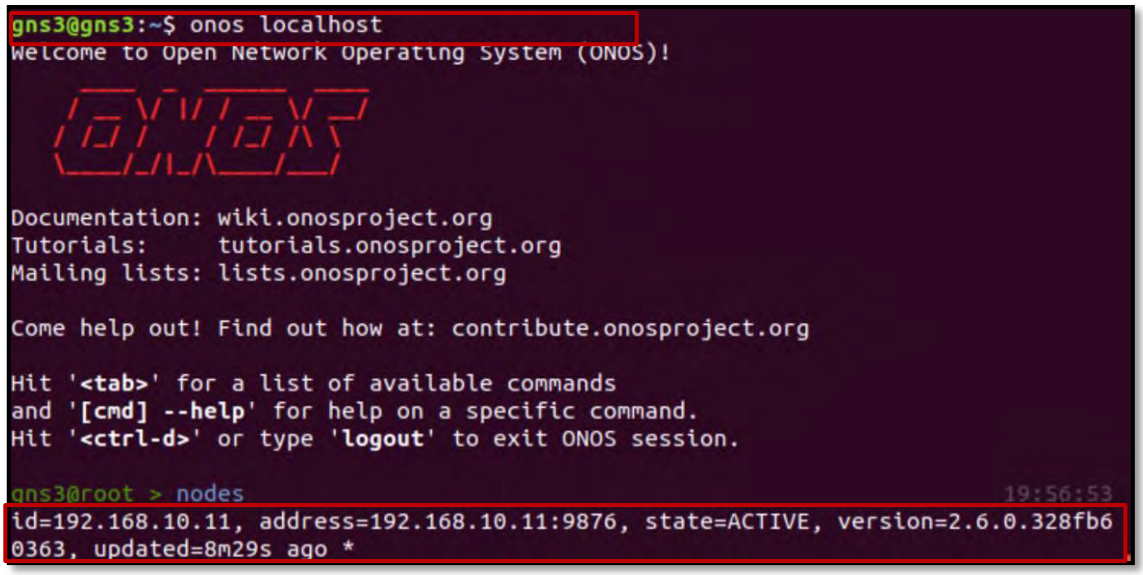


Figura 4-52 CLI - ONOS
Fuente: *Autor*

Para habilitar el *front-end* de ONOS, se debe activar la App GUI2 en el CLI de ONOS:

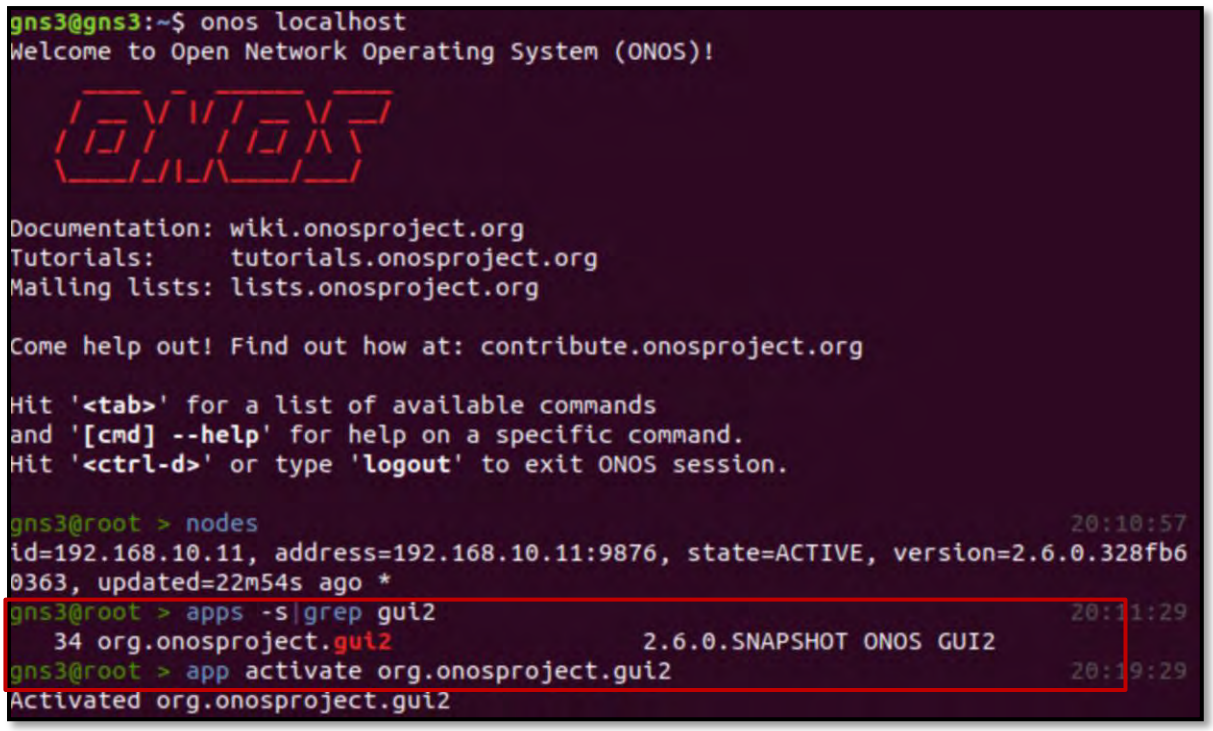


Figura 4-53 Activación de GUI2 en CLI - ONOS
Fuente: *Autor*

Una vez activa la GUI, es posible entrar de forma gráfica al controlador ONOS usando las siguientes credenciales:

onos (como usuario) y rocks (como contraseña)

La URL de ingreso es:

Dir-IP-ONOS:8181/onos/ui/login.html

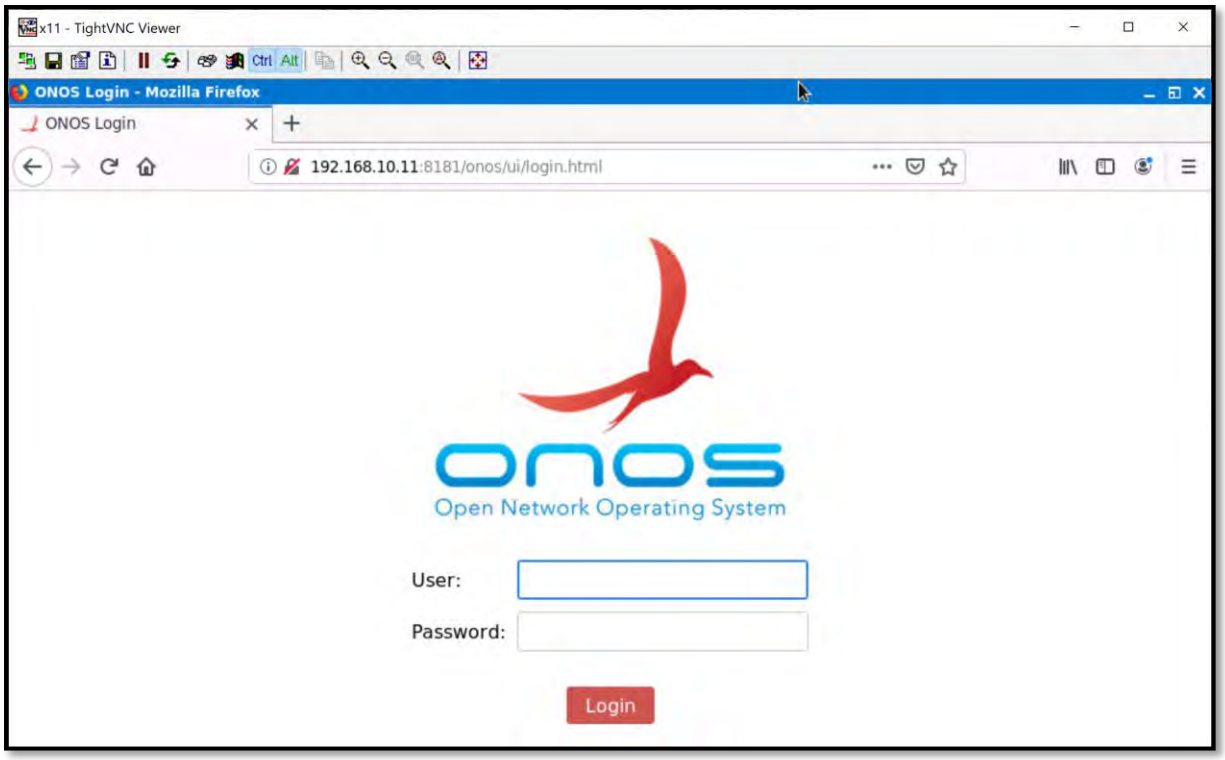
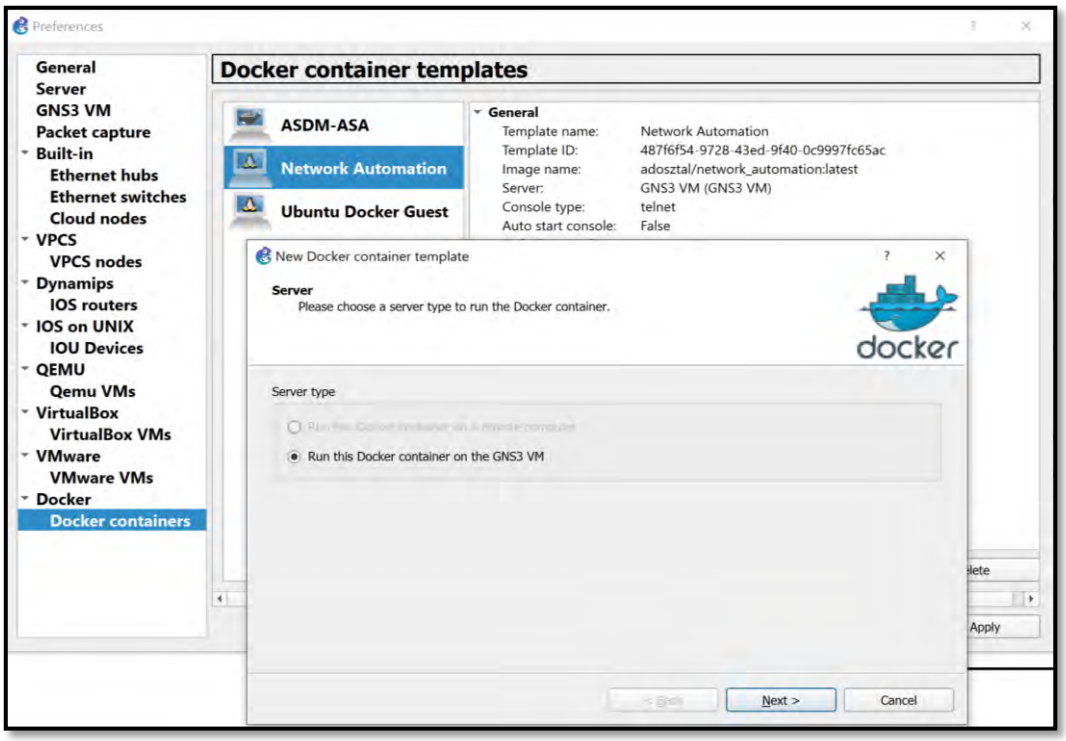


Figura 4-54 Front-End de ONOS
Fuente: *Autor*

Si por alguna razón no se realiza con éxito el *Build*, se puede importar un *Docker* de ONOS previamente diseñado e instarlo en GNS3-VM.

Para tener ONOS mediante un Docker, se instalará en GNS3 dicho *Docker Container* creado por Muhammad Luthfias. Un beneficio de hacerlo así, es que ONOS será un *appliance* de GNS3.



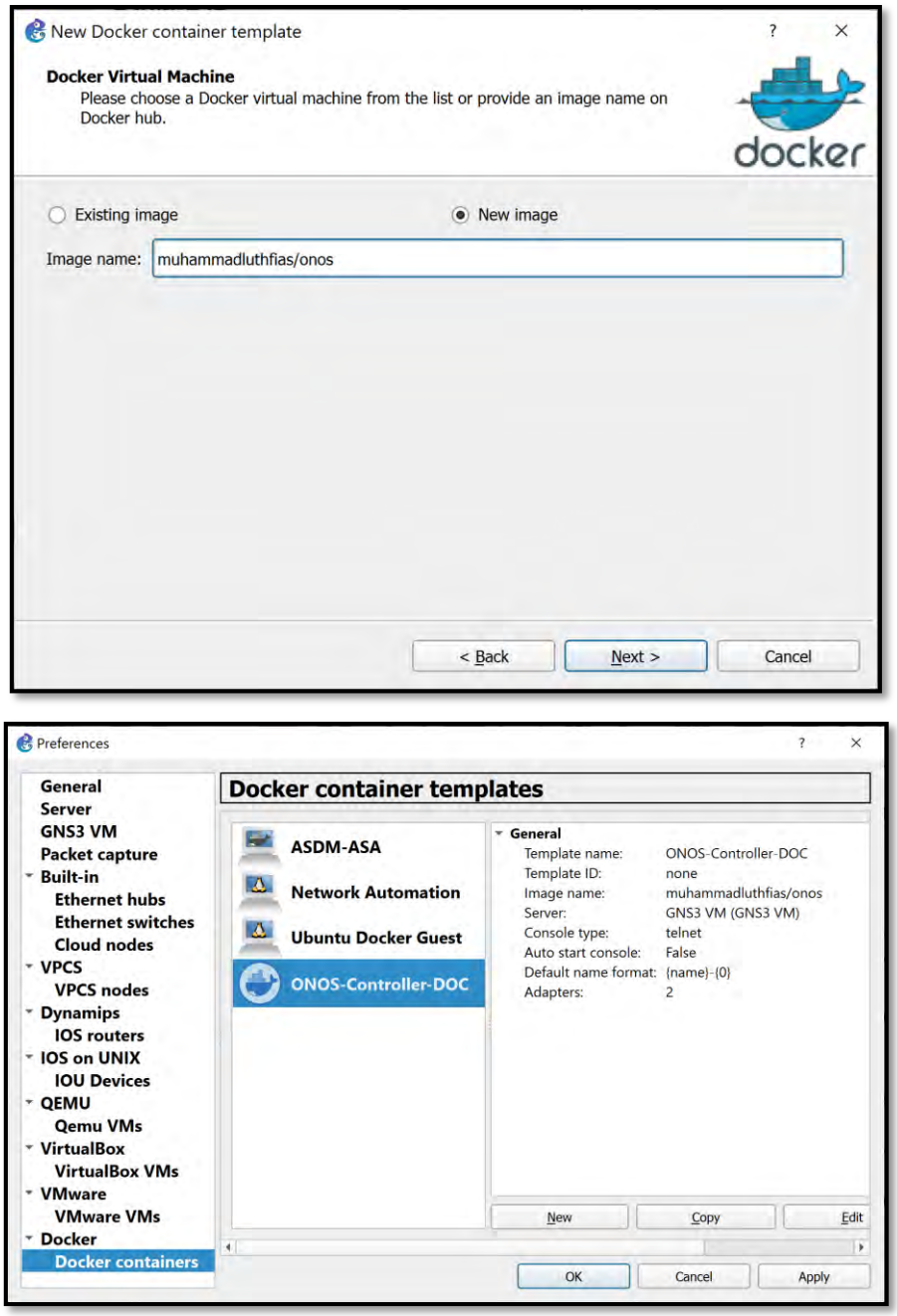


Figura 4-55 Instalación de ONOS como Docker en GNS3-VM
Fuente: *Autor*

La primera vez que se arrastre el Controlador a la zona de emulación de GNS3, empezará el proceso de *pulling* del *Docker* desde el repositorio (*Docker Hub*).

Es necesario editar el direccionamiento estático de ONOS según la topología del PoC dando clic secundario sobre el dispositivo en *Edit Config* (*Fig. 4-56*).

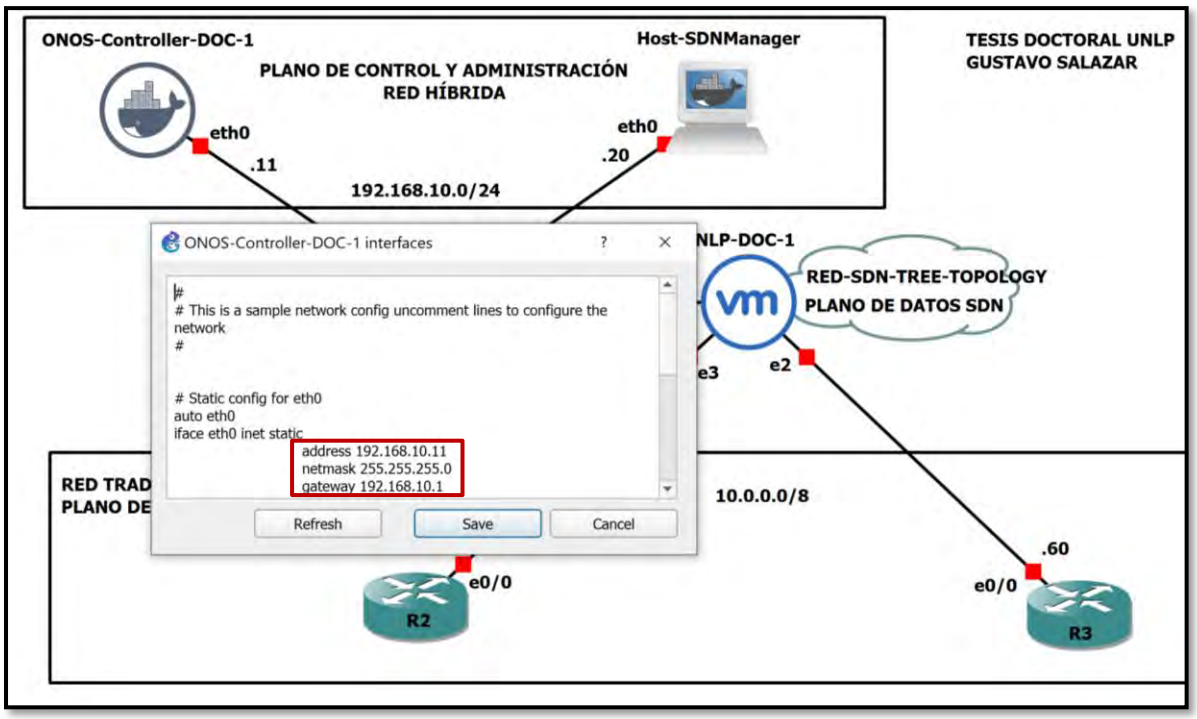


Figura 4-56 Cambio de Direccionamiento en Controlador ONOS
Fuente: *Autor*

Si la instalación concluyó exitosamente, será posible ingresar a la GUI de ONOS indicada en la URL anteriormente con los mismos usuarios y contraseña (**onos/rocks**)

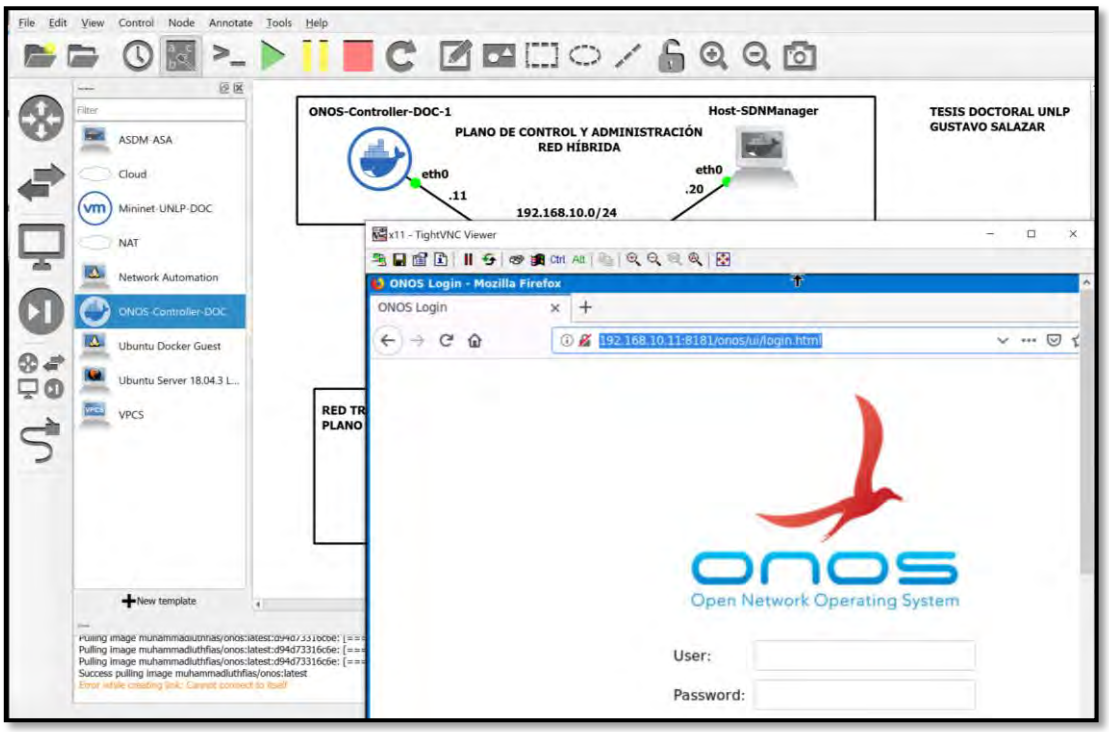


Figura 4-57 Front-End de ONOS – Instalación en Docker de GNS3
Fuente: *Autor*

Para tener *hosts* disponibles en la PoC y acceder a la GUI de ODL/ONOS, se usarán *Dockers* de Ubuntu[59], así como un Docker especial creado por la comunidad de GNS3 para entornos de seguridad[60]. El *Docker* de Ubuntu se puede descargar del *Marketplace* de GNS3. No está por demás decir que los *dockers* no son persistentes, por lo que su configuración se perderá en cada reinicio del emulador:

Figura 4-58 Docker de Ubuntu para GNS3-VM
Fuente: *Autor*

La primera vez que se arrastre este *applicance* a la zona de trabajo de GNS3, se hará un proceso de descarga (*pulling*), luego del cual se tendrá un *host* CLI de Ubuntu:

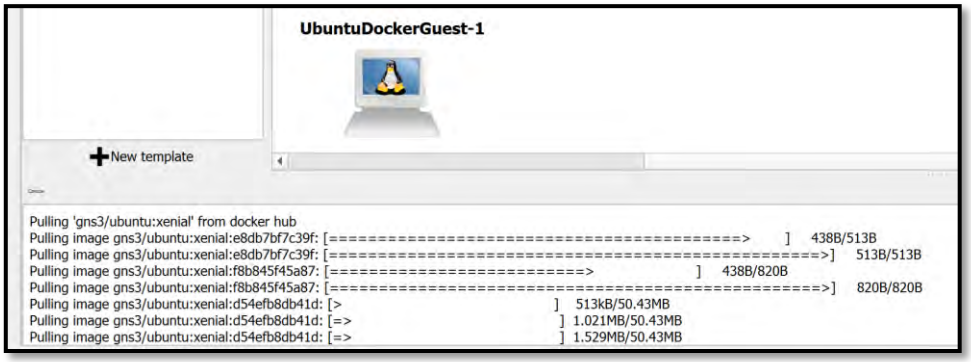
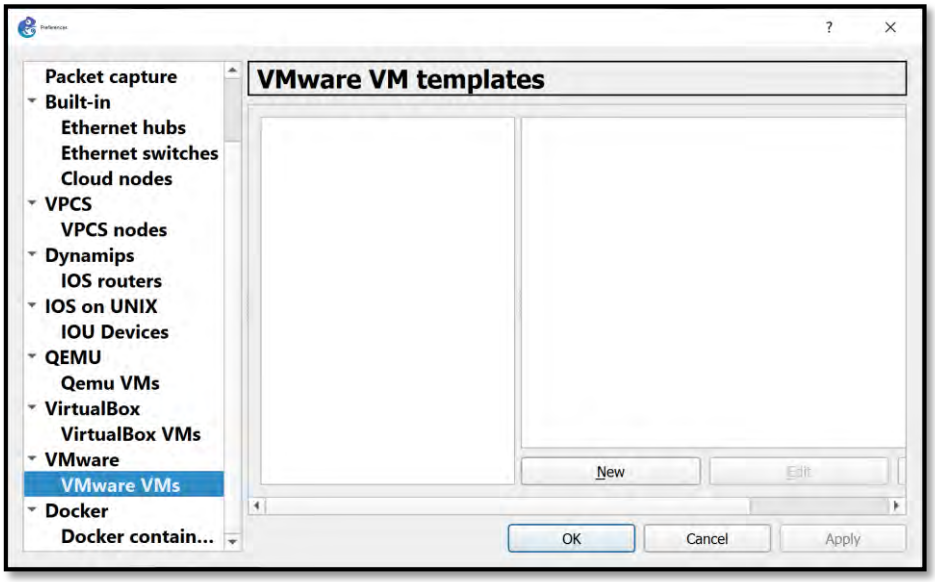


Figura 4-59 Pull de descarga de Docker de Ubuntu para GNS3-VM
Fuente: *Autor*

Continuando con el PoC, para añadir en GNS3 a Mininet, se la agregará como una VM de VMWare adicional siguiendo los pasos generales desde Edit-Preference-VMWare mostrados en la *Fig. 4-60*:



[59] Ubuntu Docker – GNS3 Marketplace: https://www.gns3.com/marketplace/appliances/ubuntu
[60] Docker creado por la comunidad GNS3 para entornos de seguridad (ASA): https://www.youtube.com/watch?v=QIuZdAcLXs0

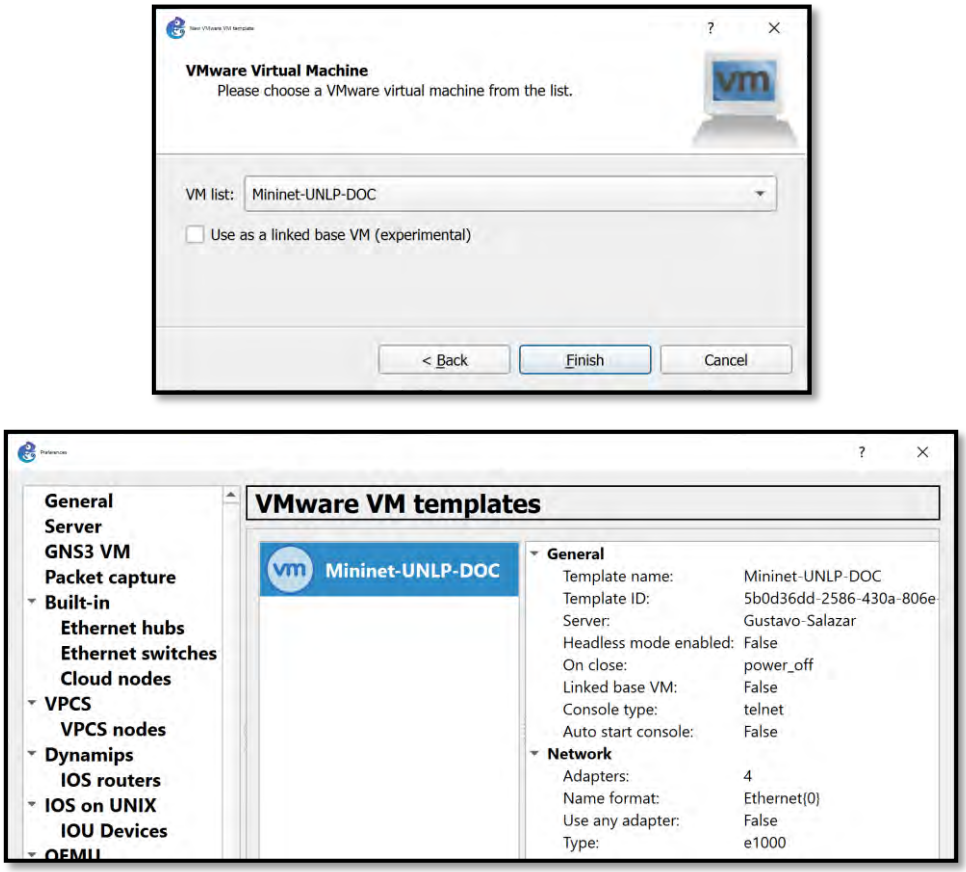


Figura 4-60 Proceso de Carga de Mininet a GNS3-VM con 4 interfaces
Fuente: *Autor*

Es importante vincular las 4 interfaces de Mininet con las interfaces virtuales generadas por VMWare con el fin de tener conectividad entre el entorno de GNS3-VM, la máquina *host* y la red emulada en Mininet. Dichas interfaces deben estar en la misma red del plano de control y plano de datos del PoC.

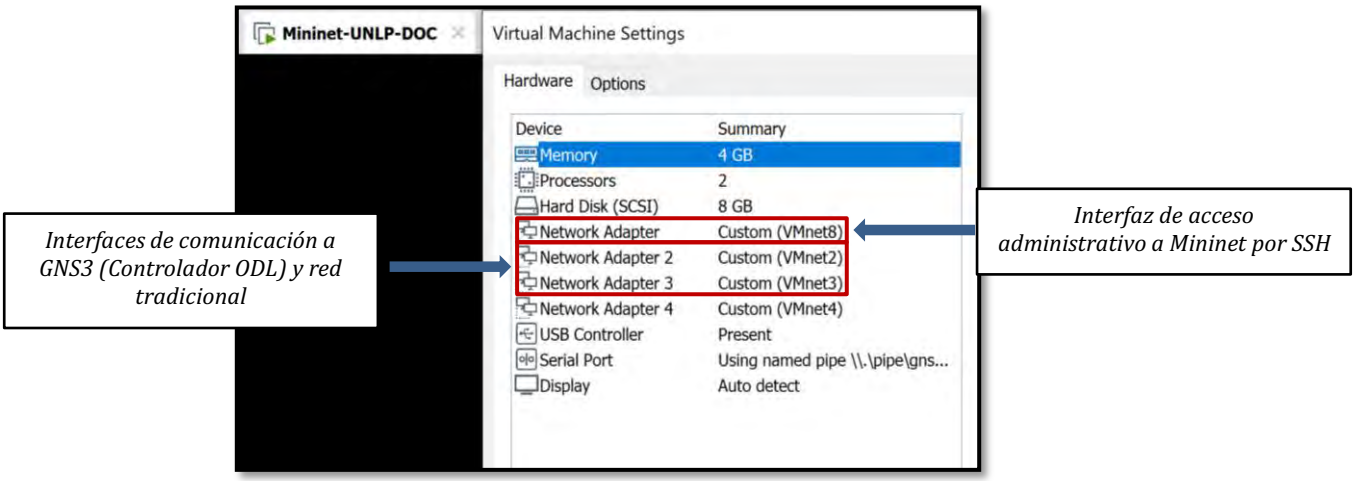


Figura 4-61 Vinculación de interfaces vmnet a GNS3-VM y Mininet
Fuente: *Autor*

- Dir. IP Plano de Control en el PoC ODL: 192.168.10.0/24 (vmnet 2)
- Dir. IP Plano de Datos SDN y Tradicional: 10.0.0.0/8 (vmnet 3 – vmnet 4)

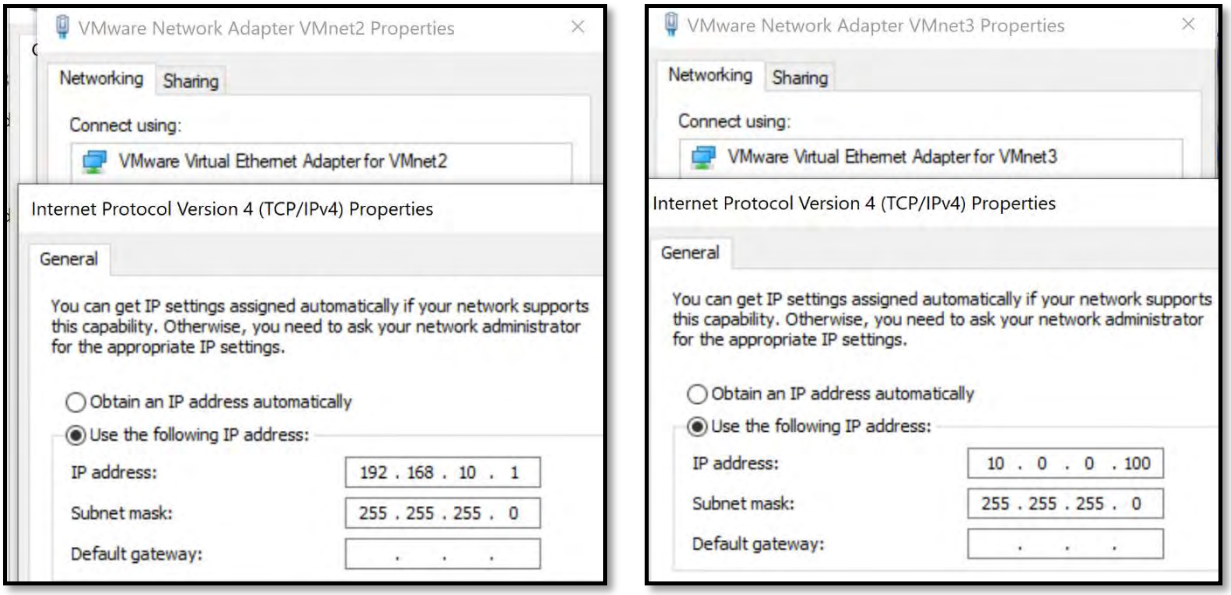


Figura 4-62 Dir. IP interfaces vmnet para PoC ODL
Fuente: *Autor*

Ya con todos los elementos instalados para el PoC, es posible plantear las topologías de emulación y comprobar la vinculación del entorno SDN con diversos controladores y el mundo del *networking* tradicional.

4.3.2 Topología y Emulación de *OpenSDN Híbrida*

La topología usada para el PoC de OpenSDN y Redes tradicionales en Mininet se indica en las *Fig. 4-63 y Fig. 4-64*, la cual fue diseñada con los siguientes dispositivos:

- OVS – *Open vSwitches*
- Controlador ODL (*OpenDayLight*) – Escenario 1 y Controlador ONOS – Escenario 2
- Switches L2 IOU/IOL y Routers Cisco L3 IOU/IOL

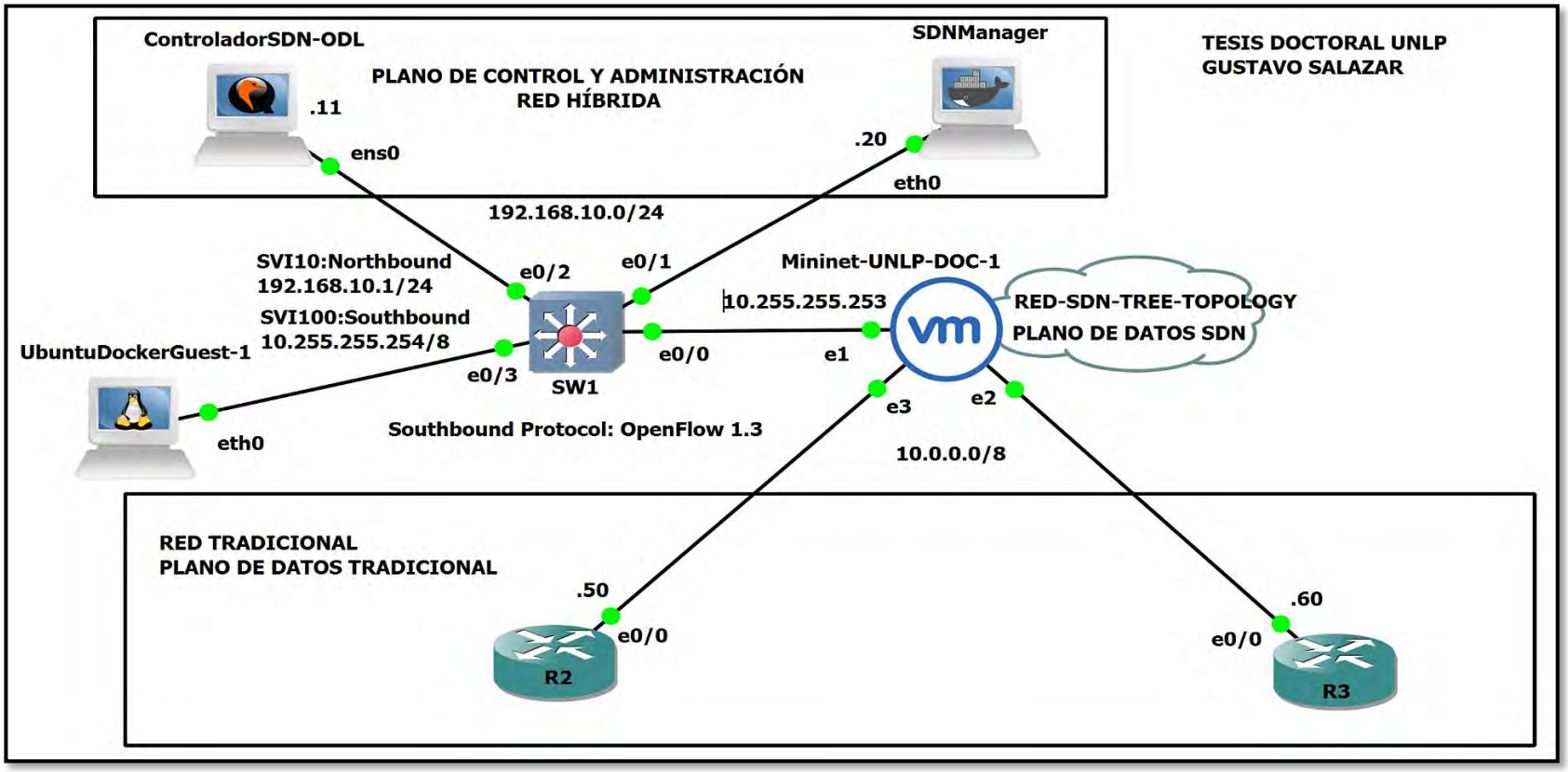


Figura 4-63 Topología PoC SDN Híbrida – Controlador ODL – Escenario 1
Fuente: Autor

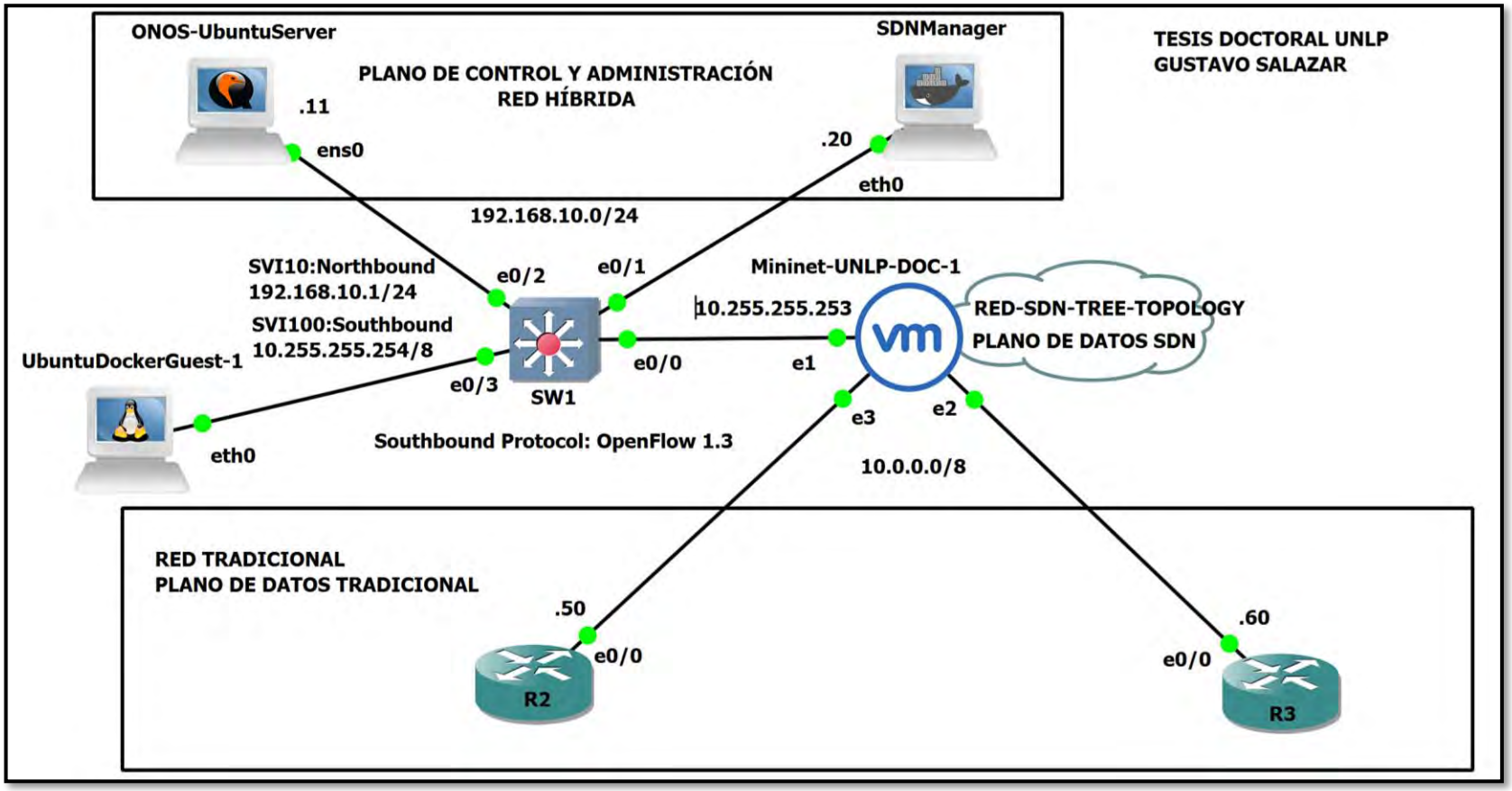


Figura 4-64 Topología PoC SDN Híbrida – Controlador ONOS – Escenario 2
Fuente: *Autor*

Para la ejecución del PoC en sus dos escenarios, se ingresa a Mininet mediante SSH utilizando su dir. IP de administración (vmnet 8 – 192.168.85.149/24) y se activarán las interfaces eth1 y eth2 correspondientes a las interfaces hacia el plano de control y plano de datos respectivamente. Se recomienda inicializar primero a Mininet, luego el controlador SDN y el plano de administración y finalmente el plano de datos.

```
mininet@mininet-vm:~$
mininet@mininet-vm:~$ sudo ip link set dev eth1 up
mininet@mininet-vm:~$ sudo ip link set dev eth2 up
mininet@mininet-vm:~$ sudo ip link set dev eth3 up
mininet@mininet-vm:~$ sudo ip link set dev ovs-system up
mininet@mininet-vm:~$ sudo ip link set dev s1 up
mininet@mininet-vm:~$ sudo ip link set dev s2 up
mininet@mininet-vm:~$ sudo ip link set dev s3 up
mininet@mininet-vm:~$
mininet@mininet-vm:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:d0:e0:2c brd ff:ff:ff:ff:ff:ff
    inet 192.168.85.149/24 brd 192.168.85.255 scope global eth0
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:d0:e0:36 brd ff:ff:ff:ff:ff:ff
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-system state UP group default qlen 1000
    link/ether 00:0c:29:d0:e0:40 brd ff:ff:ff:ff:ff:ff
5: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:d0:e0:4a brd ff:ff:ff:ff:ff:ff
6: ovs-system: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1
    link/ether 0a:29:f1:9e:bf:2c brd ff:ff:ff:ff:ff:ff
7: s1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1
    link/ether 00:0c:29:d0:e0:40 brd ff:ff:ff:ff:ff:ff
8: s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1
    link/ether 9a:ef:9e:31:53:46 brd ff:ff:ff:ff:ff:ff
9: s2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1
    link/ether 0e:2b:05:90:1a:40 brd ff:ff:ff:ff:ff:ff
```

Figura 4-65 Activación de interfaces en Mininet dentro de GNS3-VM para PoC ODL/ONOS
Fuente: *Autor*

Mininet debe alcanzar tanto al controlador SDN como al plano de Administración y al plano de datos físico tradicional tal como se muestra en la *Fig. 4-66*:

```
mininet@mininet-vm:~$ ping 192.168.10.11
PING 192.168.10.11 (192.168.10.11) 56(84) bytes of data.
64 bytes from 192.168.10.11: icmp_seq=1 ttl=128 time=3.21 ms
64 bytes from 192.168.10.11: icmp_seq=2 ttl=128 time=6.23 ms
64 bytes from 192.168.10.11: icmp_seq=3 ttl=128 time=5.66 ms
64 bytes from 192.168.10.11: icmp_seq=4 ttl=128 time=6.55 ms
^C
--- 192.168.10.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3014ms
rtt min/avg/max/mdev = 3.215/5.418/6.554/1.313 ms
mininet@mininet-vm:~$
mininet@mininet-vm:~$ ping 192.168.10.20
PING 192.168.10.20 (192.168.10.20) 56(84) bytes of data.
64 bytes from 192.168.10.20: icmp_seq=1 ttl=128 time=1.83 ms
64 bytes from 192.168.10.20: icmp_seq=2 ttl=128 time=4.85 ms
64 bytes from 192.168.10.20: icmp_seq=3 ttl=128 time=4.44 ms
64 bytes from 192.168.10.20: icmp_seq=4 ttl=128 time=4.74 ms
^C
--- 192.168.10.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3010ms
rtt min/avg/max/mdev = 1.830/3.969/4.859/1.247 ms
```

```
mininet@mininet-vm:~$ ping 10.0.0.50
PING 10.0.0.50 (10.0.0.50) 56(84) bytes of data.
64 bytes from 10.0.0.50: icmp_seq=1 ttl=128 time=2.06 ms
64 bytes from 10.0.0.50: icmp_seq=2 ttl=128 time=5.57 ms
64 bytes from 10.0.0.50: icmp_seq=3 ttl=128 time=5.14 ms
64 bytes from 10.0.0.50: icmp_seq=4 ttl=128 time=5.22 ms
^C
--- 10.0.0.50 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3012ms
rtt min/avg/max/mdev = 2.065/4.500/5.570/1.417 ms
mininet@mininet-vm:~$
mininet@mininet-vm:~$
mininet@mininet-vm:~$ ping 10.0.0.60
PING 10.0.0.60 (10.0.0.60) 56(84) bytes of data.
64 bytes from 10.0.0.60: icmp_seq=1 ttl=128 time=2.02 ms
64 bytes from 10.0.0.60: icmp_seq=2 ttl=128 time=5.21 ms
64 bytes from 10.0.0.60: icmp_seq=3 ttl=128 time=5.32 ms
64 bytes from 10.0.0.60: icmp_seq=4 ttl=128 time=5.14 ms
^C
--- 10.0.0.60 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3010ms
rtt min/avg/max/mdev = 2.029/4.429/5.323/1.388 ms
```

Figura 4-66 Conectividad de Mininet hacia el Plano de Control-Administración dentro de GNS3-VM en PoC ODL/ONOS
Fuente: *Autor*

Para que los *pings* sean exitosos, en Mininet se debe configurar las interfaces eth1 con la Dir. IP 10.255.255.253/8 y añadir una ruta por defecto apuntando al SW Multicapa (**sudo route add default gw 10.255.255.254)**

Para habilitar las interfaces físicas de Mininet e interactúen con los *routers* del PoC, se debe habilitar las interfaces de *Open vSwitch* de la siguiente manera:

```
mininet@mininet-vm:~$ sudo ovs-vsctl add-port s1 eth3
mininet@mininet-vm:~$ sudo ovs-vsctl add-port s2 eth2
mininet@mininet-vm:~$
mininet@mininet-vm:~$ sudo ovs-vsctl show
063b1597-ee6f-47f2-b09f-cf8e06c009e0
    Bridge "s3"
        Controller "tcp:192.168.10.11:6653"
            is_connected: true
        Controller "ptcp:6656"
        fail_mode: secure
        Port "s3-eth3"
            Interface "s3-eth3"
        Port "s3-eth2"
            Interface "s3-eth2"
        Port "s3-eth1"
            Interface "s3-eth1"
        Port "s3"
            Interface "s3"
```

Figura 4-67 Habilitación de Interfaces OVS en PoC ODL/ONOS
Fuente: *Autor*

Al existir conectividad entre Mininet y los planos de datos y control, se debe inicializar al controlador ODL activando *Apache Karaf* para el Escenario 1. Para el Escenario 2 basta que ONOS esté corriendo (**ok clean**).

```
gns3@gns3:~$ echo 'export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre' >> ~/.bashrc
gns3@gns3:~$
gns3@gns3:~$ source ~/.bashrc
gns3@gns3:~$
gns3@gns3:~$ echo $JAVA_HOME
/usr/lib/jvm/java-8-openjdk-amd64/jre
gns3@gns3:~$
gns3@gns3:~$ export TERM=xterm-color
gns3@gns3:~$
gns3@gns3:~$ cd /usr/local/karaf/karaf-0.7.3/bin/
gns3@gns3:/usr/local/karaf/karaf-0.7.3/bin$ ls
aaa-cli-jar.jar                contrib                  karaf               shell      status.bat
client                         custom_shard_config.txt  karaf.bat           shell.bat  stop
client.bat                     idmtool                  setenv              start      stop.bat
configure-cluster-ipdetect.sh  instance                 setenv.bat          start.bat  upgrade
configure_cluster.sh           instance.bat             set_persistence.sh  status
gns3@gns3:/usr/local/karaf/karaf-0.7.3/bin$ sudo ./karaf
[sudo] password for gns3:
karaf: JAVA_HOME not set; results may vary
Apache Karaf starting up. Press Enter to open the shell now...
100% [========================================================================]
Karaf started in 64s. Bundle stats: 376 active, 377 total
```

Figura 4-68 Arranque de Controlador en PoC ODL
Fuente: *Autor*

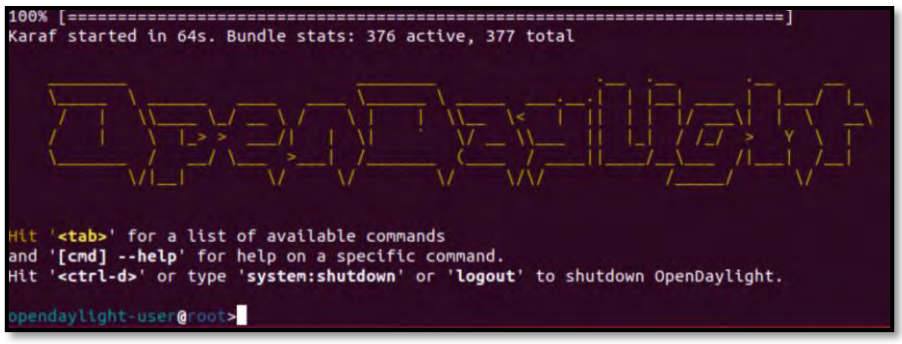


Figura 4-69 Activación de ODL (Apache Karaf) en PoC ODL
Fuente: *Autor*

Ahora, se creará la red dentro de Mininet (topología SDN) apuntando a la Dir. IP del controlador ODL (192.168.10.11/24), la cual corresponde a una arquitectura tipo *Tree* de 2 niveles, con 2 *Hosts* en cada *switch* de acceso.

```
mininet@mininet-vm:~$
mininet@mininet-vm:~$ sudo mn --topo=tree,depth=2,fanout=2 --mac --controller=remote,ip=192.168.10.11,por
t=6633 --switch ovs,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>
```

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=4379>
<Host h2: h2-eth0:10.0.0.2 pid=4382>
<Host h3: h3-eth0:10.0.0.3 pid=4385>
<Host h4: h4-eth0:10.0.0.4 pid=4388>
<OVSSwitch{'protocols': 'OpenFlow13'} s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=4394>
<OVSSwitch{'protocols': 'OpenFlow13'} s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=4397>
<OVSSwitch{'protocols': 'OpenFlow13'} s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=4400>
<RemoteController{'ip': '192.168.10.11', 'port': 6633} c0: 192.168.10.11:6633 pid=4373>
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

Figura 4-70 Arranque de Mininet en PoC ODL/ONOS
Fuente: *Autor*

4.3.3 Resultados de Emulación OpenSDN Híbrida

Escenario 1: Controlador *OpenDayLight*

Según la topología planteada en ***4.3.2 Topología y Emulación de OpenSDN Híbrida***, la ejecución del PoC fue exitosa, tal como se aprecia en las siguientes figuras:

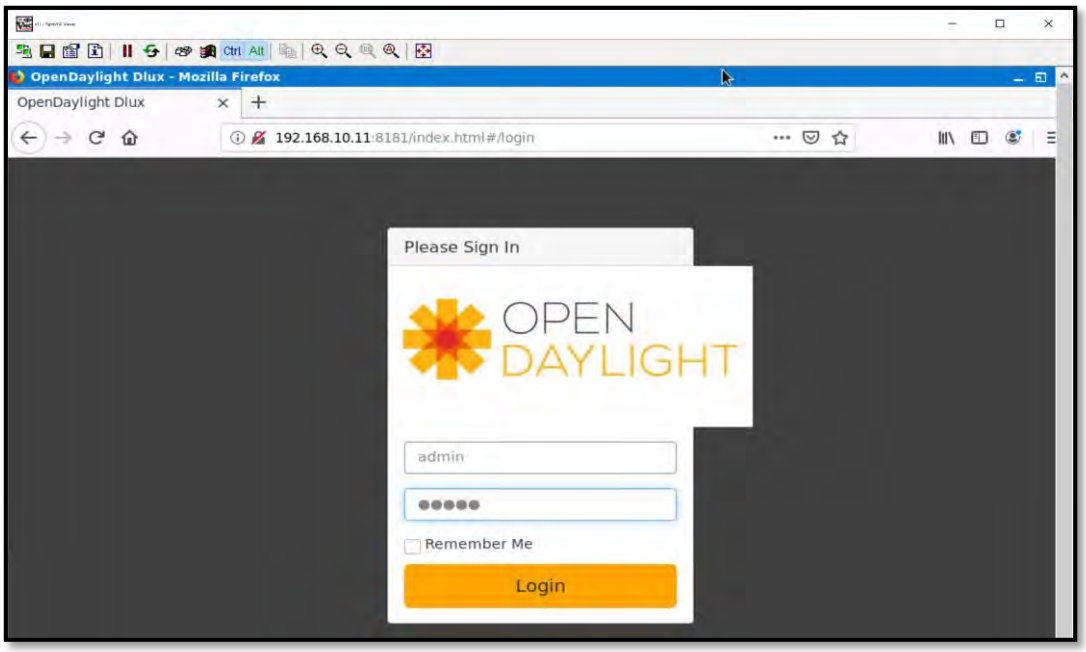


Figura 4-71 Ingreso al GUI de Controlador en PoC ODL
Fuente: *Autor*

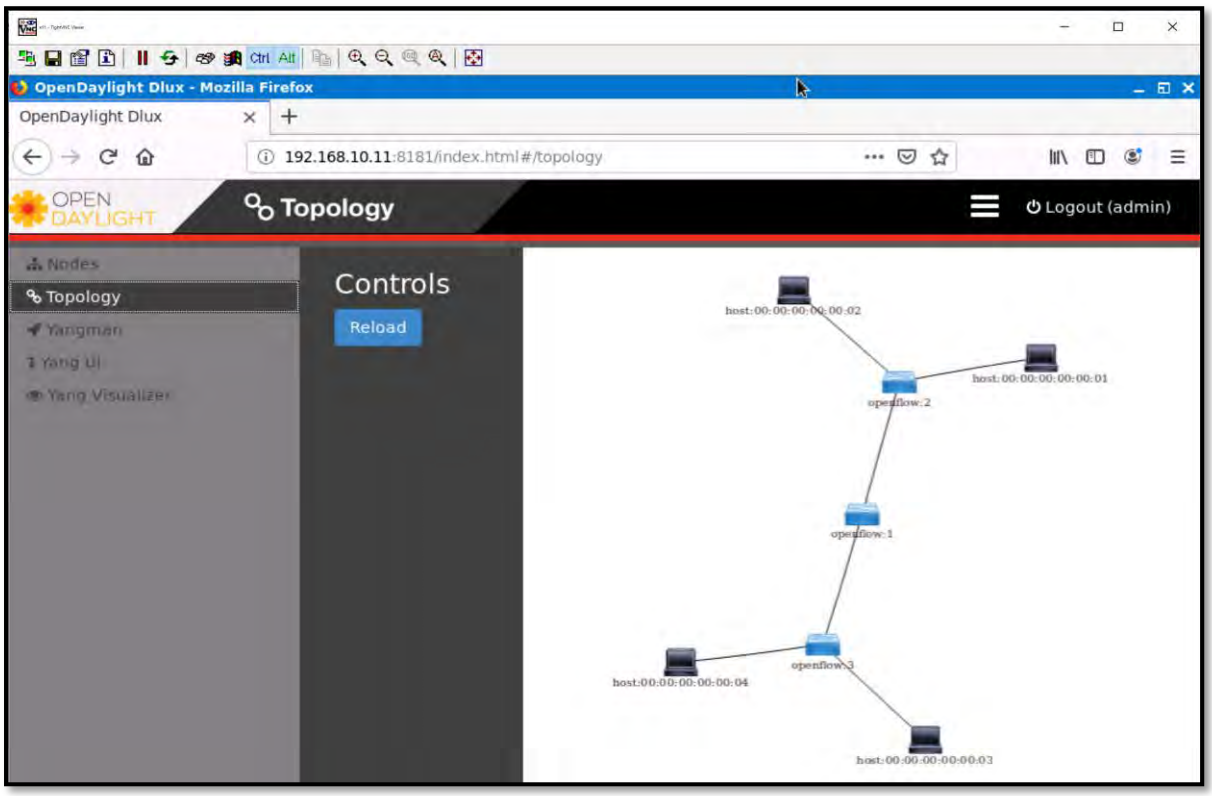


Figura 4-72 Topología descubierta por Controlador en PoC ODL
Fuente: *Autor*

```
mininet>
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=1440>
<Host h2: h2-eth0:10.0.0.2 pid=1443>
<Host h3: h3-eth0:10.0.0.3 pid=1446>
<Host h4: h4-eth0:10.0.0.4 pid=1449>
<OVSSwitch{'protocols': 'OpenFlow13'} s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1455>
<OVSSwitch{'protocols': 'OpenFlow13'} s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=1458>
<OVSSwitch{'protocols': 'OpenFlow13'} s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=1461>
<RemoteController{'ip': '192.168.10.11', 'port': 6633} c0: 192.168.10.11:6633 pid=1434>
mininet>
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.178 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.194 ms
^C
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.178/0.186/0.194/0.008 ms
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.783 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.155 ms
^C
```

Figura 4-73 Prueba de conectividad en Hosts de Mininet en PoC ODL
Fuente: *Autor*

ODL cuenta con herramientas gráficas para visualizar los nodos (*OVSs* y enlaces)

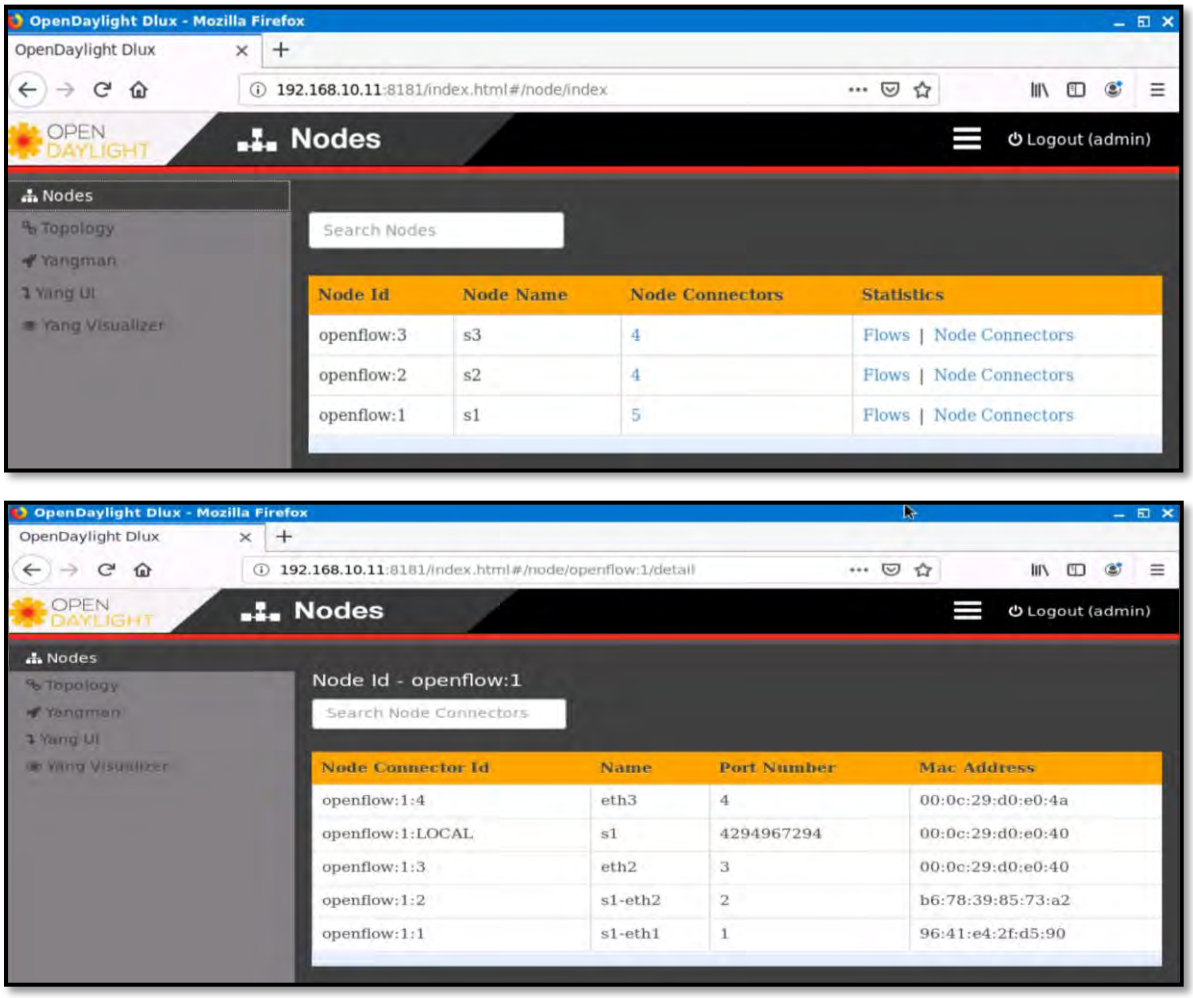


Figura 4-74 Herramientas de visualización de nodos y enlaces en PoC ODL
Fuente: *Autor*

Una de las ventajas de usar ODL es la variedad de APIs con las que se puede interactuar con la infraestructura. Cabe decir que no todas pueden funcionar, pues depende del tipo de plano de datos que esté controlando *OpenDayLight*:

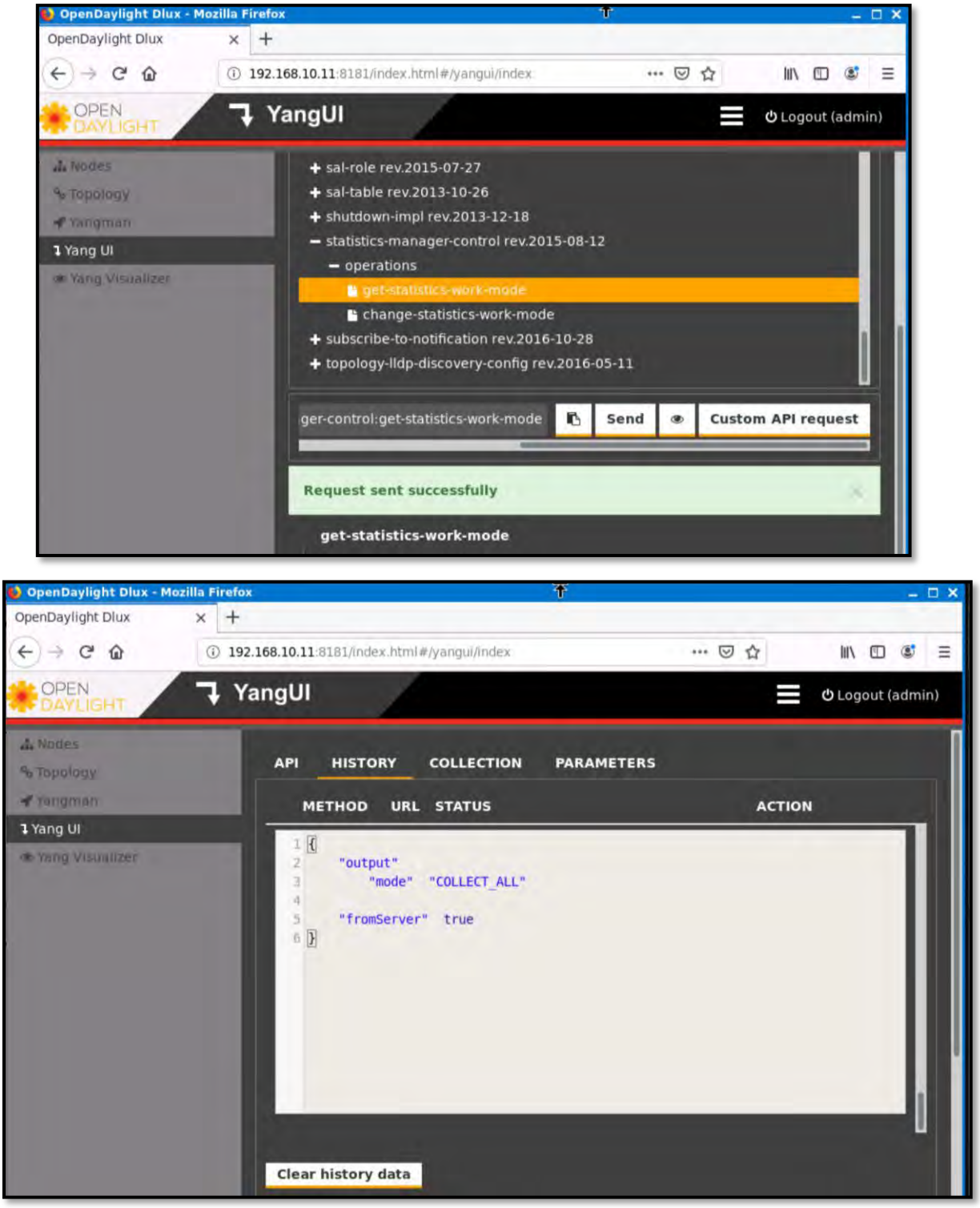


Figura 4-75 Uso de YangUI en PoC ODL: get statics-work-mode
Fuente: *Autor*

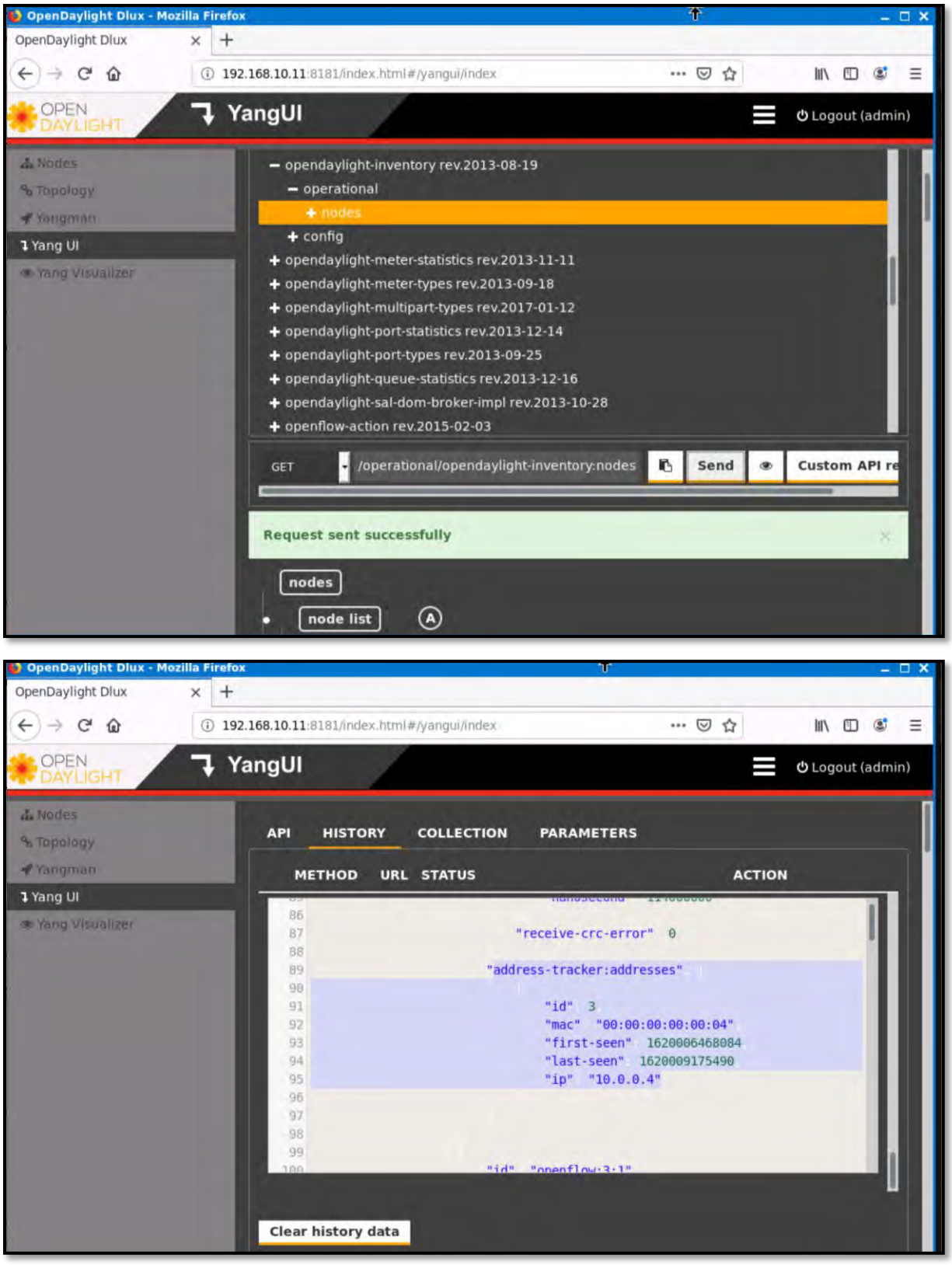


Figura 4-76 Uso de YangUI en PoC ODL: OpenDayLight Inventory (nodes)
Fuente: *Autor*

Además de contar con APIs que vienen por defecto en ODL, es factible usar *Yangman* para crear APIs personalizadas, así como editar los mensajes GET/SET utilizados en la interacción entre Controlador y Plano de datos. Un ejemplo se aprecia en la *Fig. 4-77* para obtener el inventario de la red:

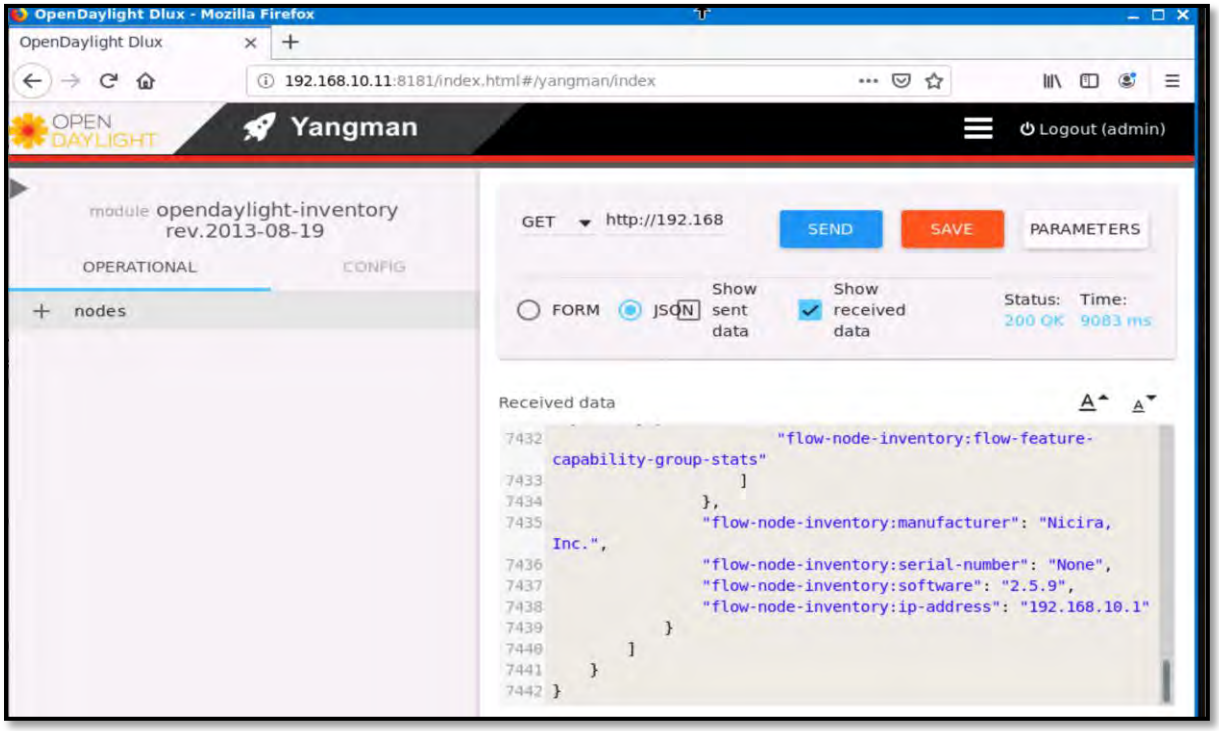


Figura 4-77 Uso de Yangman en PoC ODL: Personalizando API Inventory (nodes)
Fuente: *Autor*

Escenario 2: Controlador ONOS

Al igual que en el Escenario 1, según la topología planteada en ***4.3.2 Topología y Emulación de OpenSDN Híbrida***, la ejecución del PoC fue exitosa, tal como se aprecia en las siguientes figuras:

```
mininet@mininet-vm:~$ sudo mn --topo=tree,depth=2,fanout=2 --mac --controller=remote,ip=192
.168.10.11,port=6653 --switch ovs,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
```

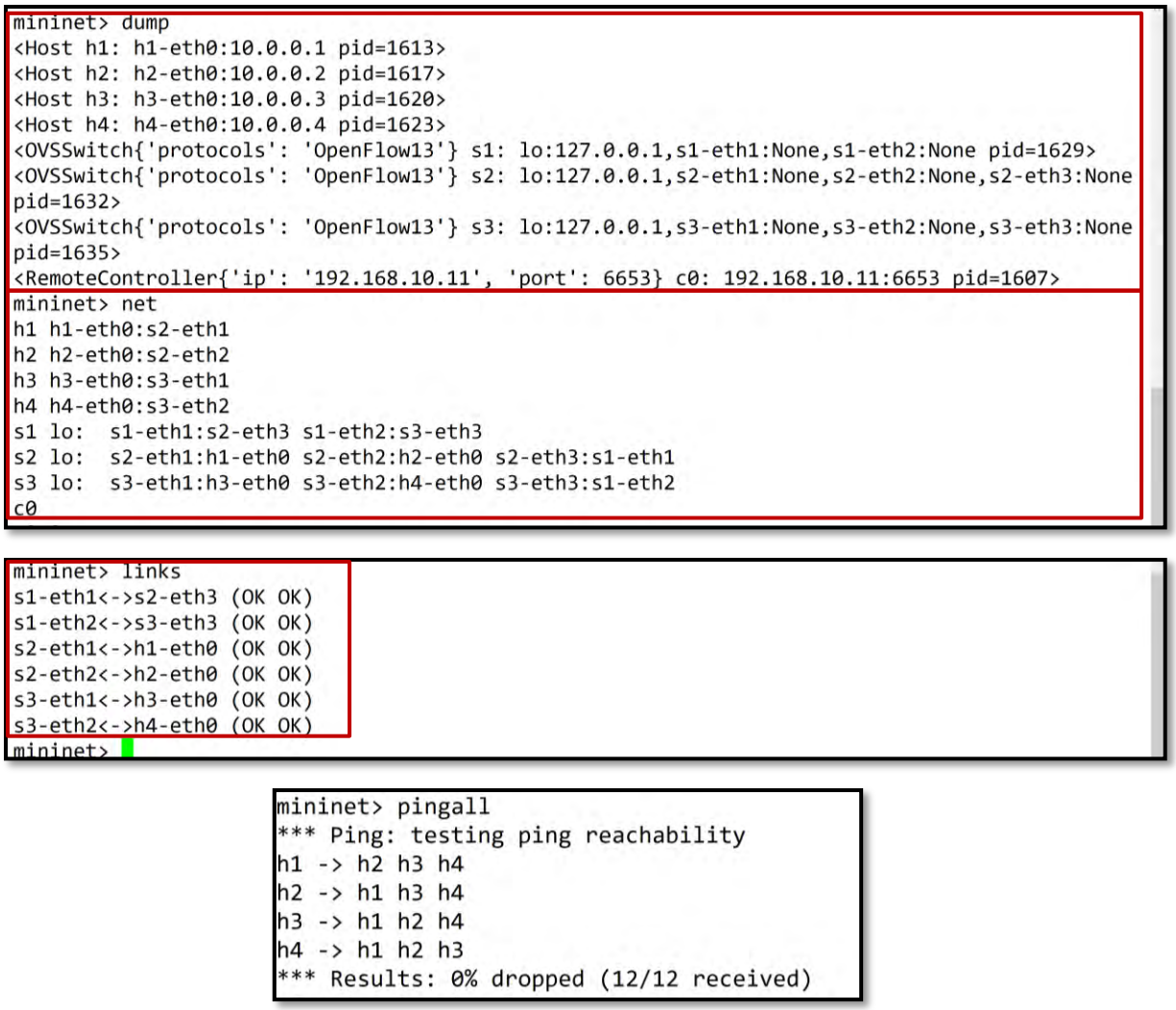


Figura 4-78 Ejecución exitosa de Mininet en PoC ONOS
Fuente: *Autor*

Al crear la topología en Mininet, es posible visualizar los logs en CLI de ONOS (*Fig. 4-79*):

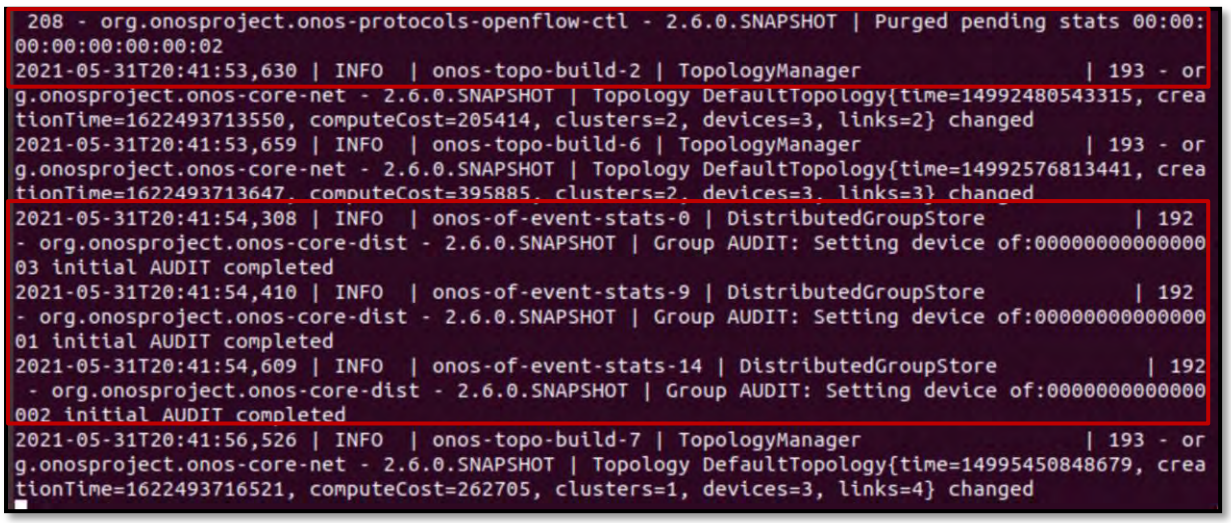


Figura 4-79 Logs en CLI de controlador en PoC ONOS
Fuente: *Autor*

En cuanto a la visualización gráfica de la topología, es posible apreciarla en las *Fig 4-80* a *4-83*:

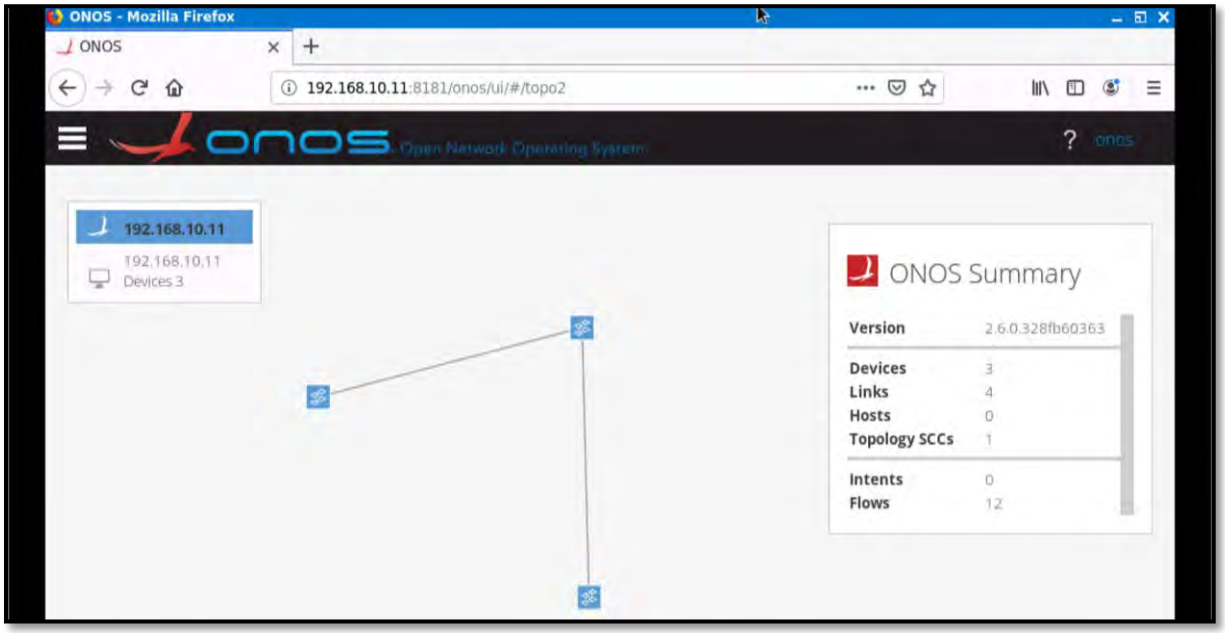
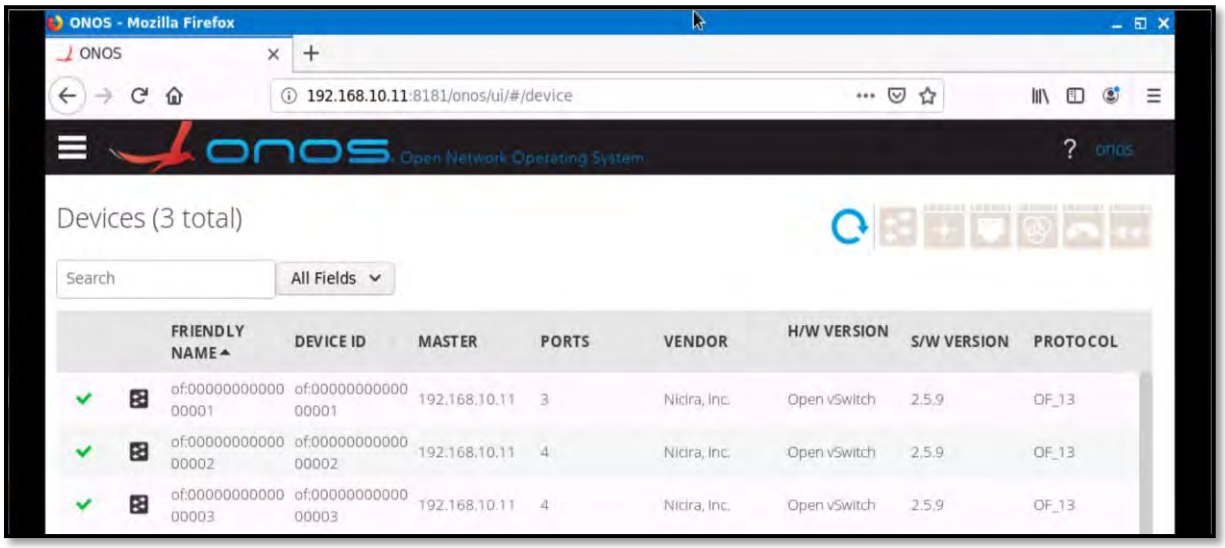
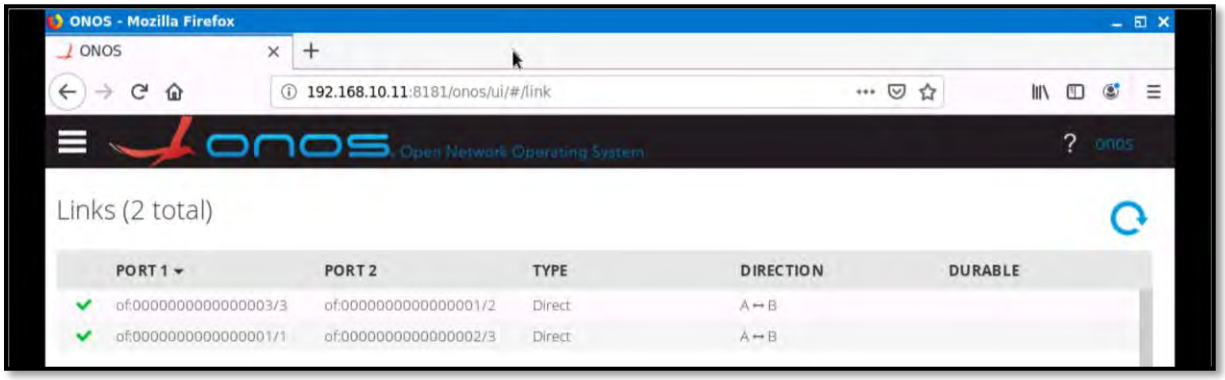


Figura 4-80 Topología descubierta por Controlador en PoC ONOS
Fuente: *Autor*





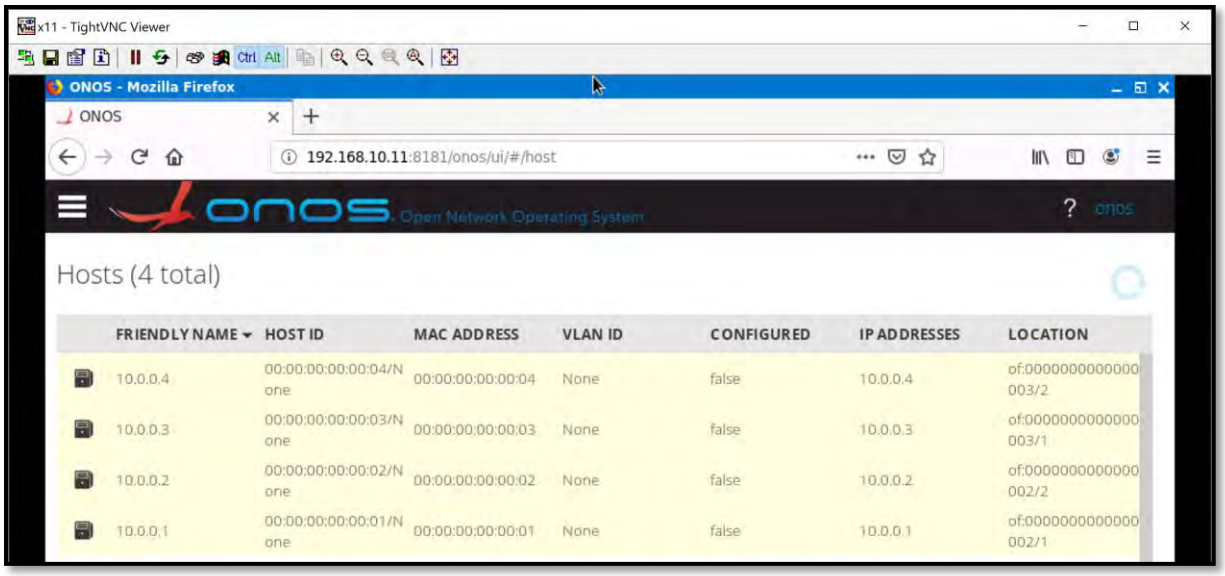


Figura 4-81 Dispositivos, Enlaces y Usuarios descubiertos en PoC ONOS
Fuente: *Autor*

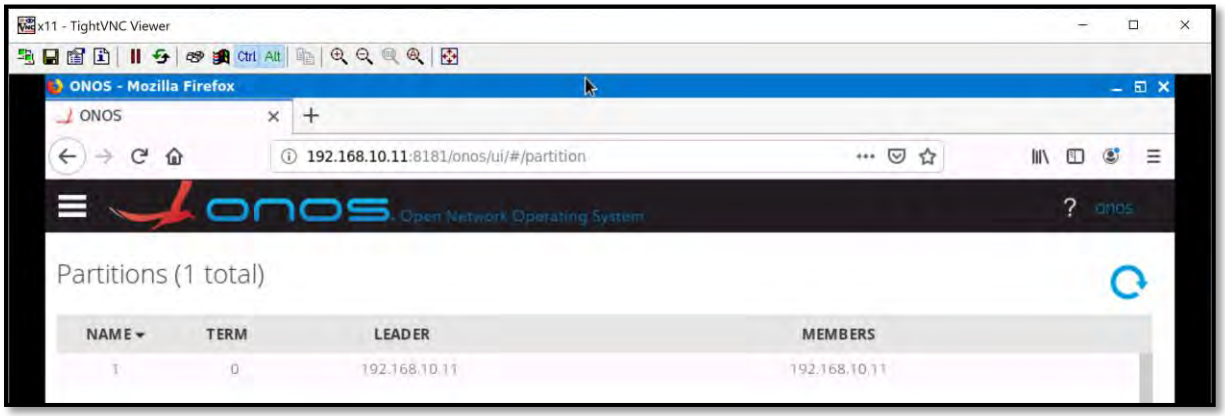


Figura 4-82 Información de Controlador en PoC ONOS
Fuente: *Autor*

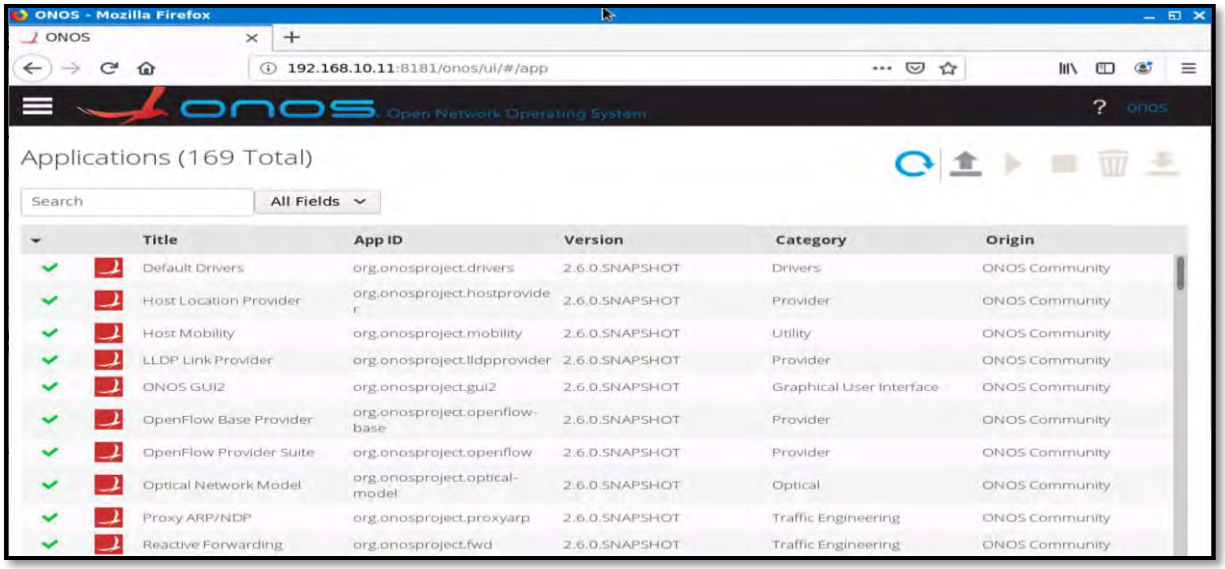


Figura 4-83 Aplicaciones disponibles en PoC ONOS
Fuente: *Autor*

Capturas de Tráfico en Tránsito (*Wireshark*)

La *Fig. 4-84* muestra la captura de tráfico entre Mininet y los Controladores ODL/ONOS, visualizando el intercambio de mensajes *OpenFlow 1.3.* Además, se verifica la encapsulación de ICMP en el protocolo *OpenFlow*, lo que muestra que la comunicación pasa por el controlador.

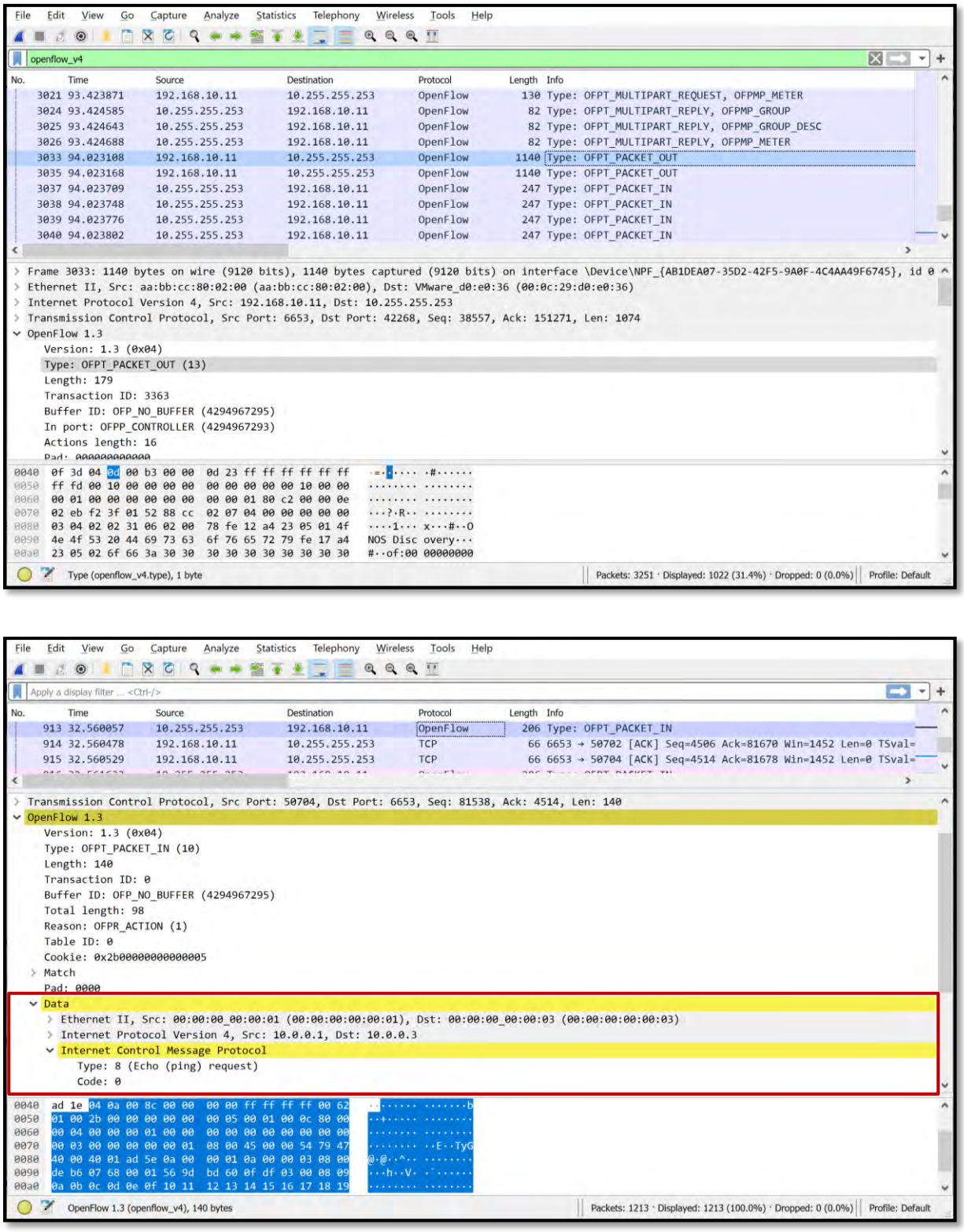


Figura 4-84 Encapsulación de ICMP dentro de OpenFlow en PoC ODL/ONOS
Fuente: *Autor*

Resultados experimentales (*Stress-Approach*)

Con el fin de comparar a ODL y ONOS, se realizaron pruebas de estrés obteniendo los siguientes resultados:

Tabla 4-1 Resultados Prueba de Conectividad (Ping) ODL vs ONOS

Controlador	Mínimo RTT [ms]	Máximo RTT [ms]	Transmisión Máxima [paquetes/seg]	Transmisión Promedio [paquetes/seg]
ODL	0.335	0.637	115	67.382
ONOS	0.098	1.986	101	57.211

Fuente: Autor

Tabla 4-2 Resultados de Tasa de Ráfaga (*Burst* Rate) ODL vs ONOS

ODL

Longitud Incremental del Paquete [Bytes]	Conteo de Paquetes (antes de falla)	Tasa de Ráfaga	Tasa de Envío [ms]	Inicio de Ráfaga
0-24	0	0.438	0.059	40.678
25-49	0	0.087	0.0082	40.789
50-99	0	0.091	0.0678	41.456
100-199	0	0.187	0.099	33.236
200-399	0	0.128	0.003	31.2578
400-799	0	0.057	0.0084	41.398
800-1599	0	0.082	0.0014	38.789
1600-3199	0	0.157	0.0345	31.078
3200-6399	430	0.141	0.0025	37.903
mayores a 6400	120	0.078	0.001	35.347

ONOS

Longitud Incremental del Paquete [Bytes]	Conteo de Paquetes (antes de falla)	Tasa de Ráfaga	Tasa de Envío [ms]	Inicio de Ráfaga
0-24	0	0.277	0.129	48.55
25-49	0	0.59	0.033	48.498
50-99	0	0.321	0.124	47.378
100-199	0	0.199	0.109	43.178
200-399	0	0.223	0.022	41.599
400-799	0	0.375	0.097	41.77
800-1599	4267	0.192	0.0014	39.121
1600-3199	160	0.256	0.0345	33.845
3200-6399	225	0.341	0.0025	41.111
mayores a 6400	404	0.148	0.001	39.587

Fuente: Autor

Los resultados obtenidos en el PoC son muy similares al análisis realizado en (Badotra & Narayan Panda, 2019).

En (Badotra & Narayan Panda, 2019), se grafican tendencias de comportamiento para tráfico TCP y UDP, así como una comparativa de *Jitter* y Tasa de Ráfaga muy interesante que vale la pena observar en las *Fig. 4-85* y *Fig. 4-86.*

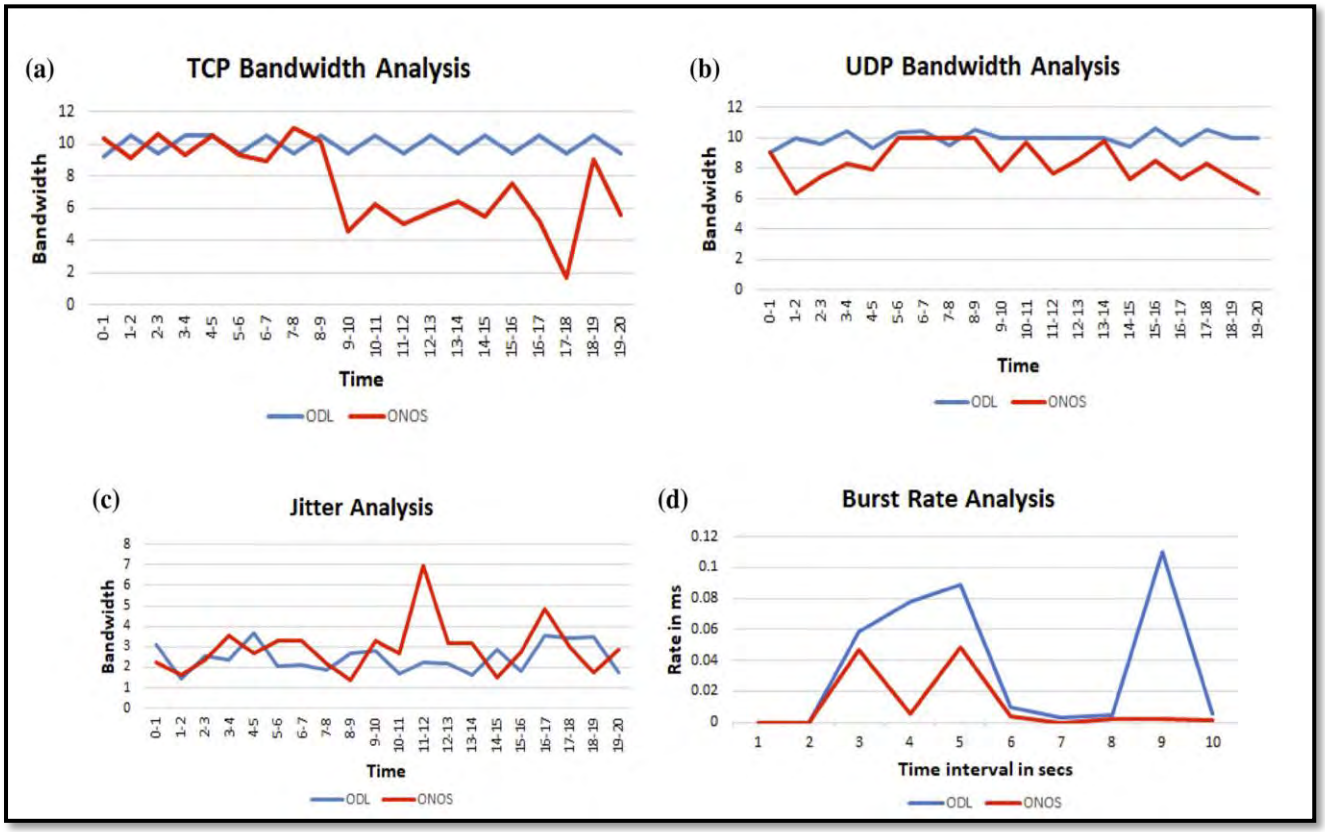


Figura 4-85 Gráficos comparativos entre ODL y ONOS
Recuperado de (Badotra & Narayan Panda, 2019)

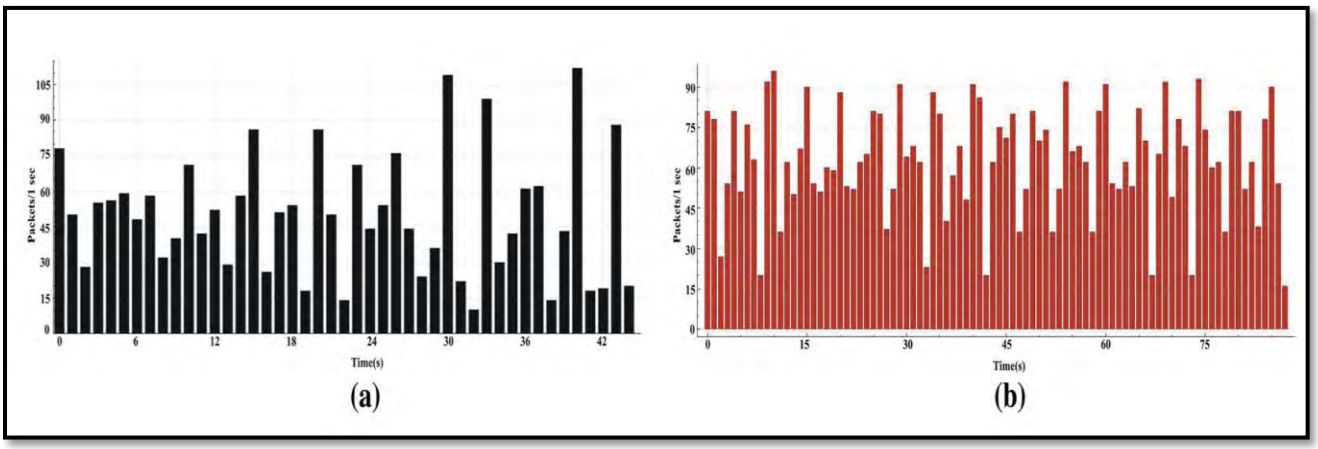


Figura 4-86 Barras sobre la Transmisión de paquetes OpenFlow: (a) ONOS y (b) ODL
Recuperado de (Badotra & Narayan Panda, 2019)

4.4 Prueba de Concepto de NG-SDN

En ***2.8.1 Fundamentos de NG-SDN: SDN de próxima Generación*** y en ***2.8.2 Programación en NG-SDN: P4 y Stratum*** se analizaron los conceptos básicos de *Next-Generation SDN*, proyecto soportado por la ONF; ahora, se pondrá a prueba una emulación de este entorno y verificar su factibilidad de uso.

Usando la VM preparada por la ONF para PoCs (la cual puede ser descargada en ***http://bit.ly/ngsdn-tutorial-ova***) se desarrollará un conjunto de pruebas para el manejo de NG-SDN.

4.4.1 Programación en P4

Antes de empezar, es necesario recordar que NG-SDN emplea P4 como lenguaje para construir y programar el plano de datos, dando una capacidad de personalización extrema a la infraestructura subyacente.

Durante todo el desarrollo de la presente tesis se estableció como válido el concepto dado por la ONF para SDN: *Separación del plano de Datos y Plano de Control con el fin de administrar, monitorear, configurar de forma centralizada la infraestructura, dotando de mayor velocidad tanto de reacción como de implementación, así como capacidad de programación en la red.* Sin embargo, los desarrollos tradicionales de SDN se basaron en un plano de datos vinculado a través de protocolos *Southbound* como *OpenFlow* y la gestión y configuración mediante modelos tipo YANG y API-RESTs como NETCONF, lamentablemente, la falta de independencia total en el plano de datos ha impedido la innovación tan deseada en este campo. En este punto, *OpenFlow* carece de especificaciones para la programabilidad del Plano de Datos, así cada fabricante tiene su propia visión y modelos propios de implementación, lo que conlleva a modificaciones y adecuaciones en el Plano de Control: La realidad es que los Plano de Control en ambientes SDN están desarrollados para *Hardware* específico y muchas veces propietario según (Cascone, 2019).

El *OpenNetworking* y NFV dieron pasos muy adecuados al incluir abstracciones entre el HW y SW como HALs (*Hardware Abstraction Layers*), pero aún no es suficiente.

NG-SDN trajo consigo conceptos de programabilidad total (incluyendo al plano de datos), independencia total de *hardware* e implementaciones *zero-touch* que mediante *Stratum* y *ONOS*, permitieron implementar modelos NG-SDN, siendo P4 el fundamento de esta capacidad de programabilidad extrema.

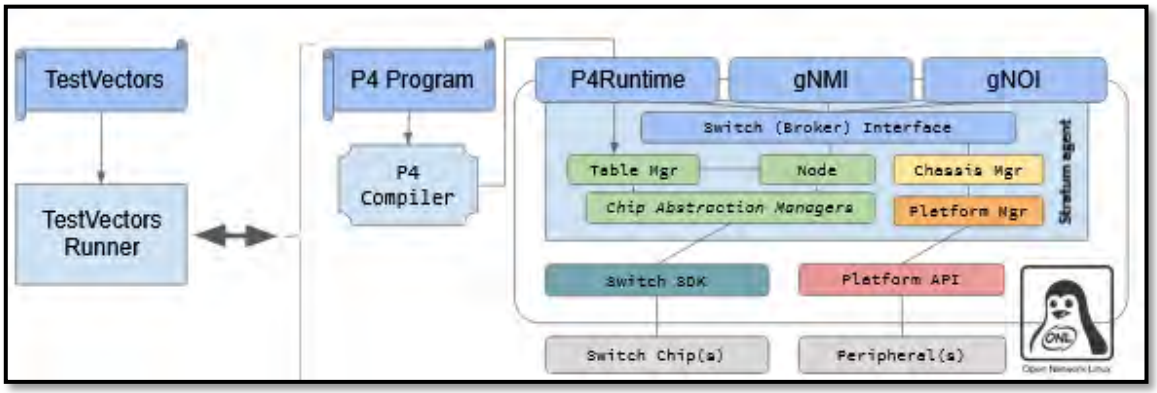


Figura 4-87 Esquematización de NG-SDN: Stratum NOS-Switch Recuperado de (Cascone, 2019)

Para comprender cómo es posible construir un equipo intermediario (un *router* o *switch*) en P4, observe rápidamente el siguiente código base para el presente PoC:

https://github.com/opennetworkinglab/ngsdn-tutorial/blob/master/p4src/main.p4

En dicho código, se establece la construcción de un *router* para transmisión de datos en IPv6. A continuación se hará un análisis de ese *script.*

Al inicio de un código en P4, se establecerá las librerías principales a usar. Para el código ***main.p4*** se empleará la librería *core.p4* y el denominado *v1model.p4*

```
// Any P4 program usually starts by including the P4 core library and the
// architecture definition, v1model in this case.
// https://github.com/p4lang/p4c/blob/master/p4include/core.p4
// https://github.com/p4lang/p4c/blob/master/p4include/v1model.p4
#include <core.p4>
#include <v1model.p4>
```

Script 32 Librerías base de P4 para main.p4
Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

La construcción de un equipo de red se da gracias al mencionado *V1Model,* estableciendo así el *pipeline* o esquematización de estos equipos:

```
// V1Model is a P4_16 architecture that defines 7 processing blocks.
//
//   +------+  +------+  +-------+  +-------+  +------+  +------+  +--------+
// ->|PARSER|->|VERIFY|->|INGRESS|->|TRAFFIC|->|EGRESS|->|UPDATE|->+DEPARSER|->
//   |      |  |CKSUM |  |PIPE   |  |MANAGER|  |PIPE  |  |CKSUM |  |        |
//   +------+  +------+  +-------+  +--------  +------+  +------+  +--------+
```

Figura 4-88 Esquematización de un equipo de red en P4 – V1Model
Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

Dentro de las librerías base también constan **CPU_PORT** y **CPU_CLONE_SESSION ID**, los cuales permiten asociar los números de puerto en el ingreso (*packet-in*) y en la salida (*packet-out*) de un paquete al ser procesado por el *P4Runtime Controller*.

De igual manera, se establecerán las variables y constantes con su tipo para construir el programa. Estas variables definen las cabeceras con información como dirección MAC, dirección IPv4/IPv6, número de puerto en capa de transporte, etc.

```
// TYPEDEF DECLARATIONS
// To favor readability.
//------------------------------------------------------------------------------
typedef bit<9>   port_num_t;
typedef bit<48>  mac_addr_t;
typedef bit<16>  mcast_group_id_t;
typedef bit<32>  ipv4_addr_t;
typedef bit<128> ipv6_addr_t;
typedef bit<16>  l4_port_t;
```

```
//------------------------------------------------------------------------------
// CONSTANT VALUES
//------------------------------------------------------------------------------
const bit<16> ETHERTYPE_IPV6 = 0x86dd;

const bit<8> IP_PROTO_TCP = 6;
const bit<8> IP_PROTO_UDP = 17;
const bit<8> IP_PROTO_ICMPV6 = 58;

const mac_addr_t IPV6_MCAST_01 = 0x33_33_00_00_00_01;

const bit<8> ICMP6_TYPE_NS = 135;
const bit<8> ICMP6_TYPE_NA = 136;
const bit<8> NDP_OPT_TARGET_LL_ADDR = 2;
const bit<32> NDP_FLAG_ROUTER = 0x80000000;
const bit<32> NDP_FLAG_SOLICITED = 0x40000000;
const bit<32> NDP_FLAG_OVERRIDE = 0x20000000;
```

Script 33 Variables y Constantes en P4 para la construcción del Router IPv6 Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

Usando esas variables y constantes, se planteará la definición de las cabeceras, por ejemplo, la de *Ethernet* y la de la capa de red (IPv6). Las de Capa de transporte (UDP/TCP) y otras son necesarias también.

```
//------------------------------------------------------------------------------
// HEADER DEFINITIONS
//------------------------------------------------------------------------------
header ethernet_t {
    mac_addr_t  dst_addr;
    mac_addr_t  src_addr;
    bit<16>     ether_type;
}

header ipv6_t {
    bit<4>   version;
    bit<8>   traffic_class;
    bit<20>  flow_label;
    bit<16>  payload_len;
    bit<8>   next_hdr;
    bit<8>   hop_limit;
    bit<128> src_addr;
    bit<128> dst_addr;
}
```

Script 34 Establecimiento de Cabeceras en P4 para la construcción del Router IPv6 Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

P4Runtime Controller es el encargado de procesar los paquetes de entrada y de salida con el fin de inicializar el *pipeline* en el arribo de un flujo de tráfico:

```
// Packet-in header. Prepended to packets sent to the CPU_PORT and used by the
// P4Runtime server (Stratum) to populate the PacketIn message metadata fields.
// Here we use it to carry the original ingress port where the packet was
// received.
@controller_header("packet_in")
header packet_in_t {
    port_num_t  ingress_port;
    bit<7>      _pad;
}
```

```
// Packet-out header. Prepended to packets received from the CPU_PORT. Fields of
// this header are populated by the P4Runtime server based on the P4Runtime
// PacketOut metadata fields. Here we use it to inform the P4 pipeline on which
// port this packet-out should be transmitted.
@controller_header("packet_out")
header packet_out_t {
    port_num_t  egress_port;
    bit<7>      _pad;
}

// We collect all headers under the same data structure, associated with each
// packet. The goal of the parser is to populate the fields of this struct.
struct parsed_headers_t {
    packet_out_t  packet_out;
    packet_in_t   packet_in;
    ethernet_t    ethernet;
    ipv6_t        ipv6;
    tcp_t         tcp;
    udp_t         udp;
    icmpv6_t      icmpv6;
    ndp_t         ndp;
}
```

Script 35 Inicialización del Pipeline al procesar un paquete de entrada/salida en P4
Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

El proceso continúa con la generación de metadata importante para el envío de datos, entre ellos, los puertos de servicio que están circulando por la red (tcp/udp o cualquier otro servicio de red).

```
//------------------------------------------------------------------------------
// USER-DEFINED METADATA
// User-defined data structures associated with each packet.
//------------------------------------------------------------------------------
struct local_metadata_t {
    l4_port_t  l4_src_port;
    l4_port_t  l4_dst_port;
    bool       is_multicast;
}
```

Script 36 Metadata al procesar un paquete de entrada/salida en P4
Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

Al llegar el paquete, éste debe ser aceptado o no debido a ciertas características. La función ***ParserImp*** se encarga de esa lógica, tal como se aprecia en el *Script 37*.

```
parser ParserImpl (packet_in packet,
                   out parsed_headers_t hdr,
                   inout local_metadata_t local_metadata,
                   inout standard_metadata_t standard_metadata)
{
    // We assume the first header will always be the Ethernet one, unless the
    // the packet is a packet-out coming from the CPU_PORT.
    state start {
        transition select(standard_metadata.ingress_port) {
            CPU_PORT: parse_packet_out;
            default: parse_ethernet;
        }
    }
```

```
    state parse_packet_out {
        packet.extract(hdr.packet_out);
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.ether_type){
            ETHERTYPE_IPV6: parse_ipv6;
            default: accept;
        }
    }
```

Script 37 Configuración parcial de función ParserImp para empezar el procesamiento de paquetes Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

Se recomienda verificar la presencia o no errores (*Checksum*) en el flujo de tráfico, caso contrario, el usuario final se encargará de verificarlo.

Posteriormente, se procesa el flujo de datos, comenzando por L2 y luego L3.

Para el caso de L2, se deberá actualizar la tabla MAC, así como realizar la conmutación respectiva dependiendo la dirección de destino (*unicast*, *broadcast, multicast*).

```
// --- l2_exact_table (for unicast entries) -------------------------------

action set_egress_port(port_num_t port_num) {
    standard_metadata.egress_spec = port_num;
}

table l2_exact_table {
    key = {
        hdr.ethernet.dst_addr: exact;
    }
    actions = {
        set_egress_port;
        @defaultonly drop;
    }
    const default_action = drop;
    // The @name annotation is used here to provide a name to this table
    // counter, as it will be needed by the compiler to generate the
    // corresponding P4Info entity.
    @name("l2_exact_table_counter")
    counters = direct_counter(CounterType.packets_and_bytes);
}
```

```
// --- l2_ternary_table (for broadcast/multicast entries) -----------------

action set_multicast_group(mcast_group_id_t gid) {
    // gid will be used by the Packet Replication Engine (PRE) in the
    // Traffic Manager--located right after the ingress pipeline, to
    // replicate a packet to multiple egress ports, specified by the control
    // plane by means of P4Runtime MulticastGroupEntry messages.
    standard_metadata.mcast_grp = gid;
    local_metadata.is_multicast = true;
}
```

```
table l2_ternary_table {
    key = {
        hdr.ethernet.dst_addr: ternary;
    }
    actions = {
        set_multicast_group;
        @defaultonly drop;
    }
    const default_action = drop;
    @name("l2_ternary_table_counter")
    counters = direct_counter(CounterType.packets_and_bytes);
}
```

Script 38 Configuración parcial de L2-Conmutación en P4
Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

El siguiente paso es sobre el enrutamiento (L3). El enrutamiento se da si L2 es el adecuado (dir. MAC dirigido al *router*). Para este proceso, los equipos deberán construir la tabla de enrutamiento y basarse en ella para realizar la elección del mejor camino. También, los *routers* modernos tienen la capacidad de hacer ECMP (*Equal Cost MultiPath*) para balanceo de carga.

```
// --- my_station_table -------------------------------------------------

// Matches on all possible my_station MAC addresses associated with this
// switch. This table defines only one action that does nothing to the
// packet. Later in the apply block, we define logic such that packets are
// routed if and only if this table is "hit", i.e. a matching entry is found
// for the given packet.

table my_station_table {
    key = {
        hdr.ethernet.dst_addr: exact;
    }
    actions = { NoAction; }
    @name("my_station_table_counter")
    counters = direct_counter(CounterType.packets_and_bytes);
}
```

Script 39 Configuración de MyStationTable para verificar si se debe enrutar o no un paquete en P4
Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

La tabla de enrutamiento IPv6 se construirá mediante *Action selectors*, los cuales son construcciones de *V1Model,* mejorando así la rapidez de procesamiento y toma de decisiones del equipo. Entre los *selectors* está el de ECMP y definir el próximo salto.

```
// --- routing_v6_table -------------------------------------------------

// To implement ECMP, we use Action Selectors, a v1model-specific construct.
// A P4Runtime controller, can use action selectors to associate a group of
// actions to one table entry. The speficic action in the group will be
// selected by perfoming a hash function over a pre-determined set of header
// fields. Here we instantiate an action selector named "ecmp_selector" that
// uses crc16 as the hash function, can hold up to 1024 entries (distinct
// action specifications), and produces a selector key of size 16 bits.

action_selector(HashAlgorithm.crc16, 32w1024, 32w16) ecmp_selector;
```

```
action set_next_hop(mac_addr_t dmac) {
    hdr.ethernet.src_addr = hdr.ethernet.dst_addr;
    hdr.ethernet.dst_addr = dmac;
    // Decrement TTL
    hdr.ipv6.hop_limit = hdr.ipv6.hop_limit - 1;
}

// Look for the "implementation" property in the table definition.
table routing_v6_table {
  key = {
      hdr.ipv6.dst_addr:          lpm;
      // The following fields are not used for matching, but as input to the
      // ecmp_selector hash function.
      hdr.ipv6.dst_addr:          selector;
      hdr.ipv6.src_addr:          selector;
      hdr.ipv6.flow_label:        selector;
      hdr.ipv6.next_hdr:          selector;
      local_metadata.l4_src_port: selector;
      local_metadata.l4_dst_port: selector;
  }
  actions = {
      set_next_hop;
  }
  implementation = ecmp_selector;
  @name("routing_v6_table_counter")
  counters = direct_counter(CounterType.packets_and_bytes);
}
```

Script 40 Procesos de Enrutamiento IPv6 en P4
Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

Otras funciones adicionales pueden configurarse en este equipo, como ACLs para filtrar tráfico, técnicas de PBR y NDP (*Neighbor Discovery Protocol*) para IPv6.

Finalmente, una vez elegido el próximo salto, el paquete debe prepararse para salir del equipo. La función ***EgressPipeImp*** logra ese cometido. El *Script 41* plantea ello.

Luego, se debe generar un nuevo *Checksum* para evitar el envío de paquete con errores, tal como se aprecia en el *Script 42.*

El último bloque del *V1Model* se conoce como ***Deparser***, bloque no menos importante pues permite el proceso de serialización del paquete al medio físico. Esta configuración se aprecia en el *Script 43.*

```
// Similarly to the ingress pipeline, the egress one operates on the parsed
// headers (hdr), the user-defined metadata (local_metadata), and the
// architecture-specific instrinsic one (standard_metadata) which now
// defines a read-only "egress_port" field.
//-----------------------------------------------------------------------------
control EgressPipeImpl (inout parsed_headers_t hdr,
                        inout local_metadata_t local_metadata,
                        inout standard_metadata_t standard_metadata) {
```

```
    apply {
        // If this is a packet-in to the controller, e.g., if in ingress we
        // matched on the ACL table with action send/clone_to_cpu...
        if (standard_metadata.egress_port == CPU_PORT) {
            // Add packet_in header and set relevant fields, such as the
            // switch ingress port where the packet was received.
            hdr.packet_in.setValid();
            hdr.packet_in.ingress_port = standard_metadata.ingress_port;
            // Exit the pipeline here.
            exit;
        }

        // If this is a multicast packet (flag set by l2_ternary_table), make
        // sure we are not replicating the packet on the same port where it was
        // received. This is useful to avoid broadcasting NDP requests on the
        // ingress port.
        if (local_metadata.is_multicast == true &&
              standard_metadata.ingress_port == standard_metadata.egress_port) {
            mark_to_drop(standard_metadata);
        }
    }
}
```

Script 41 Preparación del paquete para salir del Equipo en P4
Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

```
// 5. CHECKSUM UPDATE
//
// Provide logic to update the checksum of outgoing packets.
//-----------------------------------------------------------------------------
control ComputeChecksumImpl(inout parsed_headers_t hdr,
                            inout local_metadata_t local_metadata)
{
    apply {
        // The following function is used to update the ICMPv6 checksum of NDP
        // NA packets generated by the ndp_reply_table in the ingress pipeline.
        // This function is executed only if the NDP header is present.
        update_checksum(hdr.ndp.isValid(),
            {
                hdr.ipv6.src_addr,
                hdr.ipv6.dst_addr,
                hdr.ipv6.payload_len,
                8w0,
                hdr.ipv6.next_hdr,
                hdr.icmpv6.type,
                hdr.icmpv6.code,
                hdr.ndp.flags,
                hdr.ndp.target_ipv6_addr,
                hdr.ndp.type,
                hdr.ndp.length,
                hdr.ndp.target_mac_addr
            },
            hdr.icmpv6.checksum,
            HashAlgorithm.csum16
        );
    }
}
```

Script 42 Actualización de Checksum antes de salir el paquete a su destino en P4
Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

```
// 6. DEPARSER
//
// This is the last block of the V1Model architecture. The deparser specifies in
// which order headers should be serialized on the wire. When calling the emit
// primitive, only headers that are marked as "valid" are serialized, otherwise,
// they are ignored.
//------------------------------------------------------------------------------
control DeparserImpl(packet_out packet, in parsed_headers_t hdr) {
    apply {
        packet.emit(hdr.packet_in);
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv6);
        packet.emit(hdr.tcp);
        packet.emit(hdr.udp);
        packet.emit(hdr.icmpv6);
        packet.emit(hdr.ndp);
    }
}
```

Script 43 Función Deparser – Serialización del Paquete en P4
Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

Con el fin de que el programa pueda correr, se instanciará la arquitectura *V1Model* con todos los bloques de control antes analizados.

```
V1Switch(
    ParserImpl(),
    VerifyChecksumImpl(),
    IngressPipeImpl(),
    EgressPipeImpl(),
    ComputeChecksumImpl(),
    DeparserImpl()
) main;
```

Script 44 Instanciamiento de Arquitectura V1Model en P4
Recuperado de (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

Al ejecutar este programa, se tendrá un *router IPv6* totalmente funcional en P4.

Entendiendo de forma global esta configuración, se podrá ejecutar con mayor facilidad el PoC de NG-SDN.

4.4.2 Topología del PoC de NG-SDN: Programación en P4, YANG, OpenConfig-gNMI (Telemetría) y ONOS

La topología usada para el PoC de NG-SDN se indica en la *Fig. 4-89*, la cual fue diseñada con los siguientes dispositivos en un entorno *Spine-Leaf* mediante Mininet y ONOS como Controlador SDN:

- Dos (2) *Switches Spines - Stratum*[61]
- Dos (2) *Switches Leaves - Stratum*
- Seis (6) *Hosts*

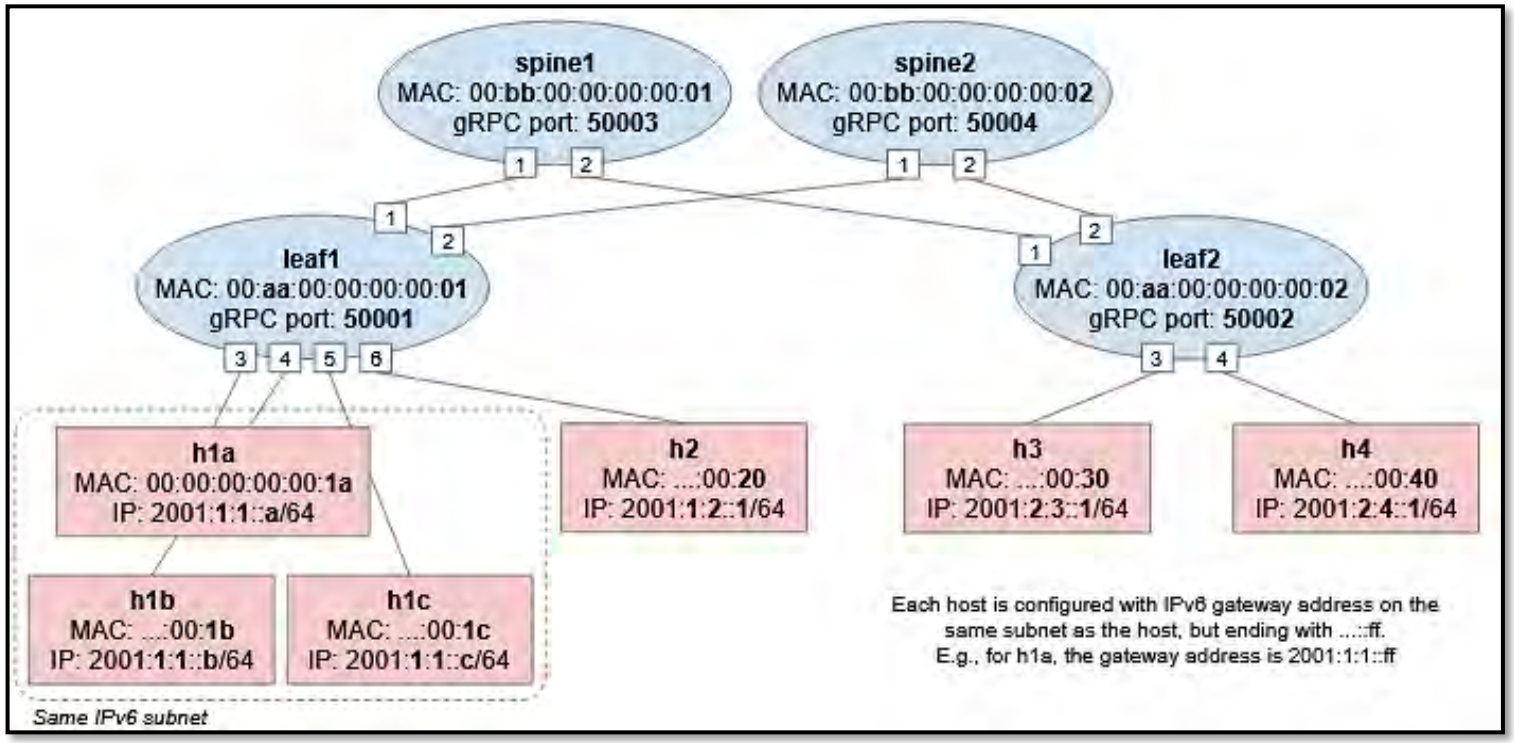


Figura 4-89 Topología PoC NG-SDN
Basado en (O'Connor & Cascone, Tutorial Next-Gen SDN - main.p4, 2019)

[61] Stratum Switches: https://opennetworking.org/stratum/

Para la ejecución del PoC de NG-SDN, se importará la VM de *ONF-Connect Tutorial*[62] en *VirtualBox*. Se sugiere al menos 4GB de RAM y 4 núcleos de CPU asignados a esta VM.

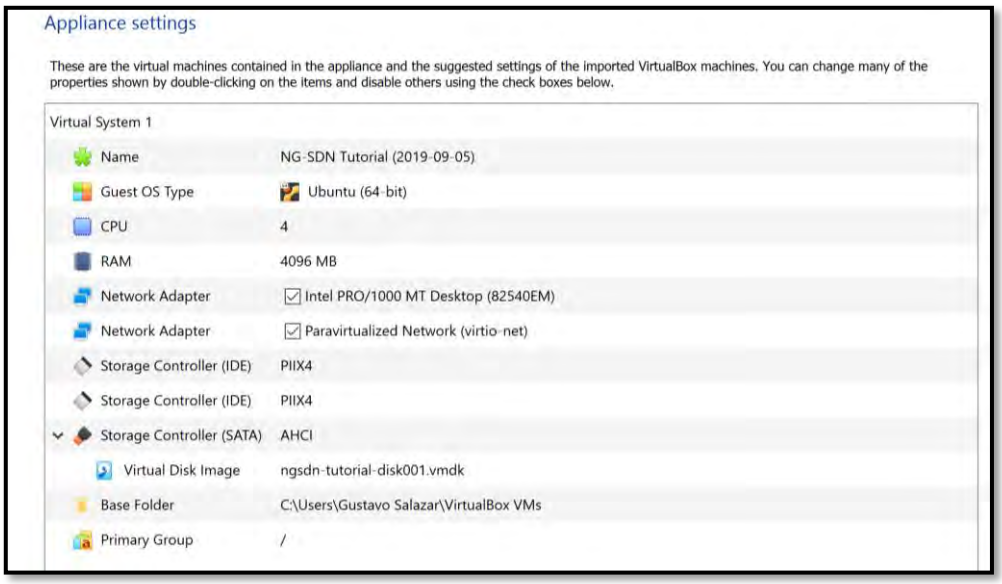


Figura 4-90 Características de VM – NG-SDN
Fuente: *Autor*

El ingreso a la VM basada en Ubuntu 16.04 es con el usuario **sdn** y la contraseña **rocks**.

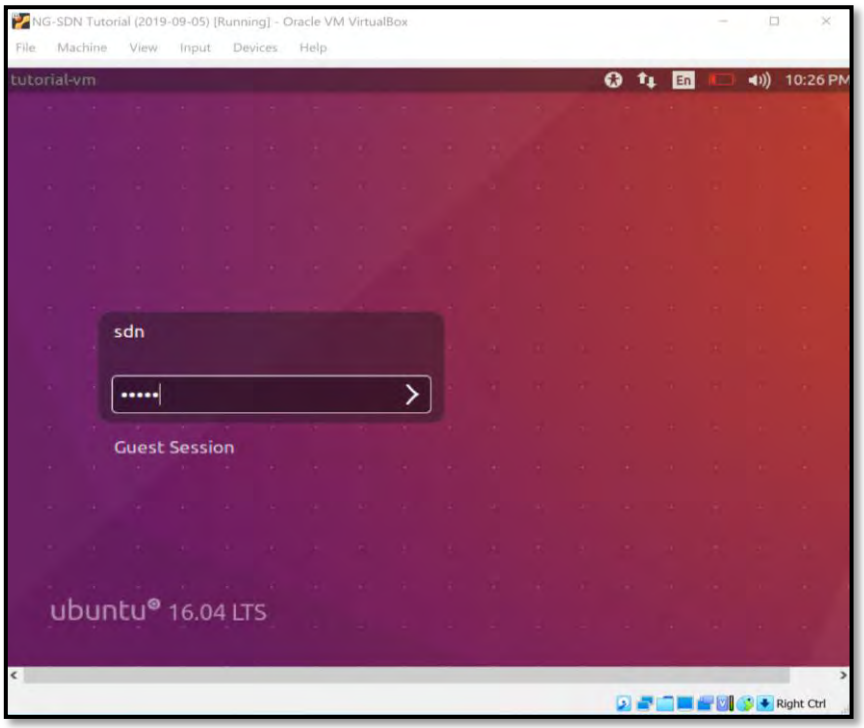


Figura 4-91 Arranque de VM – NG-SDN
Fuente: *Autor*

[62] Descargar de VM ONF Tutorial – NG-SDN: http://bit.ly/ngsdn-tutorial-ova

La VM cuenta con todo lo necesario para el PoC, no obstante, es necesario una actualización:

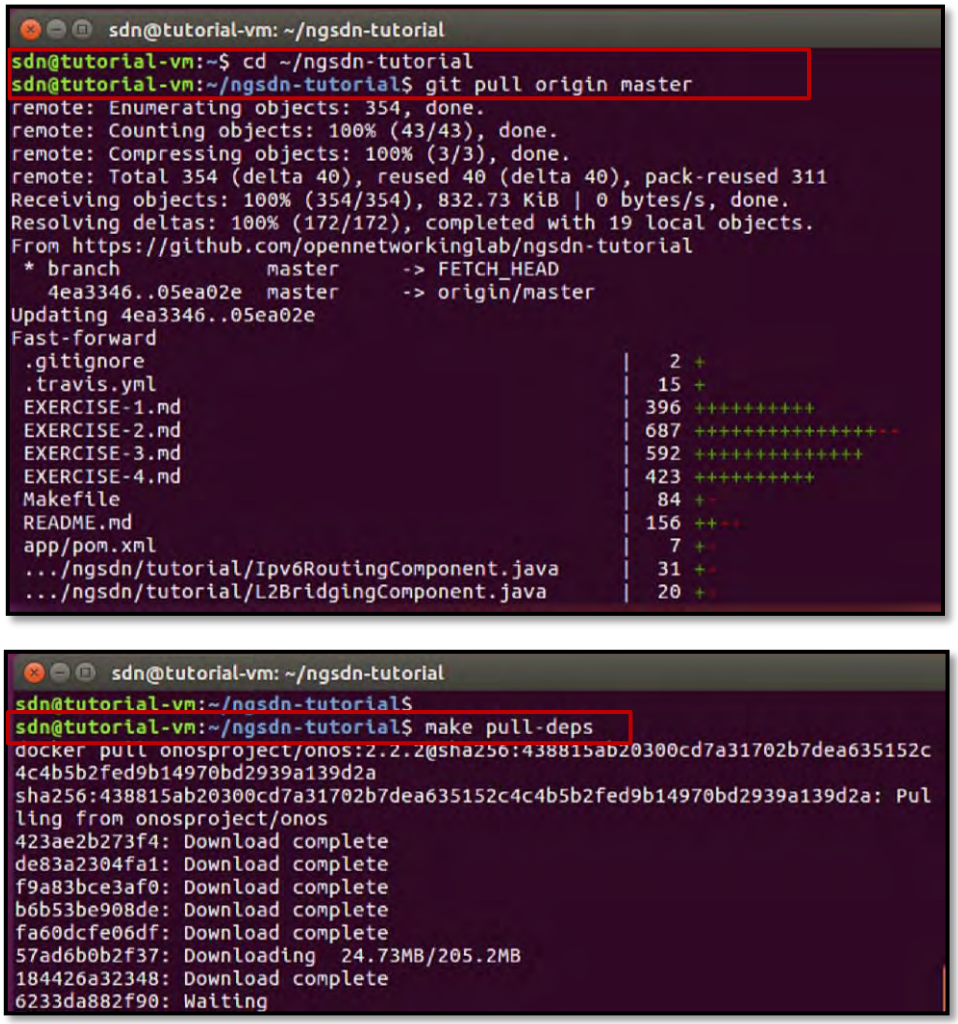


Figura 4-92 Actualización e instalación de dependencias de VM – NG-SDN
Fuente: *Autor*

Los equipos de la infraestructura mostrada en la *Fig. 4-89* están diseñados mediante P4 y se empleará el compilador denominado *p4c*[63].

El primer paso es compilar el programa de nombre *simple_switch (p4c-bm2-ss),* basado en *main.p4* (analizado anteriormente) para entender su funcionamiento, para ello se usará el siguiente comando:

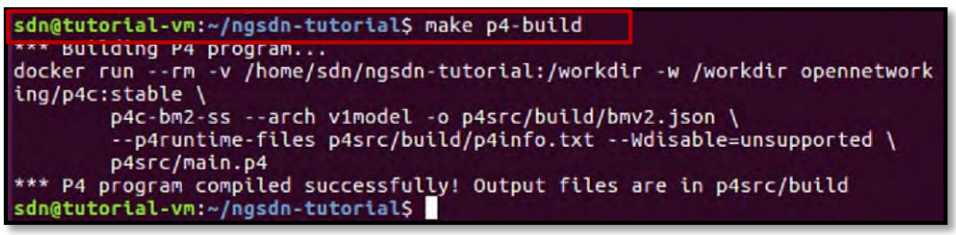


Figura 4-93 Compilación en p4c – PoC NG-SDN
Fuente: *Autor*

[63] p4c – P4 Compiler: https://github.com/p4lang/p4c

El archivo ***bmv2.json*** define la configuración del *switch* con el fin de indicar el cómo se procesará el flujo de datos cuando el equipo lo reciba según el programa P4.

La VM de este PoC cuenta con comandos ***make*** que facilitan el desarrollo de la prueba de concepto. Como una referencia rápida de estos comandos puede ir a ***Anexo F: Scripts para la Implementación de PoC NG-SDN***

Es momento de iniciar Mininet para correr la arquitectura emulada. El comando ***make start*** inicializará dos *dockers*, uno para Mininet y otro para ONOS como controlador SDN.

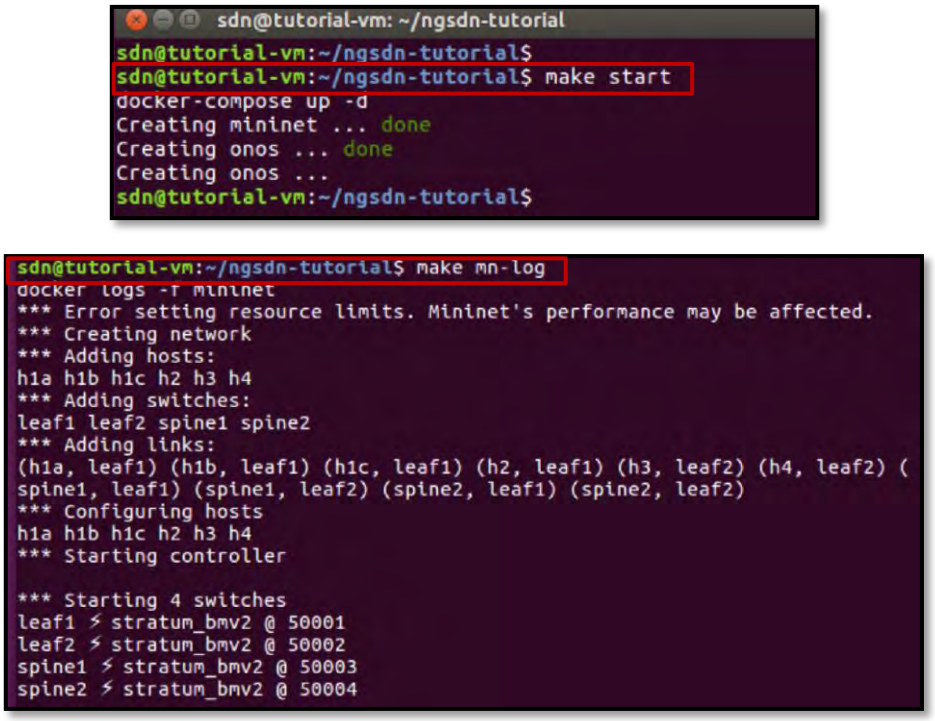


Figura 4-94 Inicialización de Mininet y ONOS – PoC NG-SDN
Fuente: *Autor*

Los parámetros establecidos para Mininet se definen en el archivo ***docker-compose.yml*** y la topología está definida en el archivo ***mininet/topo.py*** (ver ***Anexo F: Scripts para la Implementación de PoC NG-SDN***).

Al iniciar Mininet, un conjunto de archivos relacionados con la ejecución de cada dispositivo se genera en el *path* ***ngsdn-tutorial/tmp***, entre ellos:

- Logs de funcionamiento de los *switches*: *tmp/leaf1/stratum_bmv2.log*
- Configuración inicial de los puertos del *switch Stratum* (*chassis config*): *tmp/leaf1/chassis-config.txt*
- Logs de *P4Runtime*: *tmp/leaf1/write-reqs.txt*

Una de las ventajas de usar P4 en una infraestructura de red es la posibilidad de personalización y de interacción. Para probar esta característica se usará al *P4Runtime Shell*, el cual permitirá la interconectividad entre un *P4Runtime Server* y los equipos y así poder invocar comandos para obtener información.

Para conectar el *P4Runtime Shell* al dispositivo *leaf1* se usa el siguiente comando:

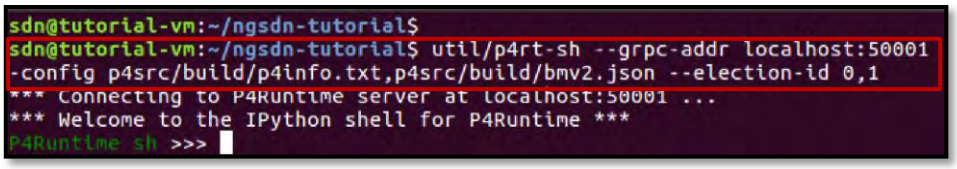


Figura 4-95 Conexión P4Runtime Shell y Leaf1 – PoC NG-SDN
Fuente: *Autor*

Conectividad en el PoC de NG-SDN

El primer paso es inicializar Mininet y a partir de su CLI, incluir las entradas *P4Runtime* requeridas.

Para que dos *hosts* dentro de la misma red puedan conectarse entre sí en una red IPv6, el protocolo **NDP (*Neighbor Discovery Protocol*)** debe funcionar con el fin de que el host pueda resolver a partir de la dir. IP, la dir. MAC del dispositivo de destino (equivalente a ARP en IPv4).

El *switch* debe operar sin problema con NDP usando la tabla de tipo ternaria, pero primero se insertarán de forma estática dos entradas NDP en los *hosts* **h1a** y **h1b** en otra terminal poniendo el comando ***make mn-cli*:**

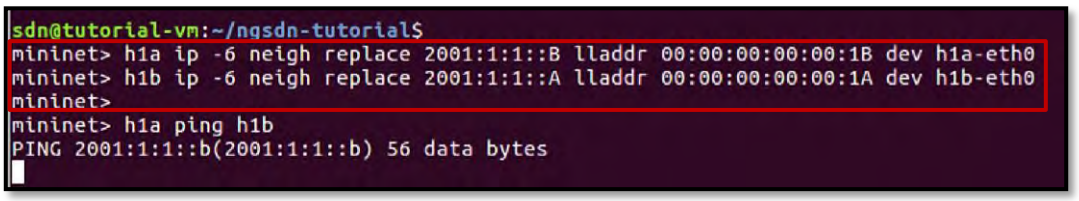


Figura 4-96 Entradas estáticas NDP y prueba de ping – PoC NG-SDN
Fuente: *Autor*

El ping en la *Fig. 4-96* aún no es exitoso pues es necesario añadir dos entradas a la tabla ***l2_exact_table*** en el *switch leaf1* a través de *P4RuntimeShell* y que el equipo realice la conmutación de tramas adecuadamente, una desde el *host h1a* hacia *h1b* y otra viceversa (conectividad de dos vías):

```
RuntimeSh >>> print(te)
able_id: 33605373 ("IngressPipeImpl.l2_exact_table")
atch {
 field_id: 1 ("hdr.ethernet.dst_addr")
 exact {
   value: "\\x00\\x00\\x00\\x00\\x00\\x1b"
 }

ction {
 action {
   action_id: 16812802 ("IngressPipeImpl.set_egress_port")
   params {
     param_id: 1 ("None")
     value: "\\x00\\x04"
   }
 }


RuntimeSh >>> te.insert()
```

Figura 4-97 Entradas de conmutación h1a hacia h1b – PoC NG-SDN
Fuente: *Autor*

```
RuntimeSh >>> te = table_entry["IngressPipeImpl.l2_exact_table"](action = "IngressPipeImpl
          ...: .set_egress_port")

RuntimeSh >>> te.match["hdr.ethernet.dst_addr"] = ("00:00:00:00:00:1A")
ield_id: 1
xact {
 value: "\000\000\000\000\000\032"


RuntimeSh >>> te.action['port_num'] = ("3")
aram_id: 1
alue: "\000\003"


RuntimeSh >>> te.insert()
```

Figura 4-98 Entradas de conmutación h1b hacia h1a – PoC NG-SDN
Fuente: *Autor*

Una vez hecho ello, la conectividad es exitosa:

```
sdn@tutorial-vm:~/ngsdn-tutorial$
mininet> h1a ip -6 neigh replace 2001:1:1::B lladdr 00:00:00:00:00:1B dev h1a-eth0
mininet> h1b ip -6 neigh replace 2001:1:1::A lladdr 00:00:00:00:00:1A dev h1b-eth0
mininet>
mininet> h1a ping h1b
PING 2001:1:1::b(2001:1:1::b) 56 data bytes
64 bytes from 2001:1:1::b: icmp_seq=648 ttl=64 time=1.64 ms
64 bytes from 2001:1:1::b: icmp_seq=649 ttl=64 time=1.51 ms
64 bytes from 2001:1:1::b: icmp_seq=650 ttl=64 time=1.92 ms
64 bytes from 2001:1:1::b: icmp_seq=651 ttl=64 time=1.51 ms
64 bytes from 2001:1:1::b: icmp_seq=652 ttl=64 time=1.07 ms
64 bytes from 2001:1:1::b: icmp_seq=653 ttl=64 time=3.65 ms
64 bytes from 2001:1:1::b: icmp_seq=654 ttl=64 time=1.58 ms
```

Figura 4-99 Conectividad exitosa entre hosts h1a y h1b – PoC NG-SDN
Fuente: *Autor*

Los siguientes comandos en Mininet muestran en forma de texto la topología de la *Fig. 4-88*:

```
mininet> dump
<IPv6Host h1a: h1a-eth0:10.0.0.1 pid=9>
<IPv6Host h1b: h1b-eth0:10.0.0.2 pid=11>
<IPv6Host h1c: h1c-eth0:10.0.0.3 pid=13>
<IPv6Host h2: h2-eth0:10.0.0.4 pid=15>
<IPv6Host h3: h3-eth0:10.0.0.5 pid=17>
<IPv6Host h4: h4-eth0:10.0.0.6 pid=19>
<StratumBmv2Switch leaf1: lo:127.0.0.1,leaf1-eth1:None,leaf1-eth2:None,leaf1-eth3:Non
e,leaf1-eth4:None,leaf1-eth5:None,leaf1-eth6:None pid=23>
<StratumBmv2Switch leaf2: lo:127.0.0.1,leaf2-eth1:None,leaf2-eth2:None,leaf2-eth3:Non
e,leaf2-eth4:None pid=27>
<StratumBmv2Switch spine1: lo:127.0.0.1,spine1-eth1:None,spine1-eth2:None pid=31>
<StratumBmv2Switch spine2: lo:127.0.0.1,spine2-eth1:None,spine2-eth2:None pid=35>
mininet>
mininet> net
h1a h1a-eth0:leaf1-eth3
h1b h1b-eth0:leaf1-eth4
h1c h1c-eth0:leaf1-eth5
h2 h2-eth0:leaf1-eth6
h3 h3-eth0:leaf2-eth3
h4 h4-eth0:leaf2-eth4
leaf1 lo:  leaf1-eth1:spine1-eth1 leaf1-eth2:spine2-eth1 leaf1-eth3:h1a-eth0 leaf1-et
h4:h1b-eth0 leaf1-eth5:h1c-eth0 leaf1-eth6:h2-eth0
leaf2 lo:  leaf2-eth1:spine1-eth2 leaf2-eth2:spine2-eth2 leaf2-eth3:h3-eth0 leaf2-eth
4:h4-eth0
spine1 lo:  spine1-eth1:leaf1-eth1 spine1-eth2:leaf2-eth1
spine2 lo:  spine2-eth1:leaf1-eth2 spine2-eth2:leaf2-eth2
mininet>
```

Figura 4-100 Infraestructura desde Mininet – PoC NG-SDN
Fuente: *Autor*

YANG y *OpenConfig* en el PoC de NG-SDN

Una de las habilidades propuesta en NG-SDN y que en este PoC corroboraremos es la capacidad de manejar estructuras de datos como YANG.

YANG es un lenguaje de modelado de datos diseñado para configuración de red, originalmente propuesto para NETCONF, tomando la estructura semántica y bases de datos del modelo utilizado por SNMP para sus MIBs. Se usa YANG para expresar estructura de datos, no el dato en sí.

Las instancias de los datos pueden expresarse en XML, JSON, YAML o *Protobuf*[64] siempre que se adhieran al esquema de YANG, esquema que se enfoca en dos puntos:

- Estructura y organización de los datos (estructura de árbol: *Path* y *Leaves*)
- Semántica de los nodos *leaves* (similar al lenguaje natural inglés).

[64] ProtoBuf – Protocol Buffers de Google: https://developers.google.com/protocol-buffers

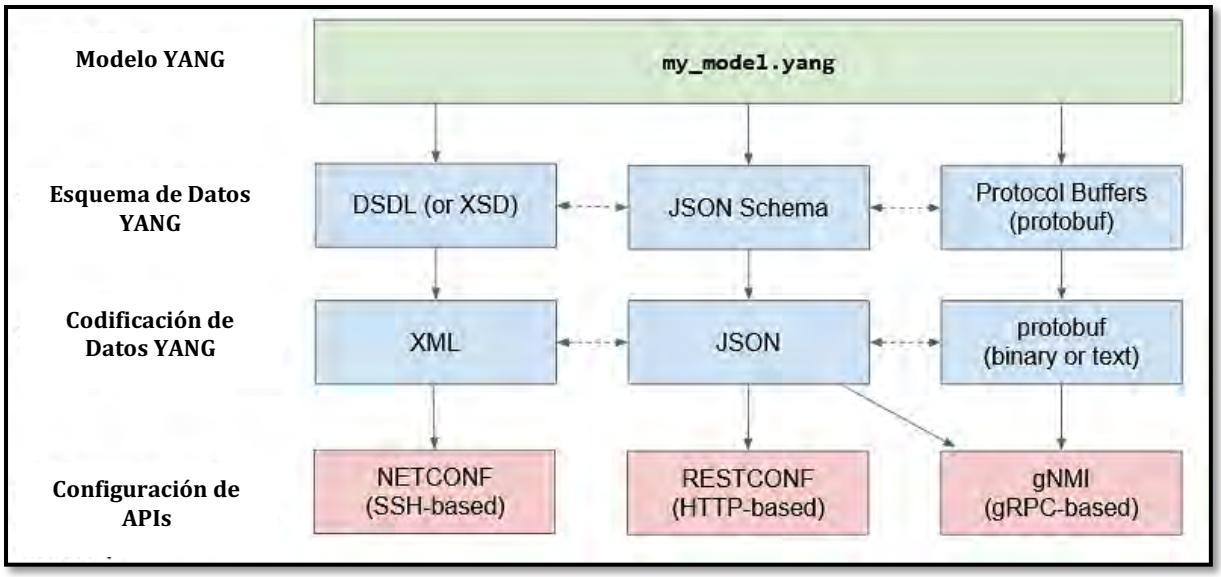


Figura 4-101 YANG en el PoC NG-SDN – gNMI: Interfaz con paradigma Get-Set
Basado en (Cascone, 2019)

La sintaxis YANG se basa en módulos, unidad más elemental para la compilación. La estructura YANG se visualiza en el siguiente *script*:

```
// A module is a self-contained tree of nodes
module demo-port {

  // YANG Boilerplate
  yang-version "1";
  namespace "https://opennetworking.org/yang/demo";
  prefix "demo-port";
  description "Demo model for managing ports";
  revision "2019-09-10" {
    description "Initial version";
    reference "1.0.0";
  }

  // ... insert rest of model here ...
}
```

Script 45 Sintaxis YANG – PoC NG-SDN
Recuperado de (Cascone, 2019)

Los tipos de datos en YANG pueden ser binarios, bits, booleanos, decimales, enteros, cadena de caracteres, etc. La lista completa se encuentra en el RFC 7950 para YANG 1.1.

Una identidad permite expresar una parte del equipo de red en YANG y son jerárquicos.

```
// Identities and Typedefs
identity SPEED {
  description "base type for port speeds";
}

identity SPEED_10GB {
  base SPEED;
  description "10 Gbps port speed";
}

typedef port-number {
  type uint16 {
    range 1..32;
  }
  description "New type for port number that ensure
    the number is between 1 and 32, inclusive";
}
```

Script 46 Identidades y tipos de datos YANG para equipo de Red – PoC NG-SDN Recuperado de (Cascone, 2019)

En caso de reusar nodos, se emplean YANG *Groupings*:

```
// Reusable groupings for port config and state
grouping port-config {
    description "Set of configurable attributes / leaves";
    leaf speed {
        type identityref {
            base demo-port:SPEED;
        }
        description "Configurable speed of a switch port";
    }
}

grouping port-state {
    description "Set of read-only state";
    leaf status {
        type boolean;
        description "Number";
    }
}
```

*Un **Leaf** es un nodo que contiene un valor (built-in o derivado) y no posee hijos*

Script 47 YANG Groupings: Asignación de Velocidad de puerto y lectura del estado del puerto – PoC NG-SDN Basado en (Cascone, 2019)

Un Contenedor YANG por su parte es un nodo que tiene hijos o herederos, donde cada módulo tiene un nivel raíz.

```
container ports {
  description "The root container for port configuration and state";
    list port {
        key "port-number";
        description "List of ports on a switch";

        leaf port-number {
            type port-number;
            description "Port number (maps to the front panel port of a switch);
              also the key for the port list";
        }
```

*Una **Lista** es un nodo que contiene múltiples hijos del mismo tipo. Una lista de elementos se le conoce como **key***

```
        // each individual will have the elements defined in the grouping
        container config {
            description "Configuration data for a port";
            uses port-config; // reference to grouping above
        }
        container state {
            config false; // makes child nodes read-only
            description "Read-only state for a port";
            uses port-state; // reference to grouping above
        }
    }
```

*Contenedores marcados como **config false** son datos de estado de tipo sólo lectura. Típicamente es usado para telemetría y estadística*

Script 48 YANG Containers: Asignación de puertos y lectura del estado del puerto solo lectura – PoC NG-SDN Recuperado de (Cascone, 2019)

Manteniendo el esquema YANG en la representación de un equipo de red, es posible reutilizar, importar o personalizar partes en la configuración.

Para dotar de visibilidad, telemetría y capacidad de administración al entorno NG-SDN, la ONF propuso a *OpenConfig* como el ente para desarrollar modelos de datos que trabajen con independencia del HW y SW, siendo además la base para *switches* **Stratum**.

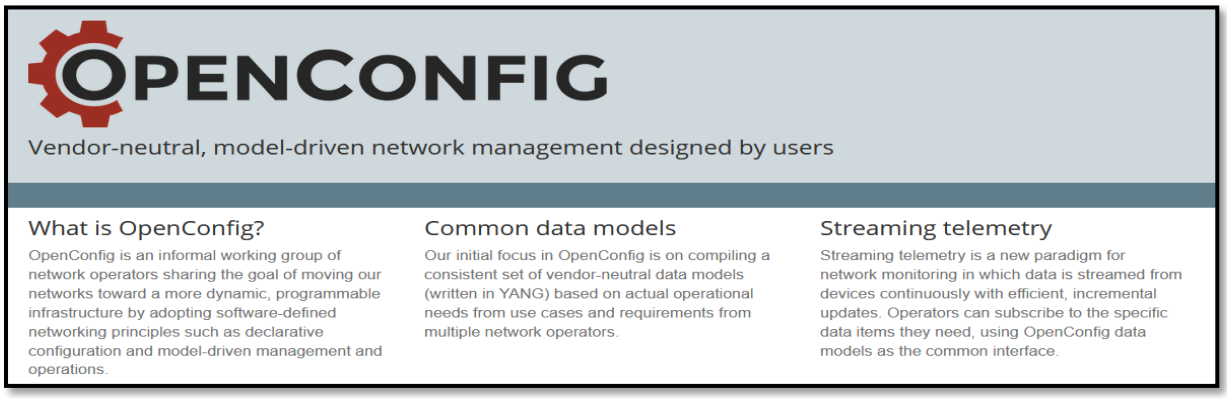


Figura 4-102 OpenConfig en PoC NG-SDN Recuperado de https://www.openconfig.net/

Si bien *OpenConfig* es capaz de definir gran cantidad de modelos, solamente un grupo de ellos es relevante para el plano de datos:

- Interfaces
- VLANs
- LACP (*Link-Aggregation Control Protocol*)
- Plataforma
- QoS
- Sistema

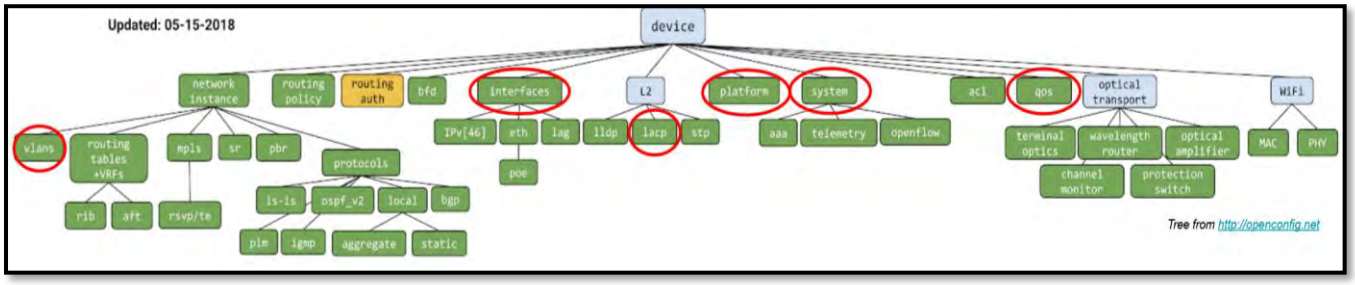


Figura 4-103 Modelos de OpenConfig Recuperado de (Cascone, 2019)

Los fabricantes de HW pueden generar nuevos modelos partiendo de los mencionados en la *Fig. 4-103*.

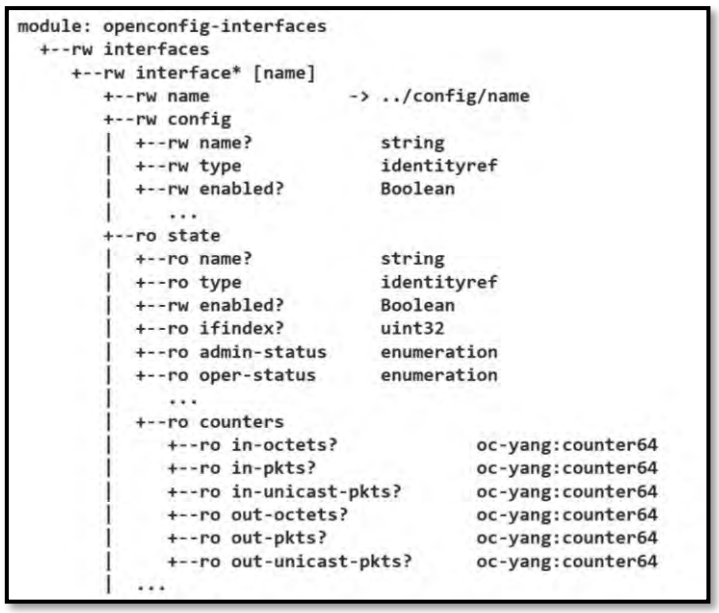


Figura 4-104 Módulo Intefaces en OpenConfig – árbol YANG
Recuperado de (Cascone, 2019)

Otro objetivo de *OpenConfig* para NG-SDN es definir protocolos de transporte modernos y eficientes para configuración de red, telemetría y operaciones: **gNMI** y **gNOI**.

gNMI: *gRPC Network Management Interface*

gNMI es considerada la evolución de NETCONF para la configuración de redes de nueva generación, al usar menos recursos y menos envío de datos al momento de la configuración.

Funcionalmente hablando es una API genérica para leer/escribir estados de configuración de un equipo de red basado en YANG como posible modelo de datos.

Tabla 4-3 Comparación gNMI vs NETCONF

	gNMI	**NETCONF**
Serialización	Protobuf *(Compact Binary Format)*	XML
Transporte	gRPC con HTTP/2.0	SSH
Orientado a Cambios	Sí Retorna solo los elementos que han cambiado desde la última lectura	No Retorna todo el árbol
Soporte Nativo para Telemetría en tiempo real	Sí gRPC soporta streaming bidireccional de telemetría	No Necesita extensión de YANG

Basado en (Cascone, 2019)

gNMI se maneja bajo el paradigma *GET/SET/Subscribe/Capabilites* para interactuar con el plano de datos.

Para poner en práctica los conceptos de YANG, *OpenConfig* y gNMI usaremos el PoC de NG-SDN.

El módulo YANG a emplear se analizó en los *Scripts 45* a *48* (***yang/demo-port.yang***), pero completo se lo puede ver en ***Anexo F: Scripts para la Implementación de PoC NG-SDN.***

Para tener un esquema más sencillo de YANG, se usará la herramienta ***pyang*** y así visualizar el modelo. Primero entraremos a un *Docker* parte de la VM del PoC denominada *yang-tools*:

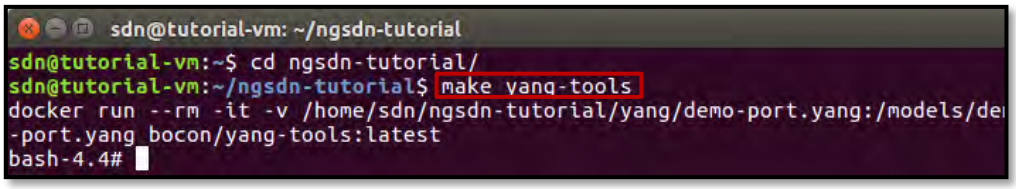


Figura 4-105 Docker yang-tools – PoC NG-SDN
Fuente: *Autor*

Ahora, se correrá *pyang* para observar el modelo YANG analizado en forma de árbol:

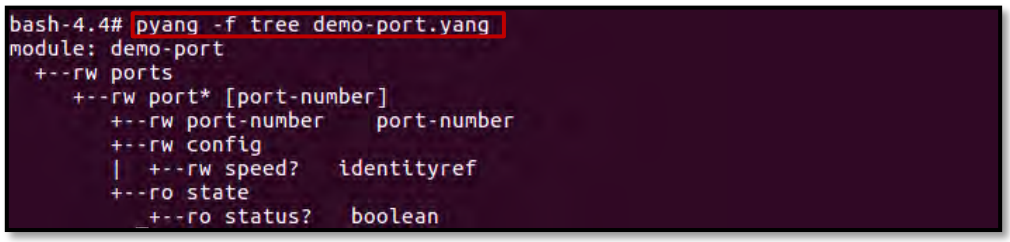


Figura 4-106 Herramienta Pyang para modelo YANG – PoC NG-SDN
Fuente: *Autor*

La representación canónica de YANG de la *Fig. 4-106* es a través de XML, como se observa a continuación. Tanto la *Fig. 4-106* y *Fig.4-107* coinciden en su estructura:

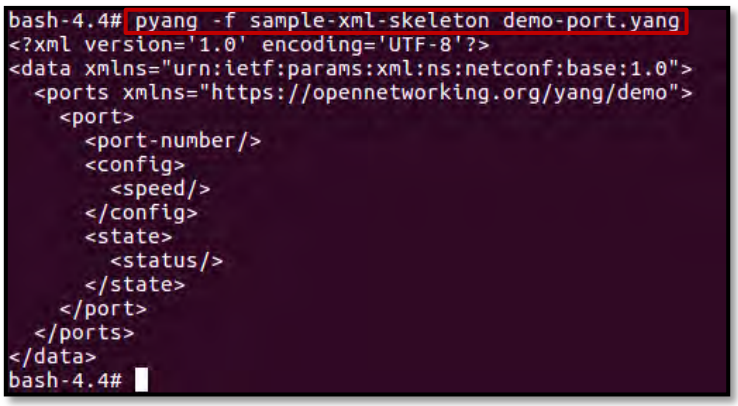
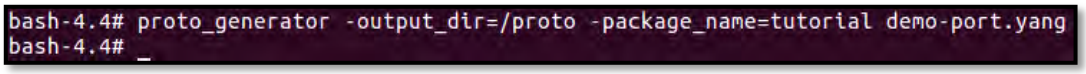


Figura 4-107 Formato canónico XML de modelo YANG – PoC NG-SDN
Fuente: *Autor*

Además de XML, es factible utilizar *Protobuf* como mecanismo de codificación, el cual es más compacto. Se utilizará ***proto_generator*** para generar mensajes *protobuf* del modelo YANG analizado:



El comando anterior genera dos archivos: **demo_port.proto** y **enums.proto.**

Para visualizar el primer archivo se usará el comando:

less /proto/tutorial/demo_port/demo_port.proto

```
// tutorial.demo_port is generated by proto_generator as a protobuf
// representation of a YANG schema.
//
// Input schema modules:
//   - demo-port.yang
syntax = "proto3";

package tutorial.demo_port;

import "github.com/openconfig/ygot/proto/ywrapper/ywrapper.proto";
import "github.com/openconfig/ygot/proto/yext/yext.proto";
import "tutorial/enums/enums.proto";

message Ports {
  message Port {
    message Config {
      tutorial.enums.DemoPortSPEED speed = 349429249 [(yext.schemapath) = "/ports/port/confi
g/speed"];
    }
    message State {
      ywrapper.BoolValue status = 431466463 [(yext.schemapath) = "/ports/port/state/status"]
;
    }
    Config config = 496638493 [(yext.schemapath) = "/ports/port/config"];
    State state = 309256978 [(yext.schemapath) = "/ports/port/state"];
  }
  message PortKey {
    uint64 port_number = 1 [(yext.schemapath) = "/ports/port/port-number"];
/proto/tutorial/demo_port/demo_port.proto
```

Figura 4-108 Protobuf demo_port de modelo YANG – PoC NG-SDN
Fuente: *Autor*

Para el segundo archivo se usará el comando:

less /proto/tutorial/enums/enums.proto

```
// tutorial.enums is generated by proto_generator as a protobuf
// representation of a YANG schema.
//
// Input schema modules:
//   - demo-port.yang
syntax = "proto3";

package tutorial.enums;

import "github.com/openconfig/ygot/proto/ywrapper/ywrapper.proto";
import "github.com/openconfig/ygot/proto/yext/yext.proto";

// DemoPortSPEED represents an enumerated type generated for the YANG identity SPEED.
enum DemoPortSPEED {
  DEMOPORTSPEED_UNSET = 0;
  DEMOPORTSPEED_SPEED_10GB = 58009674 [(yext.yang_name) = "SPEED_10GB"];
}
```

Figura 4-109 Protobuf enums (velocidad de interfaz) de modelo YANG – PoC NG-SDN
Fuente: *Autor*

Hasta lo analizado al momento, existen muchos protocolos agnósticos para modelos YANG que pueden ser usados para obtener datos o configurar equipos: NETCONF, RESTCONF y gNMI.

En esta parte del PoC usaremos *protobuf* codificado sobre gNMI.

Primero, debemos confirmar que el contenedor de Mininet esté corriendo y funcional. Luego, se usará *gNMI Client CLI*[65] y así leer la configuración desde el *switch Stratum leaf1*:

```
sdn@tutorial-vm:~/ngsdn-tutorial$ make start
docker-compose up -d
mininet is up-to-date
onos is up-to-date
sdn@tutorial-vm:~/ngsdn-tutorial$ util/gnmi-cli --grpc-addr localhost:50001 get /
**************************
REQUEST
path {
}
type: CONFIG
encoding: PROTO

**************************
**************************
RESPONSE
notification {
  update {
    path {
    }
    val {
      any_val {
        type_url: "type.googleapis.com/openconfig.Device"
        value: "\252\221\231\304\001\031\n\005leaf1\022\020\242\313\237\312\004\n\250\314\33
3\264\010\276\200\200\224\001\252\221\231\304\001 \n\005:lc-1\022\027\202\377\211H\t\322\315
\356\351\007\003\n\0011\212\212\344\255\007\003\n\0011\252\221\231\304\001X\n\nleaf1-eth1\02
```

Figura 4-110 gNMI Client CLI – PoC NG-SDN
Fuente: *Autor*

La primera parte de la *Fig. 4-110* muestra la petición (*request*) realizada por el CLI y la segunda parte muestra la respuesta (*response*) desde el *Switch Stratum* de tipo *openconfig.Device* como mensaje tipo *protobuf*.

La parte de la respuesta no es legible, por lo que, para tener una buena apreciación, se usa un decodificador mediante el siguiente comando:

$ util/gnmi-cli --grpc-addr localhost:50001 get / | util/oc-pb-decoder | less

```
**************************
REQUEST
path {
}
type: CONFIG
encoding: PROTO

**************************
**************************
RESPONSE
notification {
  update {
    path {
    }
    val {
      any_val {
        type_url: "type.googleapis.com/openconfig.Device"
        value:
component {
  name: "leaf1"
  component {
    chassis {
      platform: OPENCONFIGHERCULESPLATFORMPLATFORMTYPE_GENERIC
    }
  }
}
```

Figura 4-111 gNMI Client CLI con respuesta legible – PoC NG-SDN
Fuente: *Autor*

[65] gNMI Client CLI: https://github.com/Yi-Tseng/Yi-s-gNMI-tool

A través de gNMI es factible obtener información (*get*) de la infraestructura, por ejemplo, para saber el estado operativo de la interfaz eth3 de Leaf1:

```
sdn@tutorial-vm:~/ngsdn-tutorial$
sdn@tutorial-vm:~/ngsdn-tutorial$ util/gnmi-cli --grpc-addr localhost:50001 get \
>     /interfaces/interface[name=leaf1-eth3]/config
**************************
REQUEST
path {
  elem {
    name: "interfaces"
  }
  elem {
    name: "interface"
    key {
      key: "name"
      value: "leaf1-eth3"
    }
  }
  elem {
    name: "config"
  }
}
encoding: PROTO
**************************
**************************
RESPONSE
notification {
  timestamp: 1625705715889486485
  update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "leaf1-eth3"
        }
      }
      elem {
        name: "config"
      }
      elem {
        name: "enabled"
      }
    }
    val {
      bool_val: true
    }
  }
}
notification {
  timestamp: 1625705715889536211
  update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "leaf1-eth3"
        }
      }
      elem {
        name: "config"
      }
      elem {
        name: "health-indicator"
      }
    }
    val {
      string_val: "GOOD"
    }
  }
}
```

Figura 4-112 gNMI Client CLI – get del estado de Interfaz – PoC NG-SDN
Fuente: *Autor*

De igual manera, se puede configurar la infraestructura (*set*), apagando una de sus interfaces, por ejemplo:

```
sdn@tutorial-vm:~/ngsdn-tutorial$
sdn@tutorial-vm:~/ngsdn-tutorial$ util/gnmi-cli --grpc-addr localhost:50001 set \
>     /interfaces/interface[name=leaf1-eth3]/config/enabled \
>     --bool-val false
***************************
REQUEST
update {
  path {
    elem {
      name: "interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
        value: "leaf1-eth3"
      }
    }
    elem {
      name: "config"
    }
    elem {
      name: "enabled"
    }
  }
  val {
    bool_val: false
  }
}
***************************
***************************
RESPONSE
prefix {
}
response {
  path {
    elem {
      name: "interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
        value: "leaf1-eth3"
      }
    }
    elem {
      name: "config"
    }
    elem {
      name: "enabled"
    }
  }
  op: UPDATE
}
timestamp: 1625706504090264207

***************************
```

Figura 4-113 gNMI Client CLI – set para pagar una Interfaz – PoC NG-SDN
Fuente: *Autor*

Para encender nuevamente la interfaz, el comando sería:

```
$ util/gnmi-cli --grpc-addr localhost:50001 set \
  /interfaces/interface[name=leaf1-eth3]/config/enabled \
  --bool-val true
```

Para finalizar este PoC, interactuaremos con el Controlador SDN que hace posible la conectividad entre los nodos *Stratum*: **ONOS**.

El comando ***make start*** inicializa tanto Mininet como ONOS, por lo que se puede ingresar a su CLI en este PoC de la siguiente forma. La contraseña es *rocks*:

```
sdn@tutorial-vm:~/ngsdn-tutorial$ make onos-cli
*** Connecting to the ONOS CLI... password: rocks
*** Top exit press Ctrl-D
Password:
Welcome to Open Network Operating System (ONOS)!

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'logout' to exit ONOS session.

onos@root >
```

Figura 4-114 Inicialización Controlador ONOS – PoC NG-SDN
Fuente: *Autor*

Las aplicaciones habilitadas en ONOS para un entorno NG-SDN son las siguientes:

```
onos@root >
onos@root > apps -a -s
*   8 org.onosproject.protocols.grpc       2.2.2    gRPC Protocol Subsystem
*   9 org.onosproject.protocols.gnmi       2.2.2    gNMI Protocol Subsystem
*  10 org.onosproject.generaldeviceprovider 2.2.2    General Device Provider
*  36 org.onosproject.protocols.p4runtime  2.2.2    P4Runtime Protocol Subsystem
*  37 org.onosproject.p4runtime            2.2.2    P4Runtime Provider
*  38 org.onosproject.drivers              2.2.2    Default Drivers
*  39 org.onosproject.drivers.p4runtime    2.2.2    P4Runtime Drivers
*  40 org.onosproject.pipelines.basic      2.2.2    Basic Pipelines
*  61 org.onosproject.hostprovider         2.2.2    Host Location Provider
*  62 org.onosproject.lldpprovider         2.2.2    LLDP Link Provider
*  73 org.onosproject.drivers.gnmi         2.2.2    gNMI Drivers
*  90 org.onosproject.protocols.gnoi       2.2.2    gNOI Protocol Subsystem
* 114 org.onosproject.drivers.gnoi         2.2.2    gNOI Drivers
* 115 org.onosproject.drivers.stratum      2.2.2    Stratum Drivers
* 116 org.onosproject.drivers.bmv2         2.2.2    BMv2 Drivers
* 144 org.onosproject.gui2                 2.2.2    ONOS GUI2
onos@root >
```

Figura 4-115 Aplicaciones ONOS – PoC NG-SDN
Fuente: *Autor*

Tal como se analizó en **2.7 Controladores SDN**, ONOS define APIs para interactuar con la infraestructura del plano de datos, las cuales se denominan Comportamientos (*Behaviors*). Un *Driver* es una colección de uno o más *Behaviors*.

Entre las APIs más importantes para el entorno NG-SDN son:

- *DeviceDescriptionDiscovery:* API que lee información del dispositivo y Puerto.
- *FlowRuleProgrammable:* Lee/Escribe reglas de flujos.

- *PortStatisticsDiscovery*: Muestra datos estadísticos de los puertos de un dispositivo (Telemetría).
- *Pipeliner:* Mapeo de la lógica de envío en la red.

Si bien ONOS fue diseñado para trabajar junto a *OpenFlow*, extendió sus funcionalidades en NG-SDN para que usuarios puedan cargar programas/apps escritos en P4 sin inconvenientes.

Algo adicional sobre ONOS en NG-SDN es generar su propio *Pipeline* para establecer el envío de tráfico en la red, el cual se ha denominado *Pipeconf*.

Pipeconf es una paquetería provista a ONOS como una app que une todo lo necesario para que ONOS entienda, controle e implemente diferentes *pipelines*.

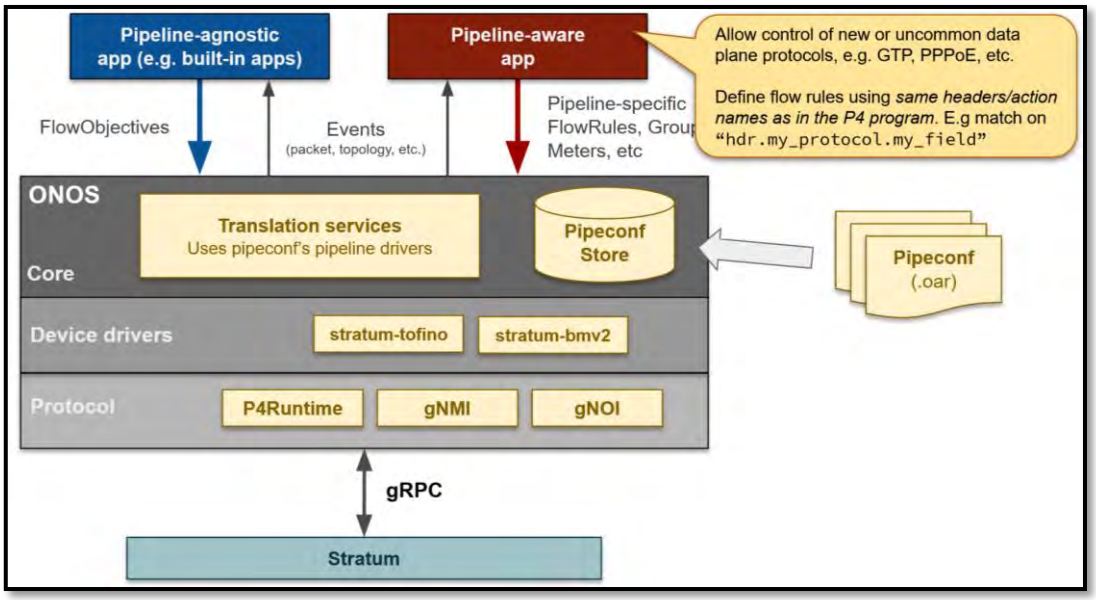


Figura 4-116 Pipeconf soportado por ONOS – PoC NG-SDN
Recuperado de (Cascone, 2019)

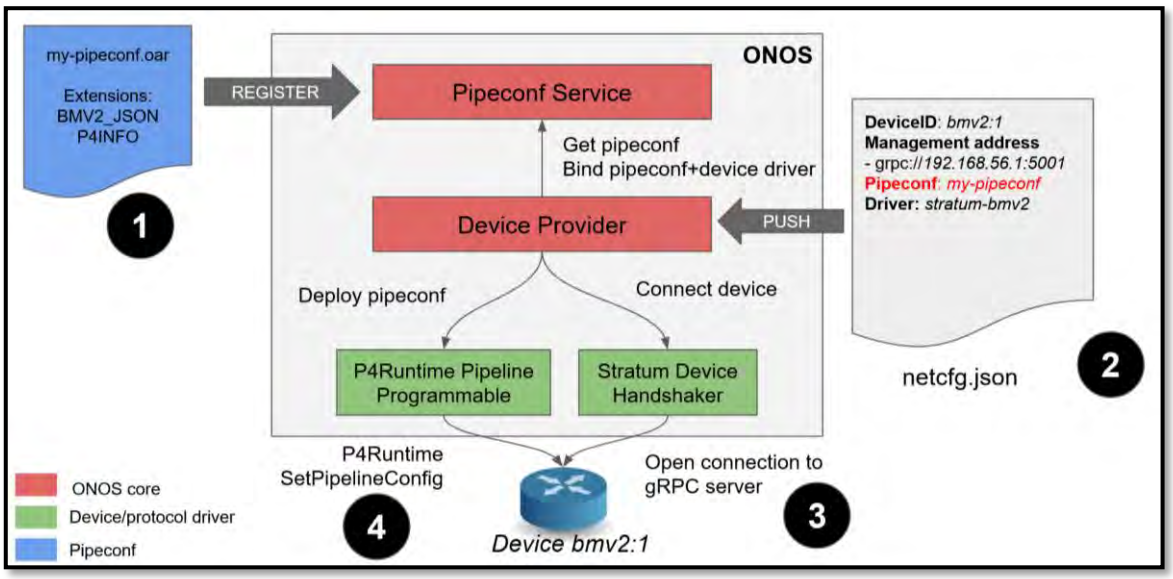


Figura 4-117 Flujograma Pipeconf en ONOS – PoC NG-SDN
Recuperado de (Cascone, 2019)

Al usar *Pipeconf* se generan tres fases de traducción y así el Plano de Datos interactúe con el Plano de Control:

- *Flow Objetive* a *Flow Rule*: Mapea un flujo objetivo a muchas reglas de flujo.
- *Flow Rule* a *Table Entry*: Mapea acciones estándar tipo ONOS a P4.
- *Table Entry* a *P4Runtime Message*: Mapea nombres a IDs en P4.

En el PoC, se requiere de cargar la App (***make app-reaload***) y registrar la infraestructura a *Pipeconf*. Para ello, se activa la app previamente construida (se encuentra en el siguiente *path* de la VM de NG-SDN: ***app/target/ngsdn-tutorial-1.0-SNAPSHOT.oar***)

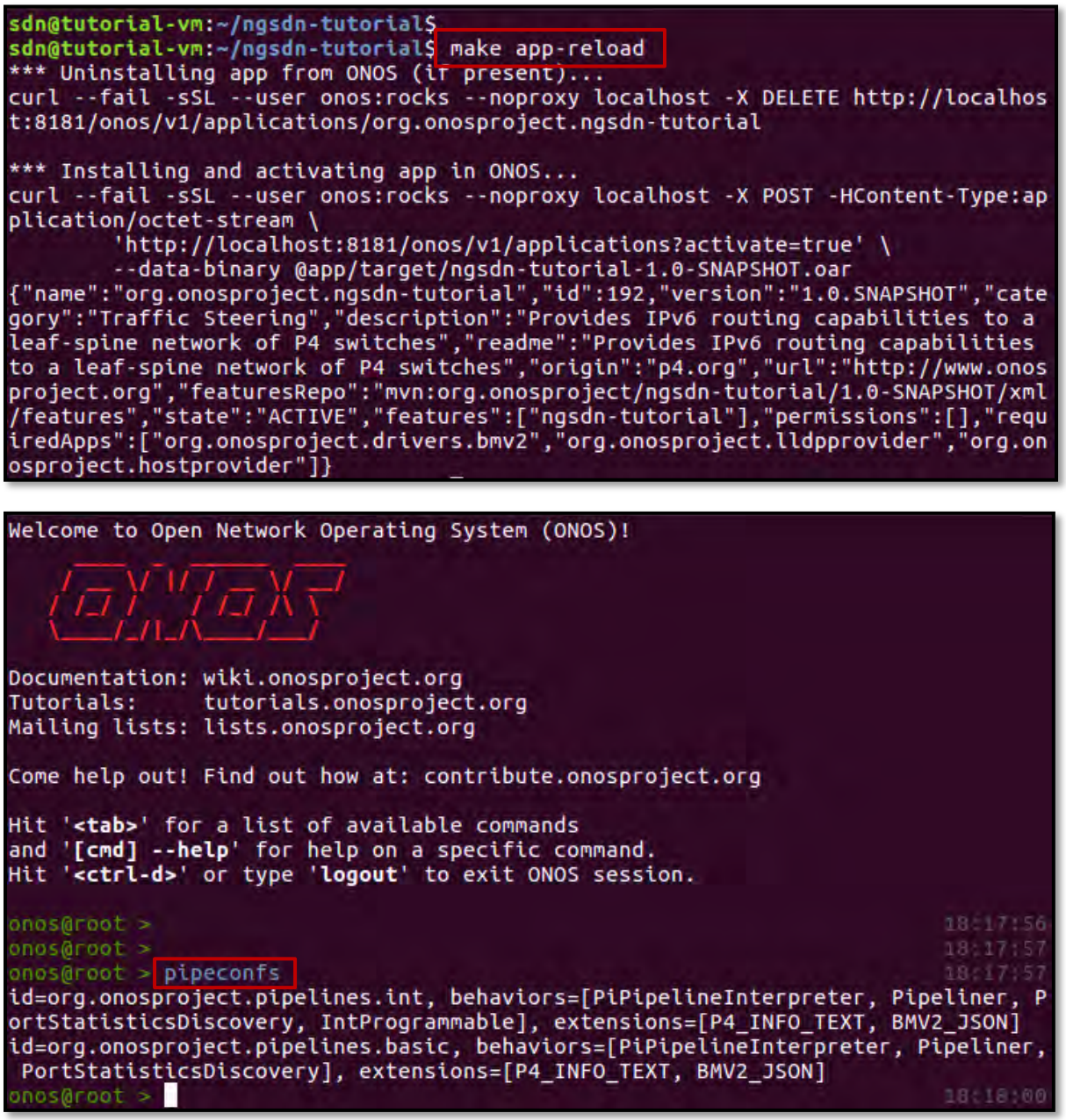


Figura 4-118 Carga de App Pipeconf ONOS – PoC NG-SDN
Fuente: *Autor*

Una vez que ONOS y Mininet está corriendo, es momento de que ONOS encuentre los *switches* de la infraestructura y los controle mediante un archivo de configuración de nombre **netcfg.json**

el cual contiene información como la dirección gRPC y los puertos asociados a cada nodo *Stratum*, los *Driver* ONOS para cada equipo (*stratum-bmv2*), el *pipeconf* para la infraestructura (*PipeconfLoader.java*). Para observar estos archivos, diríjase a ***Anexo F: Scripts para la Implementación de PoC NG-SDN.***

En una nueva terminal, mediante el comando ***make netcfg*** se realiza un *push* del archivo *netcfg.json* a ONOS y así descubrir los *switches Stratum*.

```
sdn@tutorial-vm:~/ngsdn-tutorial$ make netcfg
*** Pushing netcfg.json to ONOS...
curl --fail -sSL --user onos:rocks --noproxy localhost -X POST -H 'Content-Type:applic
ation/json' \
       http://localhost:8181/onos/v1/network/configuration -d@./mininet/netcfg.json
```

Figura 4-119 Push del archivo netcfg.json a ONOS – PoC NG-SDN
Fuente: *Autor*

Para comprobar que la configuración fue enviada correctamente, en ONOS se usa el comando:

```
onos@root >
onos@root > netcfg
{
  "devices" : {
    "device:spine2" : {
      "fabricDeviceConfig" : {
        "myStationMac" : "00:bb:00:00:00:02",
        "isSpine" : true
      },
      "basic" : {
        "managementAddress" : "grpc://mininet:50004?device_id=1",
        "driver" : "stratum-bmv2",
        "pipeconf" : "org.onosproject.ngsdn-tutorial",
        "locType" : "grid",
        "gridX" : 600,
        "gridY" : 400
      }
    },
```

Figura 4-120 Comprobación de Push del archivo netcfg.json a ONOS – PoC NG-SDN
Fuente: *Autor*

A partir de ese momento, ONOS ha descubierto los cuatro *switches* de la infraestructura

```
onos@root > devices -s                                                  02:16:42
id=device:leaf1, available=true, role=MASTER, type=SWITCH, driver=stratum-bmv2:org.onos
project.ngsdn-tutorial
id=device:leaf2, available=true, role=MASTER, type=SWITCH, driver=stratum-bmv2:org.onos
project.ngsdn-tutorial
id=device:spine1, available=true, role=MASTER, type=SWITCH, driver=stratum-bmv2:org.ono
sproject.ngsdn-tutorial
id=device:spine2, available=true, role=MASTER, type=SWITCH, driver=stratum-bmv2:org.ono
sproject.ngsdn-tutorial
onos@root >                                                             02:16:55
onos@root > links                                                       02:16:56
src=device:leaf1/1, dst=device:spine1/1, type=DIRECT, state=ACTIVE, expected=false
src=device:leaf1/2, dst=device:spine2/1, type=DIRECT, state=ACTIVE, expected=false
src=device:leaf2/1, dst=device:spine1/2, type=DIRECT, state=ACTIVE, expected=false
src=device:leaf2/2, dst=device:spine2/2, type=DIRECT, state=ACTIVE, expected=false
src=device:spine1/1, dst=device:leaf1/1, type=DIRECT, state=ACTIVE, expected=false
src=device:spine1/2, dst=device:leaf2/1, type=DIRECT, state=ACTIVE, expected=false
src=device:spine2/1, dst=device:leaf1/2, type=DIRECT, state=ACTIVE, expected=false
src=device:spine2/2, dst=device:leaf2/2, type=DIRECT, state=ACTIVE, expected=false
```

Figura 4-121 Descubrimiento de Switches Stratum y sus enlaces en ONOS – PoC NG-SDN
Fuente: *Autor*

ONOS posee una GUI interactiva que también es factible de ingresar desde la VM del PoC abriendo un navegador con la URL **http://127.0.0.1:8181/onos/ui**. El usuario es **onos** y la contraseña es **rocks**.

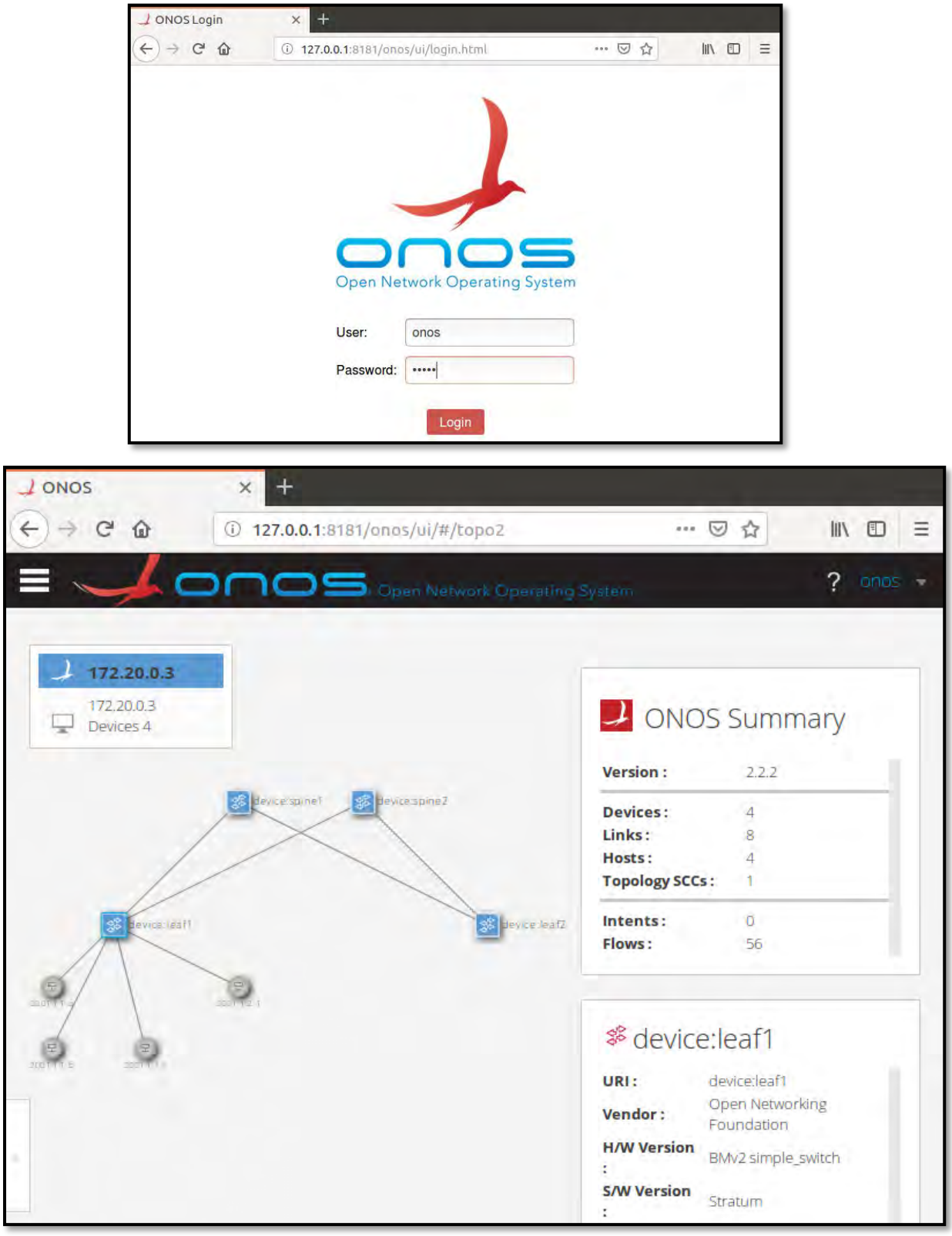


Figura 4-122 ONOS GUI – PoC NG-SDN
Fuente: *Autor*

4.5 PoC de SD-WAN en EVE-ng

En ***3.3 Software-Defined WAN: Propuesta de rediseño del transporte de información y control de camino en redes Underlay y Overlay*** se analizó los conceptos básicos de *SD-WAN*, profundizando el estudio de SD-WAN Viptela como una solución robusta para redes de transporte de próxima generación.

En este PoC se pondrá a prueba los conceptos teóricos llevados a la práctica en un entorno de emulación muy real en EVE-ng.

4.5.1 Cisco *DevNet Sandbox* y *dCloud*

EVE-ng, si bien es un emulador excelente para este tipo de PoCs, se requiere de una computadora de grandes prestaciones (al menos 32 a 64GB de RAM, con 4-8 núcleos y CPU robusto), por ello existen entornos alternativos, donde quizá no toda la capacidad de personalización existe, pero son adecuados para dar un vistazo a nuevas tecnologías. Uno de ellos es el entorno conocido como *DevNet Sandbox*, el cual es un ambiente de pruebas desarrollado por la Comunidad de DevNet con el fin de realizar laboratorios de diversos tópicos tecnológicos, desde Redes Tradicionales, Seguridad en Tecnologías de la Información, Redes de Próxima Generación, *Blockchain*, APIs, lenguajes de programación, Tecnologías *OpenSource,* entre otros, totalmente gratis y abierto al público.

Para acceder, basta con ingresar a esta URL y usar las credenciales de GitHub, Gmail, Facebook, Cisco ID (NetAcad) o Webex:

https://developer.cisco.com/site/sandbox/

Existen dos tipos de *Sandboxes* en *DevNet*:

- *Always-On*: Este tipo de laboratorios de prueba están siempre disponibles y no requiere de una reservación previa. Es ideal para dar un primer vistazo a una determinada tecnología o producto. Este ambiente es "compartido" entre todos los usuarios de momento.
- *Reservation:* Este tipo de laboratorios requiere de una reserva previa y tiene una duración máxima de 8 horas continuas. La principal ventaja es la posibilidad de contar con acceso administrativo total y exclusivo a todos los nodos, razón por la cual se establece una conexión a través de una VPN. La instanciación de estos laboratorios puede ser entre 10-40 minutos dependiendo del tópico.

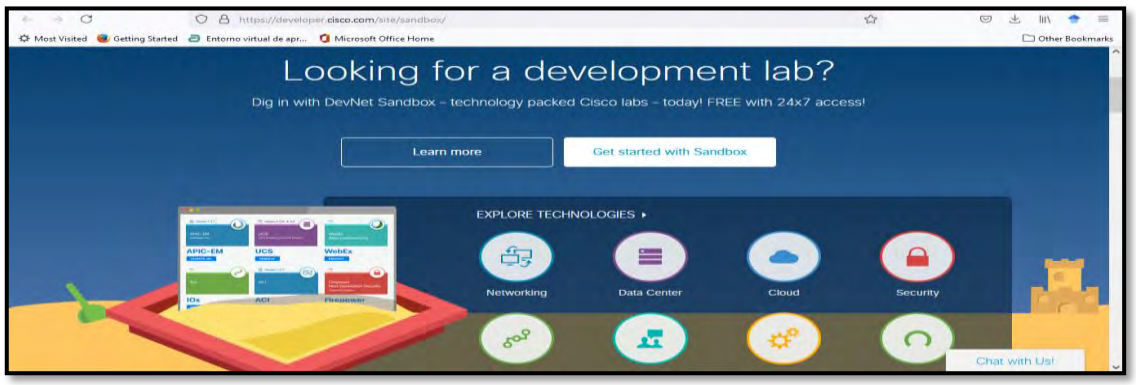


Figura 4-123 Página principal de DevNet Sandbox– PoC SDWAN[66]
Fuente: *Autor*

[66] DevNet Sandbox Documentación: https://developer.cisco.com/docs/sandbox/#!getting-started/what-is-devnet-sandbox

Por otro lado, en un entorno más enfocado hacia Pruebas de Concepto empresariales, está *dCloud*, entorno en el que se pone en marcha laboratorios de tópicos vanguardistas y probar soluciones Cisco mediante un sistema de reserva del entorno y conexión a través de VPNs a los servidores de emulación. Este tipo de PoCs son demos precargados y guiados de temas como DNA (*Digital Network Architecture*), *Secure SD-WAN* con integración *Cloud*, ISE (*Identity Services Engine*), etc.

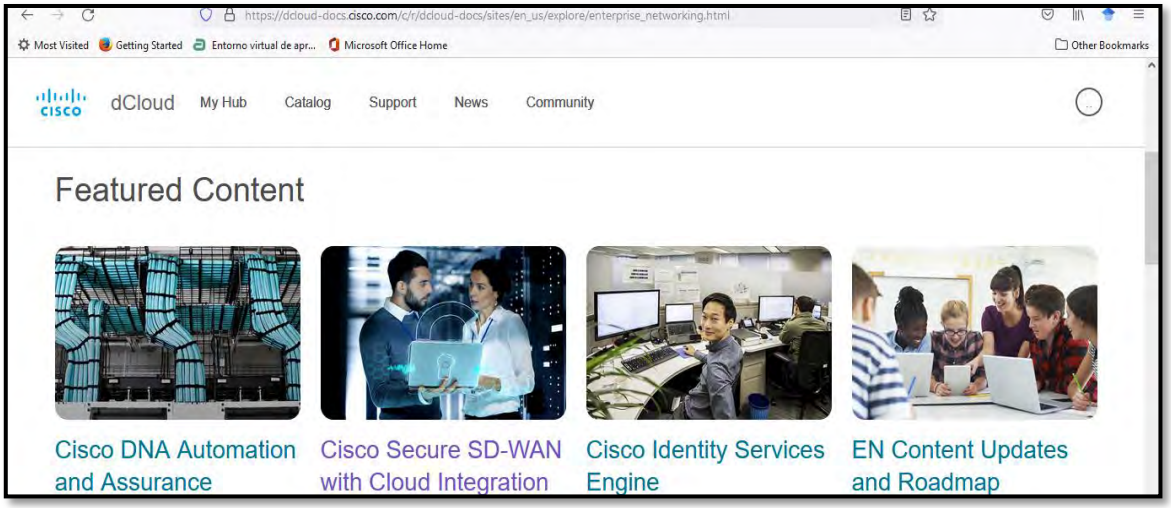


Figura 4-124 Página de tópicos de dCloud– PoC SDWAN[67]
Fuente: *Autor*

Para acceder, basta con ingresar a esta URL y usar las credenciales de GitHub, Gmail, Facebook, Cisco ID (NetAcad) o Webex:

https://dcloud-docs.cisco.com/c/r/dcloud-docs/sites/en_us/explore/enterprise_networking.html

4.5.2 SD-WAN en la vida real

En ***3.3.3 SD-WAN: Componentes de la Implementación tipo Viptela***, se fundamentó a esta tecnología, definiendo su terminología y funciones clave.

Ahora, se tendrá una visión de cómo SD-WAN Viptela puede implementarse en el mundo real. Para ello, es importante mencionar cómo se evolucionó del enrutamiento tradicional a los nuevos paradigmas basados en SDN.

En un enrutamiento tradicional existen dos tipos de paquetes que son enviados y recibidos por un *router*:

- Paquetes de Enrutamiento: Mensajes generados por los protocolos de enrutamiento para formar la RIB (*Routing Information Base*) y su forma comprimida a nivel de *Hardware*, FIB (*Forwarding Information Base*).
- Paquetes enrutados: Mensajes que deben ser enrutados por el equipo. Son los mensajes de usuario.

[67] DevNet Sandbox Documentación: https://developer.cisco.com/docs/sandbox/#!getting-started/what-is-devnet-sandbox

La siguiente figura muestra las partes fundamentales de un *router* tradicional.

Figura 4-125 Router Tradicional – PoC SDWAN
Recuperado de (Prabhu, 2019)

En la *Fig. 4-125* se aprecian los tres elementos principales que permiten que un paquete de datos sea enrutado de forma correcta:

- Módulo de Entrada/Salida: Módulo por el que ingresan tanto los paquetes de enrutamiento como los enrutados. Son las interfaces físicas.
- Módulo de Plano de Control: CPU del equipo donde se forma la RIB y una copia resumida pasa al Módulo de E/S en forma de FIB.
- *Switch Fabric*: Conexiones de *backplane* que interconecta el Módulo E/S con el Plano de Control.

En un entorno SD-WAN, esos elementos se desacoplan del equipo y se distribuyen en la infraestructura con el fin de dotar de flexibilidad, escalabilidad y resiliencia a la red.

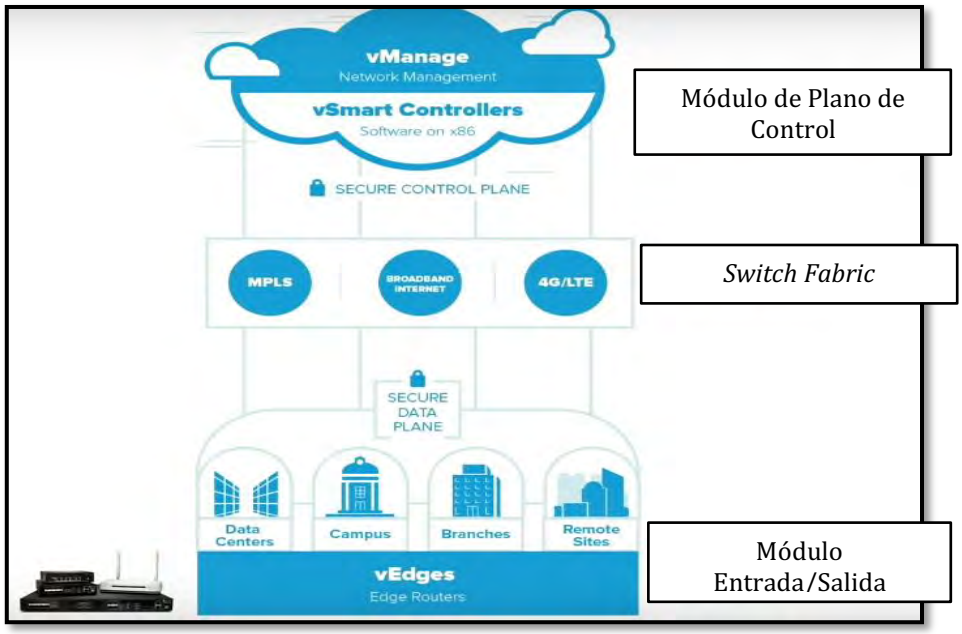


Figura 4-126 Esquema de SDWAN – PoC SDWAN
Recuperado de (Prabhu, 2019)

Según la *Fig. 4-126*, el módulo E/S se conoce como *vEdges* al ser equipos virtuales/*Cloud* o *cEdges* al ser equipos físicos, el plano de control pasa a ser *vManage/vSmart* y el *Switch Fabric* es el mismo Internet, ya que este se ha convertido en el medio de transporte ideal, pues ahora es predecible, estable y lo más seguro posible.

Con este esquema, no es necesario que los equipos se hablen entre ellos para llegar al punto de convergencia y tengan una imagen individual de la red a través de sus RIB/FIB, en el paradigma SD-WAN, un Plano de Control (*vSmart*) común, normalmente ubicado en la nube, cambia una política (en *vManage*) y esta se distribuye (*pushes*) desde *vSmart* a los *vEdges* mediante OMP.

Como se comentó anteriormente, OMP (*Overlay Management Protocol*), permite el establecimiento del canal de comunicación entre los *vEdges* y el Plano de Control (*vSmart*) y el enrutamiento coherente de todo el entorno, canal que se asegura mediante DTLS/TLS. Con ello, el *Overlay* para SD-WAN son los túneles inteligentes de comunicación que tienen al Internet como *Fabric* entre el plano de datos y el de control.

El elemento faltante es el orquestador de la red, denominado *vBond*, el cual permite que los *vEdges* conozcan donde están los *vManage/vSmart*. Este es el único elemento de la arquitectura SD-WAN Viptela que debe estar en la red pública, pues es el puente inicial de comunicación al que los equipos llegan al inicializarse y posteriormente se establece el *Overlay* final mediante OMP.

También, se genera el concepto de *Secure Extensible Network* o SEN, el cual se basa en el uso de llaves públicas-privadas (*keys*) para mantener la encriptación, autenticación y control de acceso al máximo nivel. Las llaves públicas se mantienen en el plano de control.

En resumen, los paquetes de enrutamiento se envían desde los *vEdges* hacia el plano de control, mientras los paquetes enrutados se envían entre *vEdges* dando conectividad de extremo a extremo entre sedes, es así que, la LAN/VLAN corporativa (red empresarial que puede ser tradicional), está detrás de los *vEdges* para su comunicación. Cabe decir que el canal de comunicación entre *vEdges* se protege mediante IPSec de igual manera al atravesar la red de transporte

Nuevas tecnologías pueden integrarse en estos entornos, un esquema de ello se aprecia en la *Fig. 4-127*.

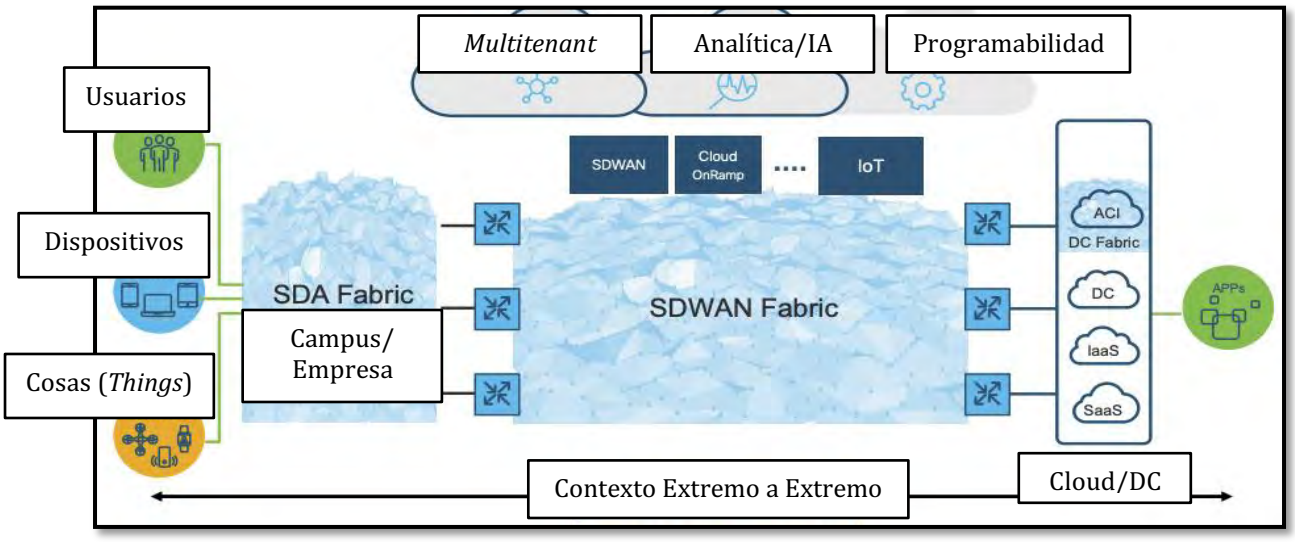


Figura 4-127 Esquema de SDWAN – PoC SDWAN
Recuperado de (Cisco Systems - DevNet Learning Labs, 2021)

4.5.3 *vManage* REST-API

Antes de comenzar el PoC de SDWAN, es necesario mencionar un tema importante en este nuevo paradigma de redes, el uso de APIs para controlar, administrar, orquestar y generar telemetría.

API es el acrónimo de *Application Programming Interface*, la cual consiste en un conjunto de reglas que describen cómo una aplicación puede interactuar con otra aplicación y el mecanismo que permite que esa interacción tenga efecto. En ***2.3.2 Application Programming Interfaces - APIs***, se ahonda más en ese concepto, además de identificar distintos tipos de APIs, pero el que interesa para el contexto de SD-WAN son las REST-APIs.

REST es el término usado para *Representational State Transfer*, el cual establece una comunicación similar a HTTP/HTTPS, usando métodos similares para extraer información de la infraestructura (*get, post, put, delete*) y pueden retornar información en formatos como XML y JSON. La *Fig. 4-128* muestra ese comportamiento:

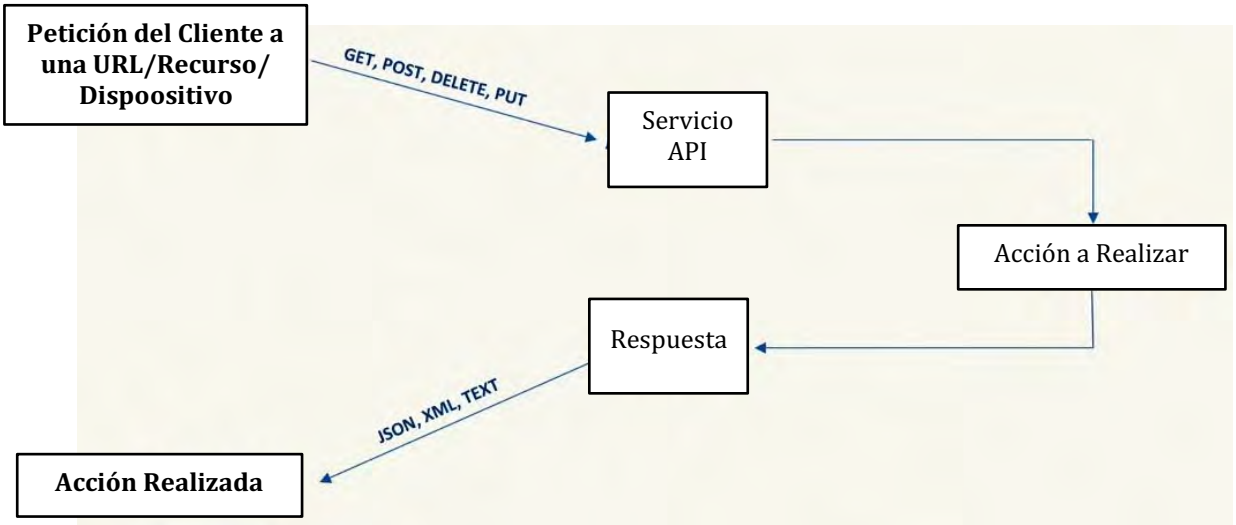


Figura 4-128 Esquema de funcionamiento de una API – PoC SDWAN
Recuperado de (Prabhu, 2019)

Uno de los elementos fundamentales de la solución SDWAN Viptela es el *vManage*, encargado como su nombre lo indica, del contacto/manejo entre la infraestructura y el administrador de la red. Para mejorar la experiencia de administración/telemetría, *vManage* empleará el concepto de REST-APIs para establecer la interfaz entre la red SDWAN y el manejo de esta para conseguir los objetivos empresariales planteados.

4.5.4 Topología del PoC de SDWAN Viptela: Prueba de Concepto en EVE-ng

La topología usada para el PoC de SD-WAN se indica en la *Fig. 4-129*, la cual fue diseñada en base a la instalación realizada en ***Anexo G: Proceso de Instalación de SDWAN Viptela en EVE-ng.***

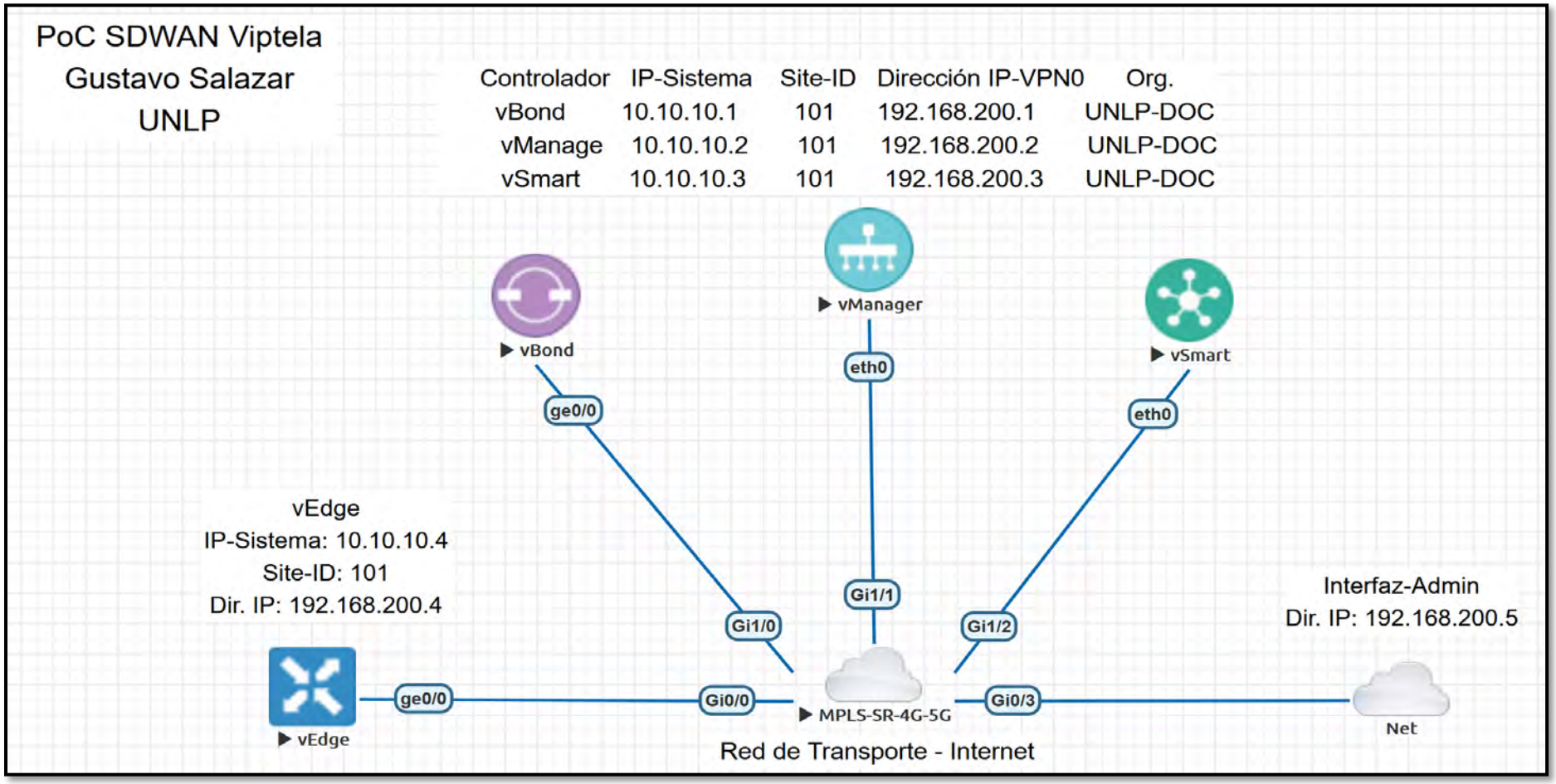


Figura 4-129 Topología SDWAN Viptela – PoC SDWAN
Fuente: *Autor*

Tal como se indicó en el proceso de instalación, los nodos se irán encendiendo uno a uno comenzando por *vManage*.

Configuración Inicial *vManage*

Una vez se dio *play* al nodo en EVE-ng, se da doble clic sobre él para ingresar a su CLI.

Las credenciales de ingreso por defecto son ***admin/admin***. Una vez el sistema esté listo, se ingresará a la CLI de *vManage*, el cual pedirá el cambio de credenciales, que para el caso del PoC se configuró como ***admin/*UNLPDOC;** luego, se indica el disco de almacenamiento para el *vManage,* el cual es el de 100GB que se creó en el proceso de instalación, por lo que se selecciona 1 y el proceso de inicialización arranca. Este proceso se muestra en la *Fig. 4-130.*

```
viptela 19.2.4

vmanage login: admin
Password:
Welcome to Viptela CLI
admin connected from 127.0.0.1 using console on vmanage
You must set an initial admin password.
Password:
Re-enter password:
Available storage devices:
vdb     100GB
hdc     3GB
1) vdb
2) hdc
Select storage device to use: 1
Would you like to format vdb? (y/n): y
```

```
Mon Jul 12 05:36:01 UTC 2021: System Ready


viptela 19.2.4

vmanage login: admin
Password:
Welcome to Viptela CLI
admin connected from 127.0.0.1 using console on vmanage
vmanage#
vmanage#
```

Figura 4-130 Inicialización de vManage – PoC SDWAN
Fuente: *Autor*

Posteriormente, se configura los parámetros base establecidos en la *Fig. 4-129* correspondiente a la topología de este PoC, indicando además la dir. IP del *vBond* y su direccionamiento IP en la VPN0 para que pueda encontrarlo.

```
vmanage# config
Entering configuration mode terminal
vmanage(config)# system
vmanage(config-system)# system-ip 10.10.10.2
vmanage(config-system)# site-id 101
vmanage(config-system)# organization-name UNLP-DOC
vmanage(config-system)# vbond 192.168.200.1
vmanage(config-system)# exit
vmanage(config)#
```

```
vmanage(config)# vpn 0
vmanage(config-vpn-0)# interface eth0
vmanage(config-interface-eth0)# ip address 192.168.200.2/24
vmanage(config-interface-eth0)# no shut
vmanage(config-interface-eth0)# exit
vmanage(config-vpn-0)#
vmanage(config-vpn-0)# commit and-quit
Commit complete.
vmanage#
```

Script 49 Configuración básica de vManage – Commit - PoC SDWAN
Fuente: *Autor*

No olvide que la VPN0 en el entorno SDWAN es la conexión hacia el Internet *Fabric*, mientras la VPN512 es para conexión fuera de banda.

Si todo está correcto, será posible acceder al *vManage* desde un navegador Web. Debe existir una conexión *cloud* direccionada hacia la *vmnet* (en *VMWare*) correspondiente. Use las credenciales creadas para su ingreso.

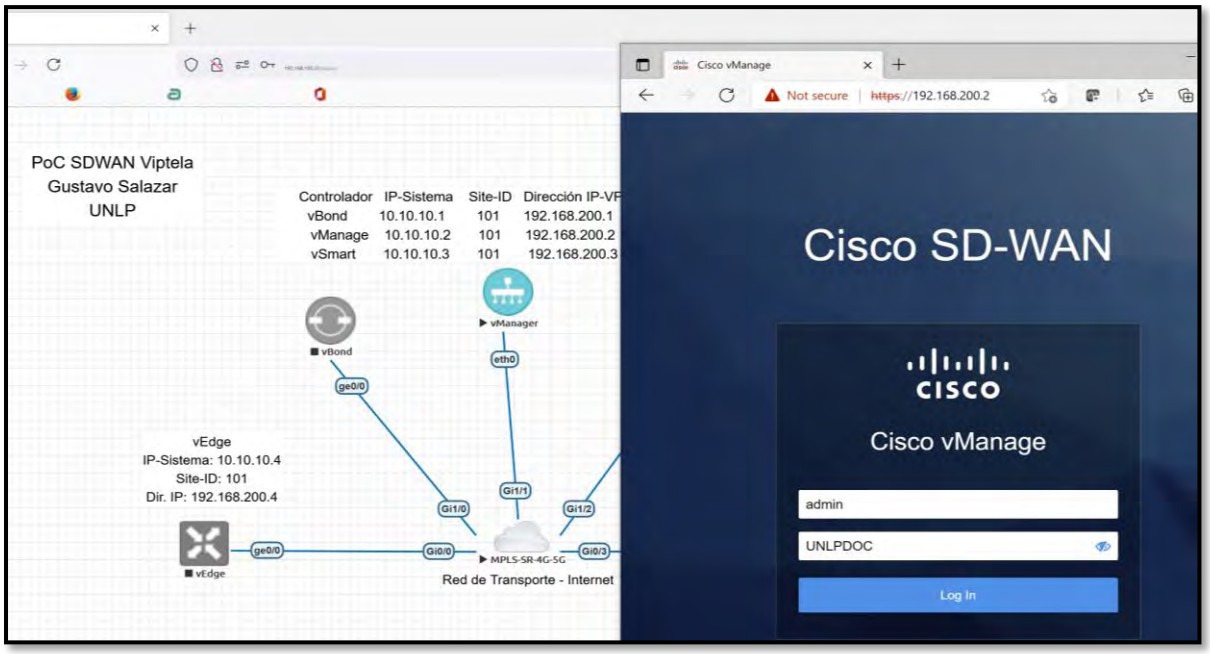


Figura 4-131 Acceso inicial vía Web a vManage - PoC SDWAN
Fuente: *Autor*

Al abrir el *vManage* por primera ocasión, sale una advertencia importante para cambiar el *username/password* de la base de datos que viene por defecto (**neo4j/password**). Se cambiará a **GSalazarUNLP/UNLPdoc123**.

```
vmanage# request nms configuration-db update-admin-user
Enter current user name:neo4j
Enter current user password:password
Enter new user name:GSalazarUNLP
Enter new user password:UNLPdoc123
Successfully updated configuration database admin user
Successfully restarted NMS application server
vmanage#
```

Figura 4-132 Cambio de credenciales en DB de vManage - PoC SDWAN
Fuente: *Autor*

El proceso de configuración del *vManage* culmina en este punto, ahora se procede a realizar la configuración base de los demás nodos, muy similar al del *vManage*. Encienda los otros "controladores". Las contraseñas de ingreso a los equipos se mantendrán: ***admin/UNLPDOC.***

Configuración Inicial *vBond*

Debido a que el *vBond* tiene la misma imagen que un *vEdge*, en este equipo también se cambiará su nombre. No olvide que la conexión desde *vBond/vEdge* debe hacerse desde su Ge0/0 y no desde su eth0.

```
You must set an initial admin password.
Password:
Re-enter password:
vedge#
vedge# config term
Entering configuration mode terminal
vedge(config)# system
vedge(config-system)# host-name vbond
vedge(config-system)# system-ip 10.10.10.1
vedge(config-system)# site-id 101
vedge(config-system)# organization-name UNLP-DOC
vedge(config-system)# vbond 192.168.200.1 local vbond-only
vedge(config-system)# exit
vedge(config)# vpn 0
vedge(config-vpn-0)# int ge0/0
vedge(config-interface-ge0/0)# ip add 192.168.200.1/24
vedge(config-interface-ge0/0)# no shut
vedge(config-interface-ge0/0)# exit
vedge(config-vpn-0)# exit
vedge(config)# commit and-quit
Commit complete.
vbond#
```

Script 50 Configuración básica de vBond - PoC SDWAN
Fuente: *Autor*

Configuración Inicial *vSmart*

```
You must set an initial admin password.
Password:
Re-enter password:
vsmart#
vsmart# config ter
Entering configuration mode terminal
vsmart(config)# system
vsmart(config-system)# system-ip 10.10.10.3
vsmart(config-system)# site-id 101
vsmart(config-system)# organization-name UNLP-DOC
vsmart(config-system)# vbond 192.168.200.1
vsmart(config-system)# exit
vsmart(config)# vpn 0
vsmart(config-vpn-0)# int eth0
vsmart(config-interface-eth0)# ip add 192.168.200.3/24
vsmart(config-interface-eth0)# no shut
vsmart(config-interface-eth0)# exit
vsmart(config-vpn-0)# exit
vsmart(config)# commit and-quit
Commit complete.
```

Script 51 Configuración básica de vSmart - PoC SDWAN
Fuente: *Autor*

Con esas configuraciones, el Plano de Control de la red estaría configurado.

Se debe comprobar conectividad entre *vBond, vSmart y vManage* (la red de transporte debe estar funcionando correctamente – Internet *Fabric*).

```
vbond# ping 192.168.200.2
Ping in VPN 0
PING 192.168.200.2 (192.168.200.2) 56(84) bytes of data.
64 bytes from 192.168.200.2: icmp_seq=1 ttl=64 time=16.3 ms
64 bytes from 192.168.200.2: icmp_seq=2 ttl=64 time=16.6 ms
64 bytes from 192.168.200.2: icmp_seq=3 ttl=64 time=28.6 ms
^C
--- 192.168.200.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 16.309/20.531/28.619/5.722 ms
vbond# ping 192.168.200.3
Ping in VPN 0
PING 192.168.200.3 (192.168.200.3) 56(84) bytes of data.
64 bytes from 192.168.200.3: icmp_seq=1 ttl=64 time=19.7 ms
64 bytes from 192.168.200.3: icmp_seq=2 ttl=64 time=25.3 ms
64 bytes from 192.168.200.3: icmp_seq=3 ttl=64 time=19.9 ms
^C
--- 192.168.200.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 19.746/21.669/25.325/2.588 ms
vbond#
```

Figura 4-133 Conectividad de vBond en el Plano de Control - PoC SDWAN
Fuente: *Autor*

Configuración de Certificados en *vManage* para el Plano de Control

El paso siguiente es la configuración del *vManage* desde *Administration-Settings*, con el fin de confirmar el ***Organization Name***, indicar la dirección IP del *vBond*, así como establecer el uso de Certificados externos para autenticación del Plano de Control (no se usará certificados Cisco).

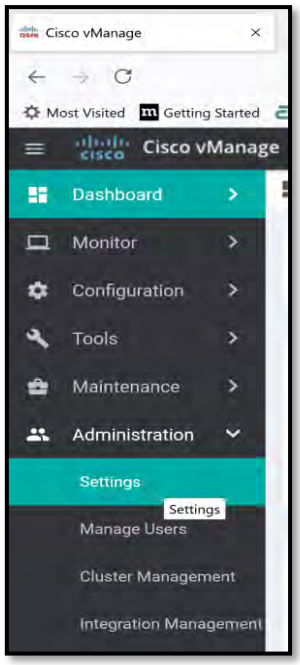


Figura 4-134 Ingreso a Administration-Setting - vManage - PoC SDWAN
Fuente: *Autor*

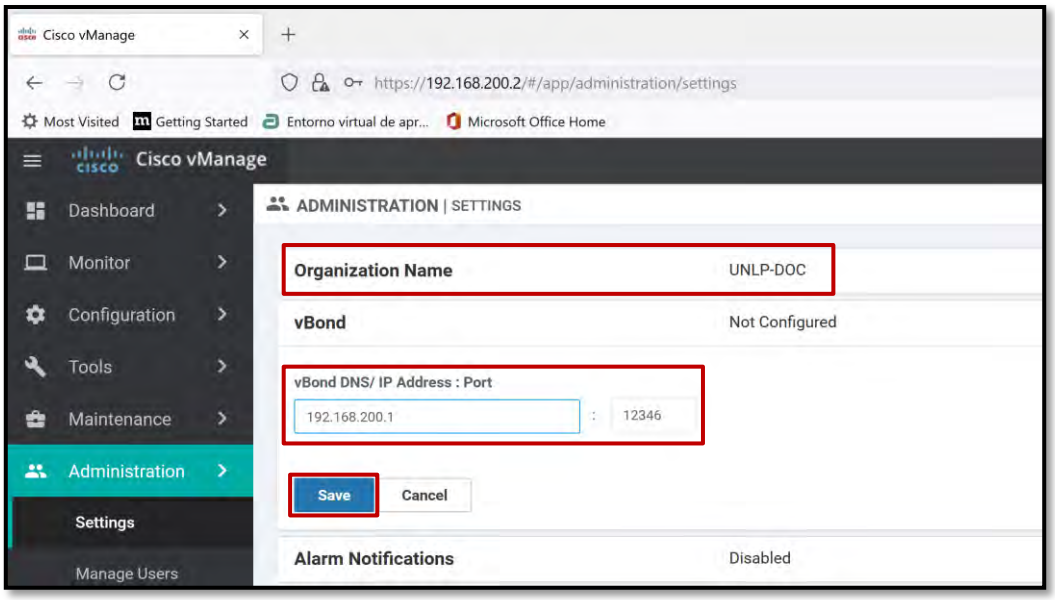


Figura 4-135 Edición de dir. IP vBond en vManage - PoC SDWAN
Fuente: *Autor*

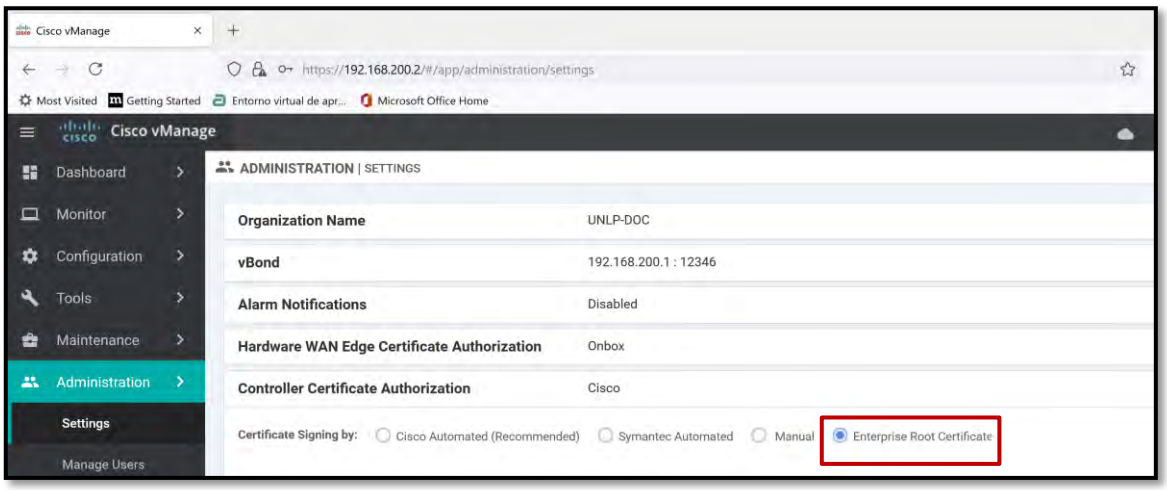


Figura 4-136 Usar certificados externos para autenticación en vManage - PoC SDWAN
Fuente: *Autor*

Se requiere que el *vManage* se convierta en un *root-CA* para que autentique los certificados de los equipos. Eso es factible de hacer mediante la consola *vShell* de dicho equipo.

```
vmanage# vshell
vmanage:~$ openssl genrsa -out ROOTCA.key 2048
Generating RSA private key, 2048 bit long modulus
.....................
.....................................
e is 65537 (0x10001)
vmanage:~$
vmanage:~$ openssl req -x509 -new -nodes -key ROOTCA.key -sha256 -days 2000 \
>     -subj "/C=EC/ST=PICHINCHA/L=QUITO/O=UNLP-DOC/CN=vmanage.lab" \
>     -out ROOTCA.pem
vmanage:~$ ls
ROOTCA.key  ROOTCA.pem  archive_id_rsa.pub
```

Script 52 Creación de Certificado en vManage - PoC SDWAN
Fuente: *Autor*

El *Script 52* muestra cómo se genera los archivos base para la generación de un certificado (ROOTCA.key y ROOTCA.pem) con varias características como protocolo de encriptación/integridad, locación y días de validez.

El contenido del archivo ROOTCA.pem se debe pegar en *vManage*, por lo que se hará un ***cat*** a ese archivo y se pegará su contenido para registrarlo en el *vManage*.

```
vmanage:~$ cat ROOTCA.pem
-----BEGIN CERTIFICATE-----
MIIDhzCCAm+gAwIBAgIJAPdwTs7wNMNQMA0GCSqGSIb3DQEBCwUAMFoxCzAJBgNV
BAYTAkVDMRIwEAYDVQQIDAlQSUNISU5DSEExDjAMBgNVBAcMBVFVSVRPMREwDwYD
VQQKDAhVTkxQLURPQzEUMBIGA1UEAwwLdm1hbmFnZS5sYWIwHhcNMjEwNzEyMDcz
NzAyWhcNMjcwMTAyMDczNzAyWjBaMQswCQYDVQQGEwJFQzESMBAGA1UECAwJUElD
SElOQ0hBMQ4wDAYDVQQHDAVRVUlUTzERMA8GA1UECgwIVU5MUC1ET0MxFDASBgNV
BAMMC3ZtYW5hZ2UubGFiMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA
sOmxcNRd7oP0t0jQdOUORbaLw2kIkjUlUPDefRx4zzvA3T7xECaLAdl23819GaI8
EeUDFDcNX2B6GoNA1CW0KpgfNxN6sWcnWBYxzOvUenG/dKcfOU0kXC8Rs6huRlZR
jEsUmqW6765/+h4197Q+ocJvujt5xTLwq9FV0/dA/CfNcLN9hmZnWDK7PDVwHFyr
xnoq+nvme2nYEYf+laPaNd4Pa3pBMtD8eYwZ0M424+GF9SDROAhK90FvnA7rh+Wu
iYbs6l1UGprBoXzdawLYajyPVs02ntyphmrrTZlKWSdydMLEfBeEjbP9XFpMZNd7
3ltupGyY+BDlqBL7H/V9LwIDAQABo1AwTjAdBgNVHQ4EFgQUY/svV6fla+jggGug
VPUzeI4pVrEwHwYDVR0jBBgwFoAUY/svV6fla+jggGugVPUzeI4pVrEwDAYDVR0T
BAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAiR9JMxsCnas/0KXODWlgKDNdr7n7
PxCXe4EayFswSgGtpH0Sift6Ei2Cm9wHxapnvSdjsDvCoUHfl/RTeQY0iGhW/Tcy
3VAR67A2E9CXmV0lH1f4aHQe/Kgg67oqnhQ6iEk1IxxTgLxIyiDbH43uHCcDIvMl
T2GgZZOD9jwMY8qlV6RQjL0JtIoXbdu6w4abgtG+DFGtE6skinXizvyfjYzm7zwQ
02oParTkDY6H98nOWuiBrGBuIUQfNuhY4bwCZ7SsG6ROEdbHt9NaMAhNj9zO7iIV
hWJawZr4YDO2SOqzFfF7pmIIXrd78NpwnzQ8UA0O2MNnQppsIwfJ7fZIGQ==
-----END CERTIFICATE-----
vmanage:~$
```

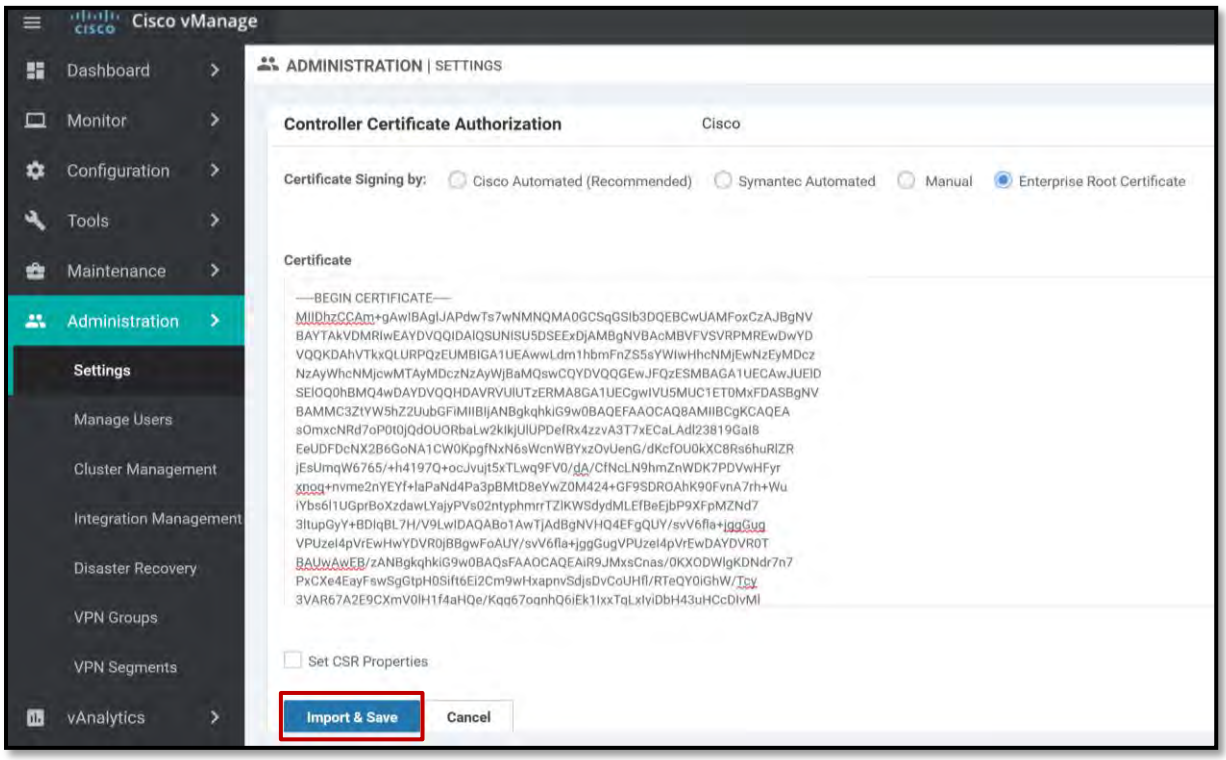
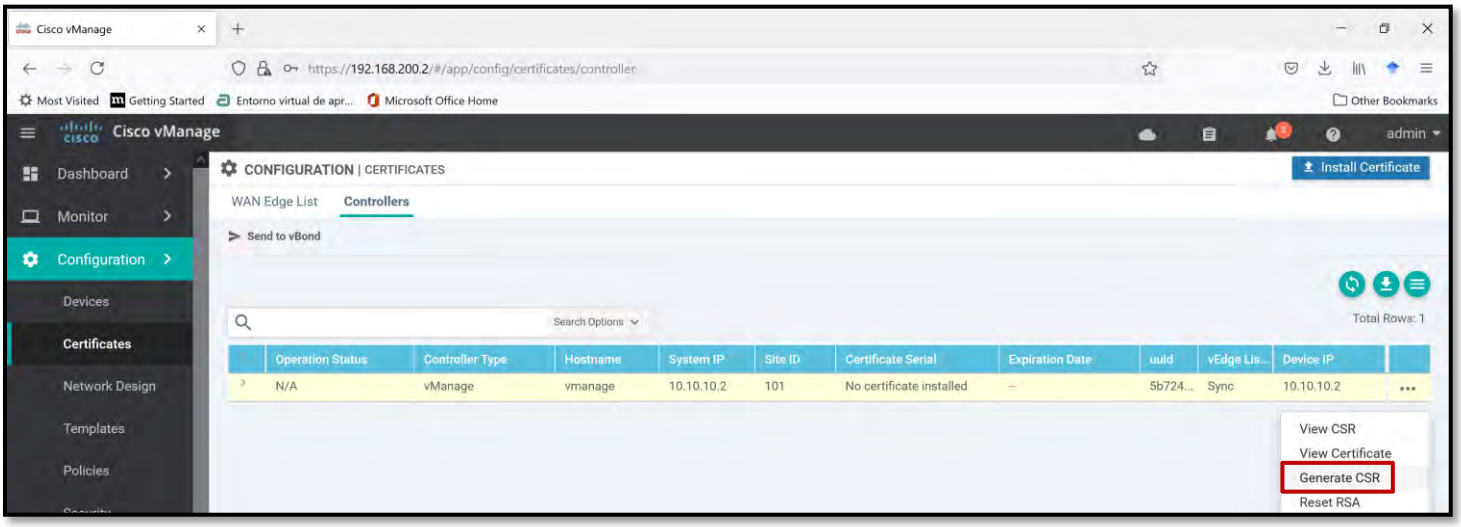


Figura 4-137 Contenido de ROOTCA.pem copiado en vManage - PoC SDWAN
Fuente: *Autor*

Ahora, es necesario que el *vManage* firme el certificado generado, para lo cual hay que ir a *Configuration-Certificates* del menú lateral. Ahí se selecciona *Controllers* y aparecerá el del *vManage*. En ese punto, se debe generar un CSR (*Certificate Signed Request*).




```
vmanage:~$
vmanage:~$ ls
ROOTCA.key  ROOTCA.pem  archive_id_rsa.pub
vmanage:~$ ls
ROOTCA.key  ROOTCA.pem  archive_id_rsa.pub  vmanage_csr
vmanage:~$
```

Figura 4-138 Generación de vmanage_csr en vManage - PoC SDWAN
Fuente: *Autor*

Para crear el archivo *vmanage.crt* de *vmanage_csr* se usará otro comando en *vShell* haciendo referencia al certificado x509[68]:

```
vmanage:~$ openssl x509 -req -in vmanage_csr \
>      -CA ROOTCA.pem -CAkey ROOTCA.key -CAcreateserial \
>      -out vmanage.crt -days 2000 -sha256
Signature ok
subject=/C=US/ST=California/L=San Jose/OU=UNLP-DOC/O=Viptela LLC/CN=vmanage-5b72
4ad2-0e2c-4ebb-b2bb-45ca788c1427-0.viptela.com/emailAddress=support@viptela.com
Getting CA Private Key
vmanage:~$ ls
ROOTCA.key  ROOTCA.srl          vmanage.crt
ROOTCA.pem  archive_id_rsa.pub  vmanage_csr
```

Script 53 Generación de vmanage.crt en vManage - PoC SDWAN
Fuente: *Autor*

El contenido del archivo *vmanage.crt* se debe instalar en *vManage*.

```
vManager
vmanage:~$ cat vmanage.crt
-----BEGIN CERTIFICATE-----
MIIDnjCCAoYCCQCikm1Vyw0JmjANBgkqhkiG9w0BAQsFADBaMQswCQYDVQQGEwJF
QzESMBAGA1UECAwJUElDSElOQ0hBMQ4wDAYDVQQHDAVRVUlUTzERMA8GA1UECgwI
VU5MUC1ET0MxFDASBgNVBAMMC3ZtYW5hZ2UubGFiMB4XDTIxMDcxMjA3NTcwMFoX
```

[68] Certificado ITU x509: https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=14033&lang=es

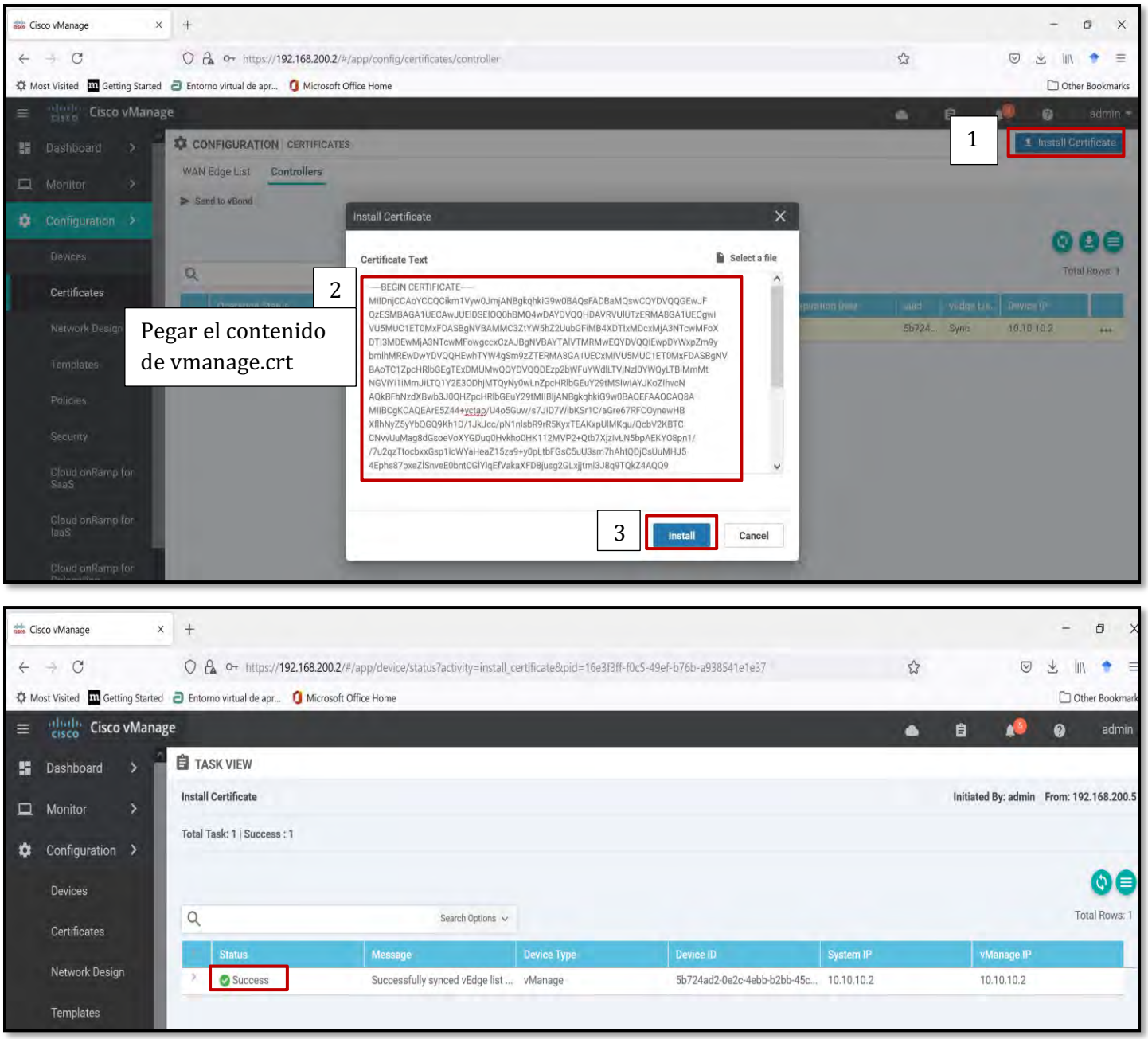


Figura 4-139 Instalación y sincronización de vmanage.crt en vManage (Success) - PoC SDWAN
Fuente: *Autor*

El registro del *vManage* debe ser exitoso, tal como se ve en la *Fig. 4-139*. Al serlo, es necesario incluir el resto de certificados de los demás equipos del Plano de Control (*vBond* y *vSmart*).

Generación y activación de Certificados en *vManage* para *vSmart* y *vBond*

El proceso es similar al certificado de *vManage*, sin embargo, al no crear los certificados directamente en los nodos, sino en el *vManage*, hay que tomar en cuenta otros comandos para agregar el *vSmart* y *vBond*.

Para añadir el resto de "controladores" se debe ingresar a *Configuration-Devices-Add Controllers* en *vManage* y poner sus direcciones IP y credenciales. Se comienza con el *vSmart*.

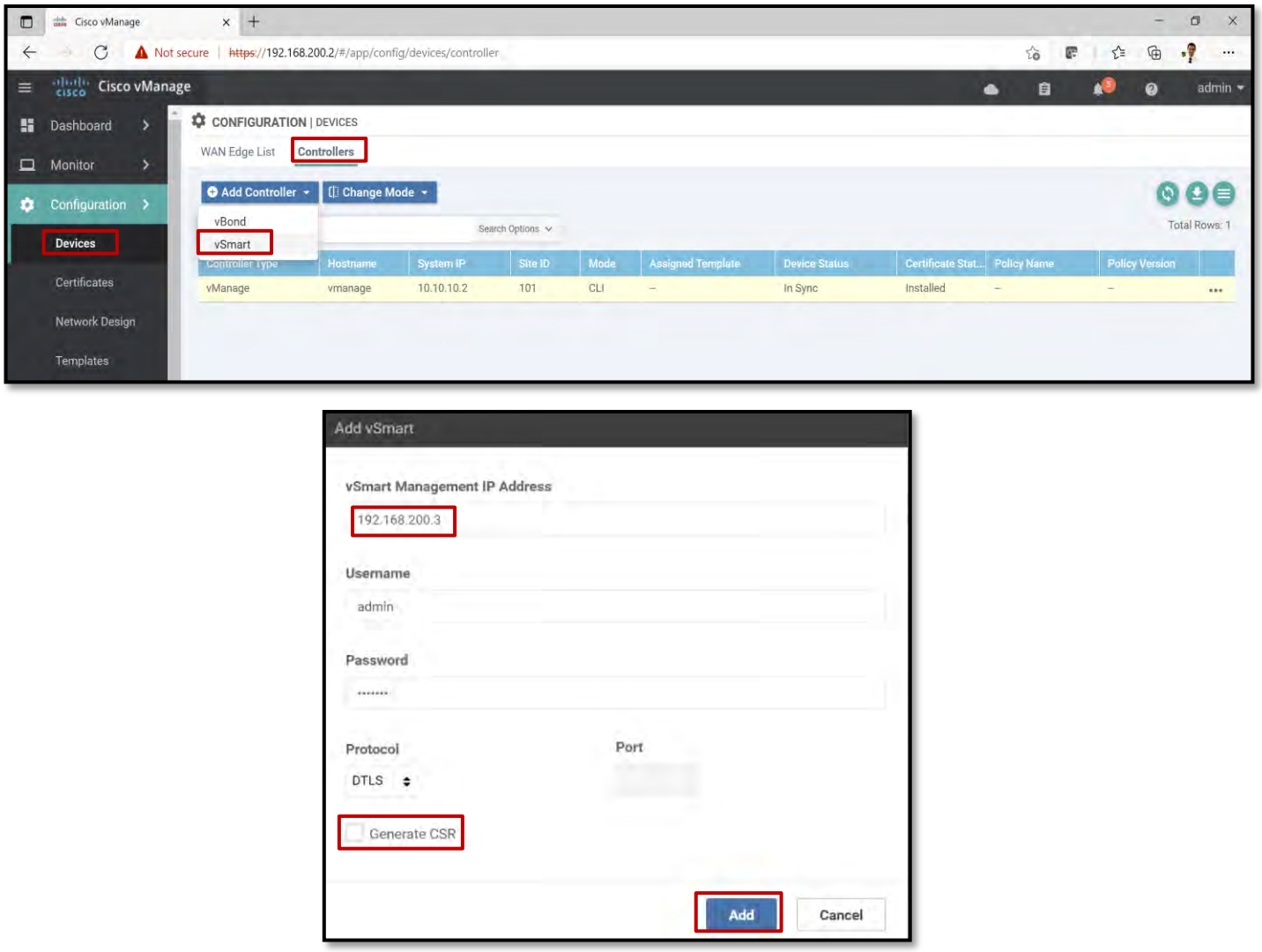


Figura 4-140 Añadir vSmart en vManage - PoC SDWAN
Fuente: *Autor*

Para agregar al *vBond*, se debe modificar la configuración de su VPN0 y que por el momento no acepte encriptación, pues aún no tiene los certificados que le habilitan hacerlo.

```
vbond# config term
Entering configuration mode terminal
vbond(config)# vpn 0
vbond(config-vpn-0)# interface ge0/0
vbond(config-interface-ge0/0)# no tunnel-interface
vbond(config-interface-ge0/0)# commit and-quit
Commit complete.
vbond#
```

Script 54 Eliminar encriptación para ingreso de vBond en vManage - PoC SDWAN
Fuente: *Autor*

Una vez realizado ello, se agrega el *vBond* así como se hizo con el *vSmart.* Al hacerlo, se tendrá el plano de control completo ingresado a *vManage*.

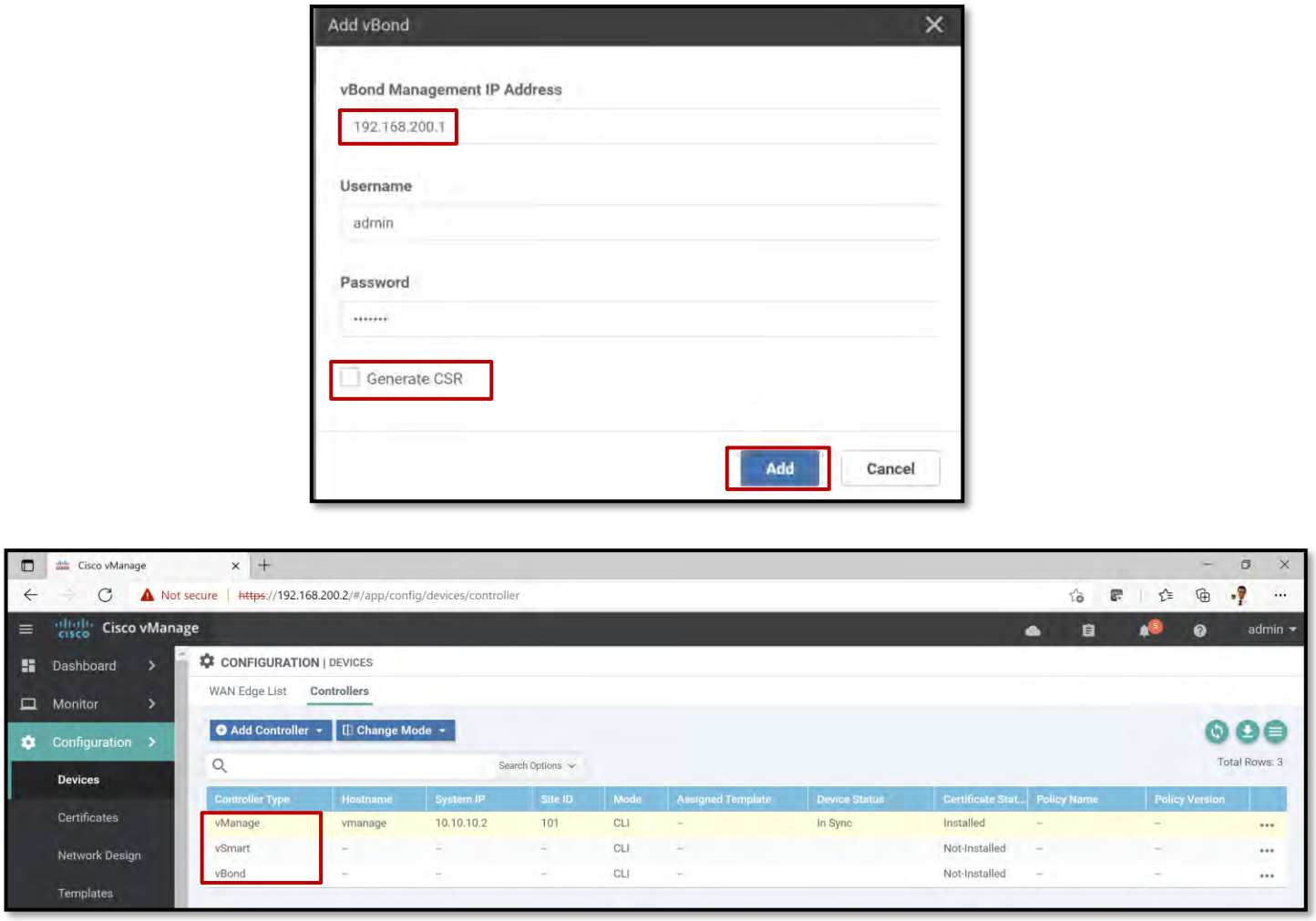


Figura 4-141 Añadir vBond en vManage - PoC SDWAN
Fuente: *Autor*

El próximo paso es crear y firmar los certificados para los *vSmart* y *vBond*. Para ello se ingresa a *Configuration-Certificates-Controllers* y se repiten los pasos de las *Fig. 4-138* y *4-139,* así como el *Script 53,* para cada dispositivo.

Certificado para *vBond*

Se genera la solicitud de certificado para *vBond*, se copia su contenido y se pega en un archivo de nombre ***vbond_csrt*** en *vManage.* Los pasos se muestran en la *Fig. 4-142*.

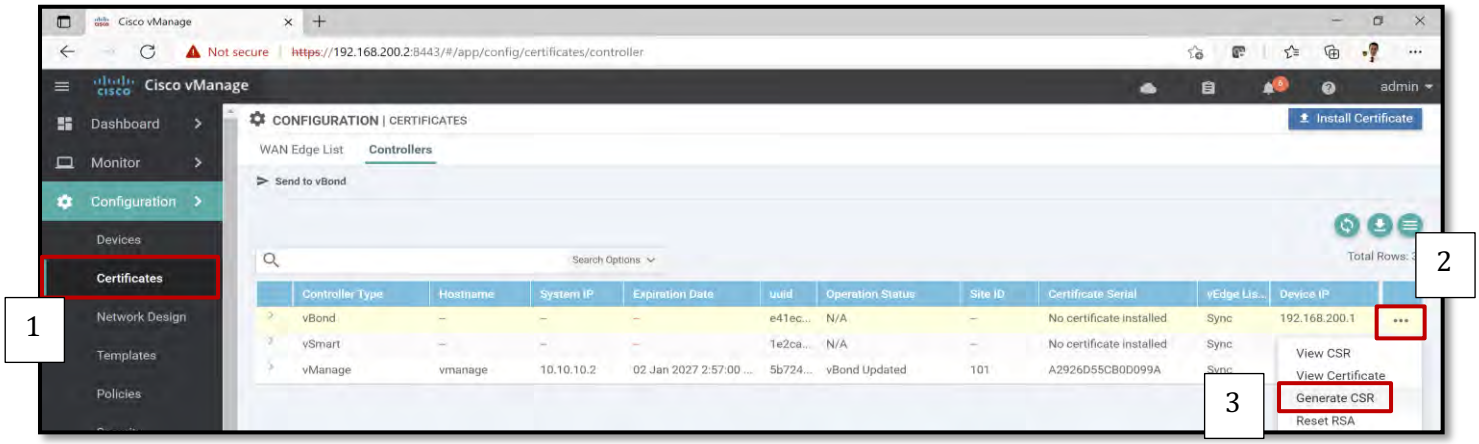


Figura 4-142 Pasos para crear CSR de vBond en vManage - PoC SDWAN
Fuente: *Autor*

Al realizar los seis (6) pasos anteriores, es momento de firmar el certificado (crear archivo *vBond.crt*), para lo cual se usará el siguiente comando:

```
vmanage:~$ openssl x509 -req -in vbond_csr \
>      -CA ROOTCA.pem -CAkey ROOTCA.key -CAcreateserial \
>      -out vbond.crt -days 2000 -sha256
Signature ok
subject=/C=US/ST=California/L=San Jose/OU=UNLP-DOC/O=Viptela LLC/CN=vbond-e41ec1
ed-bb08-49c9-a27e-1da51ec202fa-0.viptela.com/emailAddress=support@viptela.com
Getting CA Private Key
vmanage:~$
vmanage:~$ ls
ROOTCA.key  ROOTCA.srl          vbond.crt  vmanage.crt
ROOTCA.pem  archive_id_rsa.pub  vbond_csr  vmanage_csr
```

Script 55 Creación de vBond.crt en vManage - PoC SDWAN
Fuente: *Autor*

El certificado firmado (*vbond.crt*) es el que se debe colocar en el *vManage* en el sitio de *vBond*:

Copiar contenido de *vbond.crt*

```
vmanage:~$
vmanage:~$ cat vbond.crt
-----BEGIN CERTIFICATE-----
MIID[illegible]DCCAoQCCQCikm1Vyw0JmzANBgkqhkiG9w0BAQsFADBaMQswCQYDVQQGEwJF
[illegible]BAGA1UECAwJUElDSElOQ0hBMQ4wDAYDVQQHDAVRVUlUTzERMA8GA1UECgwI
[illegible]C1ET0MxFDASBgNVBAMMC3ZtYW5hZ2UubGFiMB4XDTIxMDcxNTA1MTEyOVoX
[illegible]DEwNTA1MTEyOVowgcUxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9y
[illegible]REwDwYDVQQHEwhTYW4gSm9zZTERMA8GA1UECxMIVU5MUC1ET0MxFDASBgNV
BAoTC1ZpcHRlbGEgTExDMUEwPwYDVQQDEzh2Ym9uZC1lNDFlYzFlZC1iYjA4LTQ5
YzktYTI3ZS0xZGE1MWVjMjAyZmEtMC52aXB0ZWxhLmNvbTEiMCAGCSqGSIb3DQEJ
```

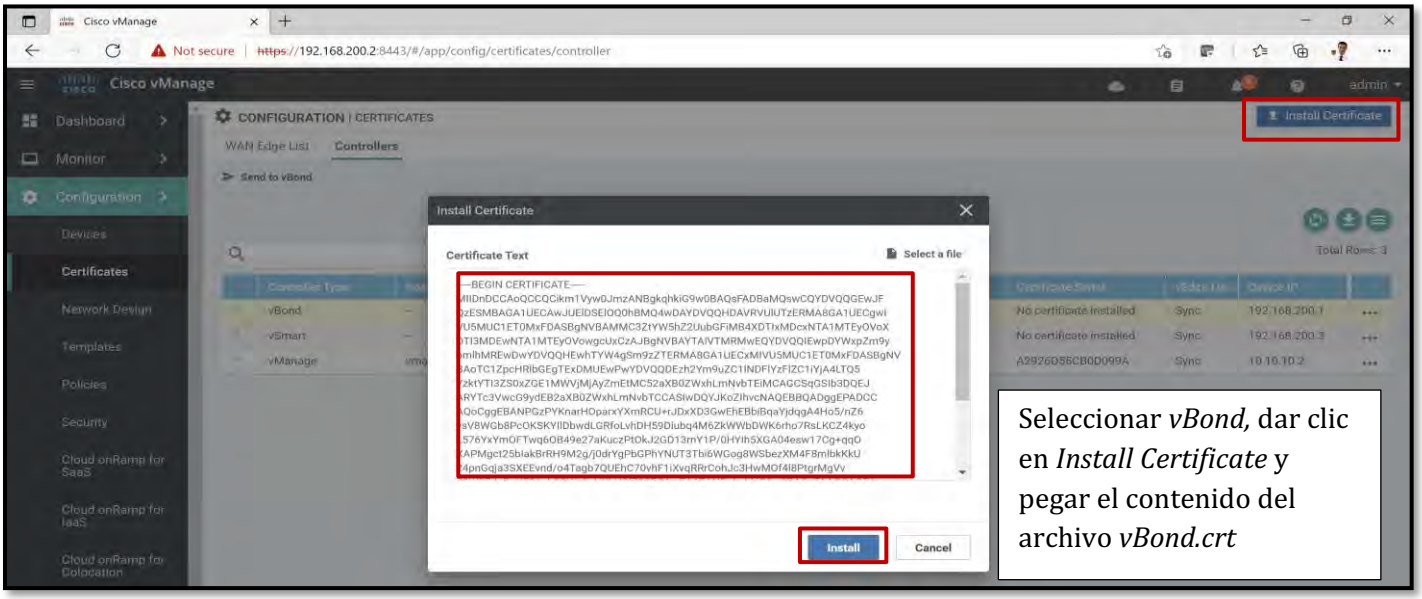


Figura 4-143 Instalar vBond.crt en vManage - PoC SDWAN
Fuente: *Autor*

Realizando este proceso, se habrá instalado el certificado para el *vBond* en *vManage*.

Certificado para *vSmart*

El proceso es el mismo que para el *vBond* (*Fig. 4-140, Fig. 4-141* y *Script 55*), pero cambiando el nombre de los certificados y archivos a *vSmart*.

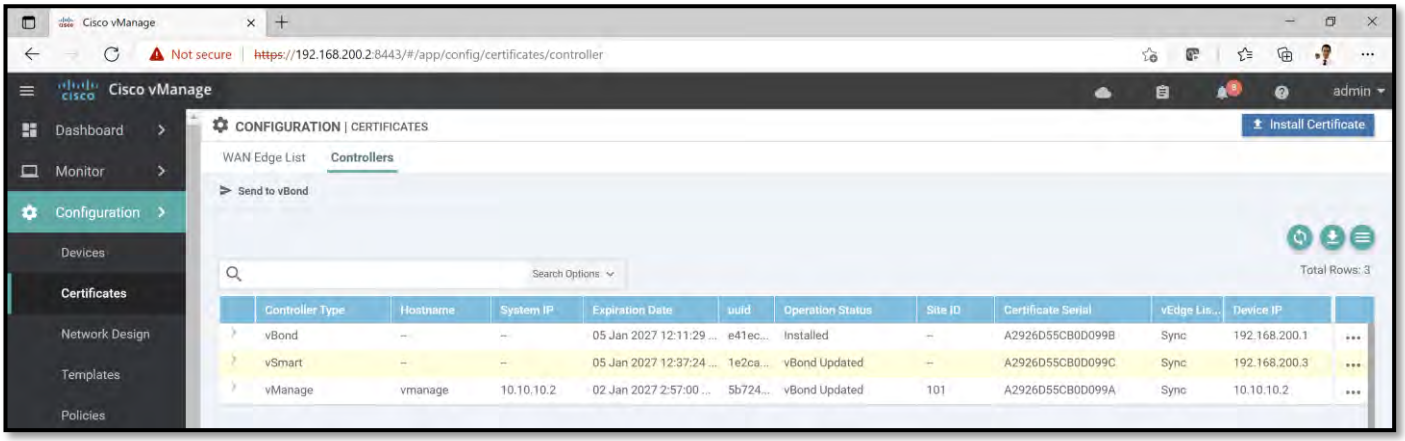


Figura 4-144 Certificados instalados para vBond, vSmart y vManage en vManage - PoC SDWAN
Fuente: *Autor*

Ahora, se debe enviar los certificados instalados al *vBond* y habilitar la comunicación mediante DTLS/IPSec de todo el plano de Control del PoC SDWAN.

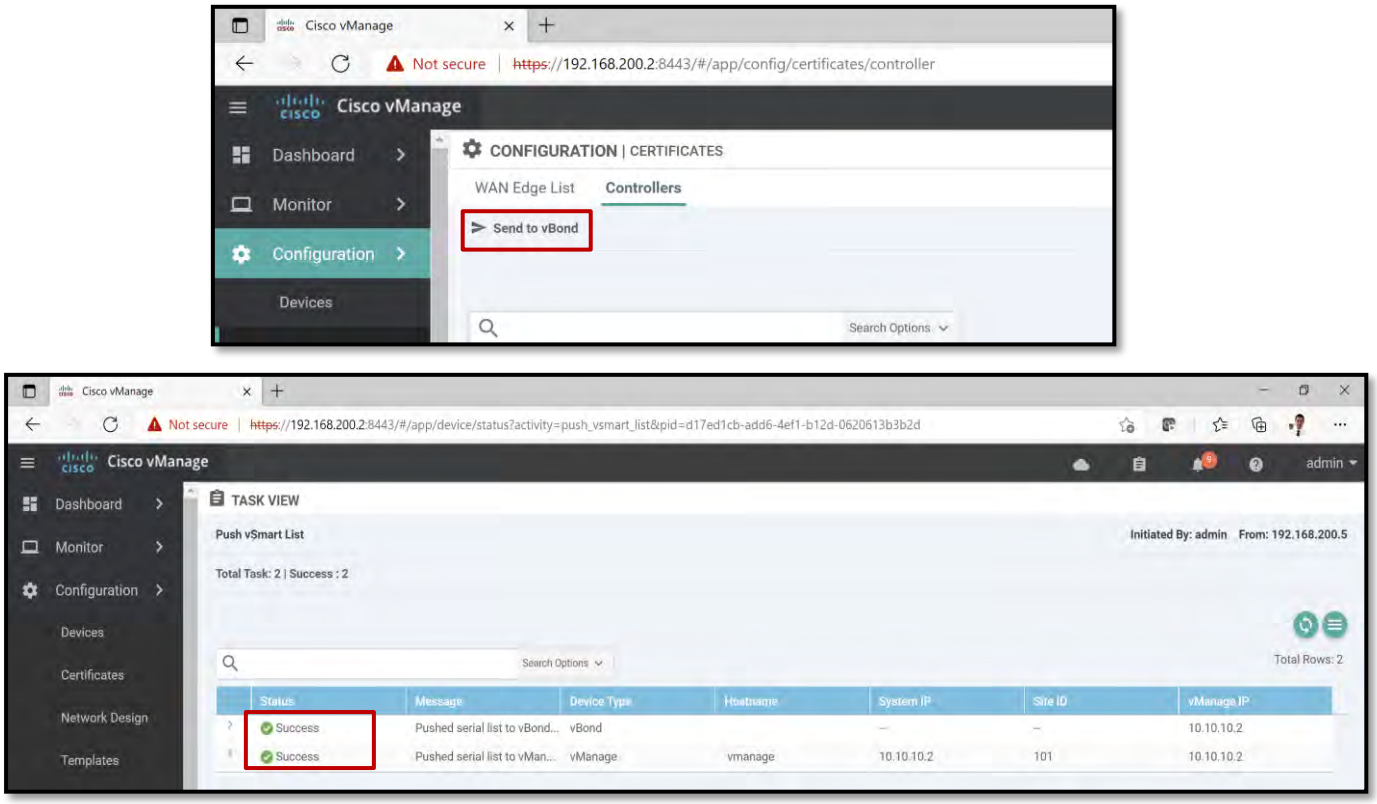


Figura 4-145 Envío exitoso de Certificados a vBond desde vManage - PoC SDWAN
Fuente: *Autor*

Con el fin de habilitar DTLS para los túneles de comunicación en el plano de control, en sus líneas de comando se debe habilitar de la siguiente manera:

```
vsmart# config t
Entering configuration mode terminal
vsmart(config)# vpn 0
vsmart(config-vpn-0)# interface eth0
vsmart(config-interface-eth0)# tunnel-interface
vsmart(config-tunnel-interface)# commit
Commit complete.
vsmart(config-tunnel-interface)#
```

```
vmanage# config t
Entering configuration mode terminal
vmanage(config)# vpn 0
vmanage(config-vpn-0)# interface eth0
vmanage(config-interface-eth0)# tunnel-interface
vmanage(config-tunnel-interface)# commit
Commit complete.
vmanage(config-tunnel-interface)#
```

```
vbond#
vbond# config t
Entering configuration mode terminal
vbond(config)# vpn 0
vbond(config-vpn-0)# interface ge0/0
vbond(config-interface-ge0/0)# tunnel-interface
vbond(config-tunnel-interface)# encapsulation ipsec
vbond(config-tunnel-interface)# commit
Commit complete.
vbond(config-tunnel-interface)#
```

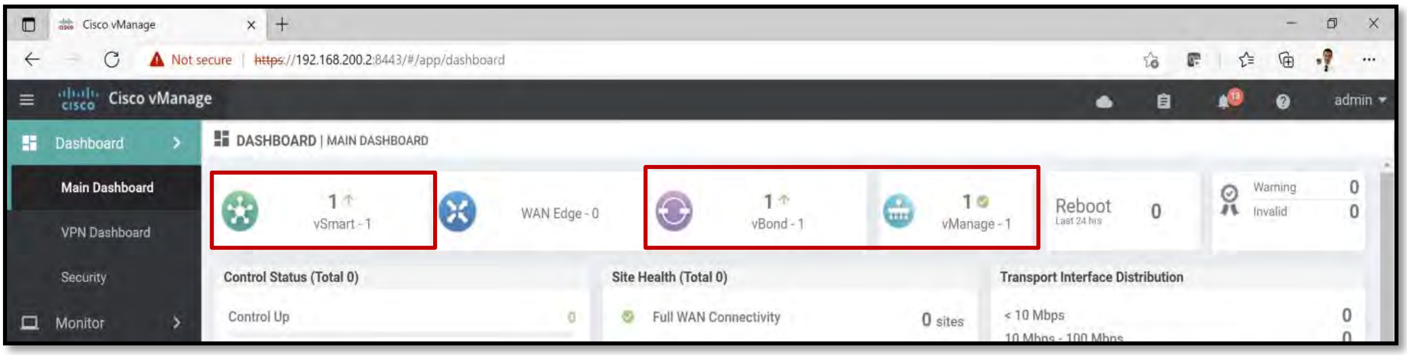
```
vsmart#
vsmart# show control connections
                                                                            PEER                                PEER
      PEER    PEER                SITE        DOMAIN PEER                   PRIV  PEER                          PUB
INDEX TYPE    PROT SYSTEM IP      ID          ID     PRIVATE IP             PORT  PUBLIC IP                     PORT  REMOTE COLOR     STATE UPTIME
---------------------------------------------------------------------------------------------------------------------------------------------------------
0     vbond   dtls 0.0.0.0        0           0      192.168.200.1          12346 192.168.200.1                 12346 default          up    0:00:13:33
0     vmanage dtls 10.10.10.2     101         0      192.168.200.2          12346 192.168.200.2                 12346 default          up    0:00:11:32
1     vbond   dtls 0.0.0.0        0           0      192.168.200.1          12346 192.168.200.1                 12346 default          up    0:00:13:32

vsmart#
```

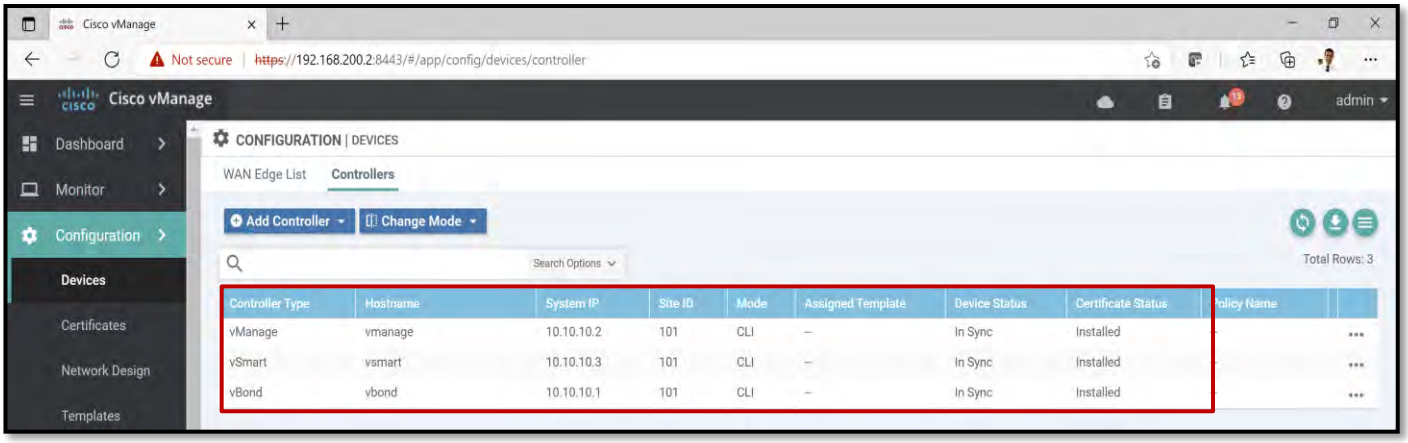
```
vmanage# show control connections
                                PEER                                                  PEER                          PEER
      PEER    PEER              CONFIGURED      SITE     DOMAIN PEER                  PRIV  PEER                    PUB
INDEX TYPE    PROT SYSTEM IP    SYSTEM IP       ID       ID     PRIVATE IP            PORT  PUBLIC IP               PORT  ORGANIZATION         REMOTE COLOR     STATE UPTIME
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
0     vsmart  dtls 10.10.10.3   10.10.10.3      101      1      192.168.200.3         12346 192.168.200.3           12346 UNLP-DOC             default          up    0:00:10:27
0     vbond   dtls 10.10.10.1   10.10.10.1      0        0      192.168.200.1         12346 192.168.200.1           12346 UNLP-DOC             default          up    0:00:10:27
1     vbond   dtls 0.0.0.0      -               0        0      192.168.200.1         12346 192.168.200.1           12346 UNLP-DOC             default          up    0:00:10:26
2     vbond   dtls 0.0.0.0      -               0        0      192.168.200.1         12346 192.168.200.1           12346 UNLP-DOC             default          up    0:00:10:26
3     vbond   dtls 0.0.0.0      -               0        0      192.168.200.1         12346 192.168.200.1           12346 UNLP-DOC             default          up    0:00:10:28

vmanage#
```

```
vbond#
vbond# show orchestrator connections
                                                                       PEER                     PEER
         PEER     PEER      PEER             SITE     DOMAIN  PEER            PRIVATE  PEER            PUBLIC                                  ORGANIZATION
INSTANCE TYPE     PROTOCOL  SYSTEM IP        ID       ID      PRIVATE IP      PORT     PUBLIC IP       PORT    REMOTE COLOR    STATE           NAME                      UPTIME
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
0        vsmart   dtls      10.10.10.3       101      1       192.168.200.3   12346    192.168.200.3   12346   default         up              UNLP-DOC                  0:00:13:14
0        vsmart   dtls      10.10.10.3       101      1       192.168.200.3   12446    192.168.200.3   12446   default         up              UNLP-DOC                  0:00:13:13
0        vmanage  dtls      10.10.10.2       101      0       192.168.200.2   12346    192.168.200.2   12346   default         up              UNLP-DOC                  0:00:11:13
0        vmanage  dtls      10.10.10.2       101      0       192.168.200.2   12446    192.168.200.2   12446   default         up              UNLP-DOC                  0:00:11:12
0        vmanage  dtls      10.10.10.2       101      0       192.168.200.2   12546    192.168.200.2   12546   default         up              UNLP-DOC                  0:00:11:13
0        vmanage  dtls      10.10.10.2       101      0       192.168.200.2   12646    192.168.200.2   12646   default         up              UNLP-DOC                  0:00:11:13

vbond#
```

Script 56 Configuración de Encriptación DTLS para TLOC en vSmart, vManage y vBond - PoC SDWAN
Fuente: *Autor*

En *Main Dashboard* y en *Configuration-Devices*, se confirmará la existencia de *vBond, vSmart* y *vManage* totalmente funcionales:






```
vmanage#
vmanage# show control local-properties
personality                     vmanage
sp-organization-name            UNLP-DOC
organization-name               UNLP-DOC
root-ca-chain-status            Installed

certificate-status              Installed
certificate-validity            Valid
certificate-not-valid-before    Jul 12 07:57:00 2021 GMT
certificate-not-valid-after     Jan 02 07:57:00 2027 GMT

dns-name                        192.168.200.1
site-id                         101
domain-id                       0
protocol                        dtls
tls-port                        23456
system-ip                       10.10.10.2
chassis-num/unique-id           5b724ad2-0e2c-4ebb-b2bb-45ca788c1427
serial-num                      A2926D55CB0D099A
```

```
vmanage# show certificate installed

Server certificate
------------------

Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            a2:92:6d:55:cb:0d:09:9a
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=EC, ST=PICHINCHA, L=QUITO, O=UNLP-DOC, CN=vmanage.lab
        Validity
            Not Before: Jul 12 07:57:00 2021 GMT
            Not After : Jan  2 07:57:00 2027 GMT
        Subject: C=US, ST=California, L=San Jose, OU=UNLP-DOC, O=Viptela LLC, CN
=vmanage-5b724ad2-0e2c-4ebb-b2bb-45ca788c1427-0.viptela.com/emailAddress=support
@viptela.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
```

```
vmanage# show interface description

                                                IF      IF      IF
                    AF                          ADMIN   OPER    TRACKER
VPN   INTERFACE     TYPE   IP ADDRESS           STATUS  STATUS  STATUS    DESC
-------------------------------------------------------------------------------
0     eth0          ipv4   192.168.200.2/24     Up      Up      -         -
0     eth1          ipv4   -                    Down    Down    -         -
0     system        ipv4   10.10.10.2/32        Up      Up      -         -
```

```
vbond# show orchestrator summary
orchestrator summary 0
 vmanage_counts          4
 vsmart_counts           2
 vedge_counts            0
 protocol                dtls
 listening_ip            0.0.0.0
 listening_ipv6          ::
 listening_port          12346
 valid_controller_counts 2
```

Figura 4-146 vManage, vBond y vSmart funcionales y con Certificado instalado - PoC SDWAN
Fuente: *Autor*

Al analizar la *Fig. 4-146*, nos indica que **no existe ningún *vEdge*** hasta el momento, por lo que se deben registrar también para comunicar los sitios empresariales.

Encendemos al *vEdge* y entramos a su CLI para configurarlo de la siguiente manera (se debe cambiar su contraseña inicial, para el caso del PoC a **UNLPDOC**):

```
vedge# config t
Entering configuration mode terminal
vedge(config)# system
vedge(config-system)# host-name vEdge
vedge(config-system)# system-ip 10.10.10.4
vedge(config-system)# site-id 101
vedge(config-system)# admin-tech-on-failure
vedge(config-system)# organization-name UNLP-DOC
vedge(config-system)# vbond 192.168.200.1
vedge(config-system)#
vedge(config-system)# vpn 0
vedge(config-vpn-0)# interface ge0/0
vedge(config-interface-ge0/0)# ip address 192.168.200.4/24
vedge(config-interface-ge0/0)# no shut
vedge(config-interface-ge0/0)# commit
Commit complete.
```

Script 57 Configuración básica de vEdge - PoC SDWAN
Fuente: *Autor*

Tal como ocurrió con los equipos del Plano de Control del PoC SDWAN, los equipos del lado del cliente o Plano de Datos, *vEdges/cEdges*, también requieren de un registro por Certificados.

Con este comando se puede ver que este *vEdge,* en un inicio no posee ningún certificado instalado:

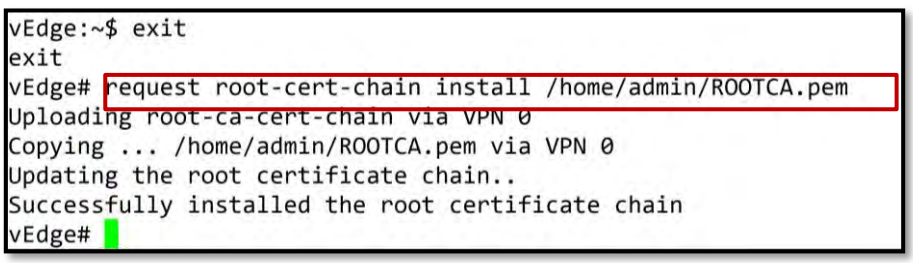
```
vEdge# show certificate serial
Certificate not yet installed ... giving up.
Chassis number: 7680a605-85c4-441a-a6c2-fc7cb5db4cc1 serial number:
vEdge#
```

Figura 4-147 vEdge sin Certificado instalado - PoC SDWAN
Fuente: *Autor*

Para crear, firmar e instalar un certificado válido en *vEdge*, se requiere del ROOTCA.pem que se encuentra en *vManage*: copiarlo y pegarlo en el *vEdge*:

Figura 4-148 Proceso de copia de ROOTCA.pem de vManage a vEdge - PoC SDWAN
Fuente: *Autor*

Con ROOTCA.pem en *vEdge*, se debe instalar dicho certificado raíz:



Script 58 Instalación de ROOTCA.pem en vEdge - PoC SDWAN
Fuente: *Autor*

Activación de *vEdges* en *vManage*

Para activar al *vEdge* se requiere de un número de *Token* y número de *Chasis*. En este PoC al ser un *vEdge Cloud*, realmente el dispositivo no cuenta con un número de chasis, pero se lo puede obtener desde *vManage* siempre que se tenga de un ***Smart Account*** asociado para sincronizar las bases de equipos de prueba y ejecutar un proceso llamado **PnP (*Plug-and-Play*)**:

Cuenta *Smart Account* de Cisco

Una cuenta de *Smart Account* es fundamental para agregar equipos virtuales y que el *vManage* tenga un conjunto de números de chasis y *tokens* listos para ser usados en un PoC.

El proceso para tener listo el *Smart Account* empieza en Cisco Software Central[69], plataforma en la que se puede descargar y gestionar *software* oficial de Cisco Systems.

La cuenta *Smart Account* debe estar habilitada y relacionada con un Cisco ID legítimo que soporte descargas del entorno SDWAN, ya sea por ser cliente, *partner* o empleado de Cisco.

Al entrar a administrar el *Smart Account* en Cisco Software Central, se creará una *Virtual Account* relacionando al PoC de forma única e inequívoca, tal como se muestra en la *Fig. 4-149*. En esta cuenta se genera el *Wan Edge List*.

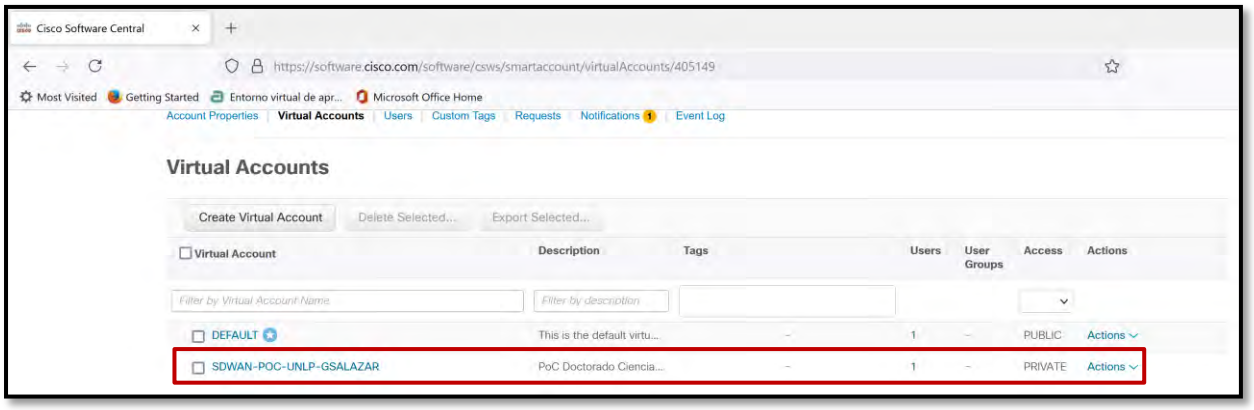
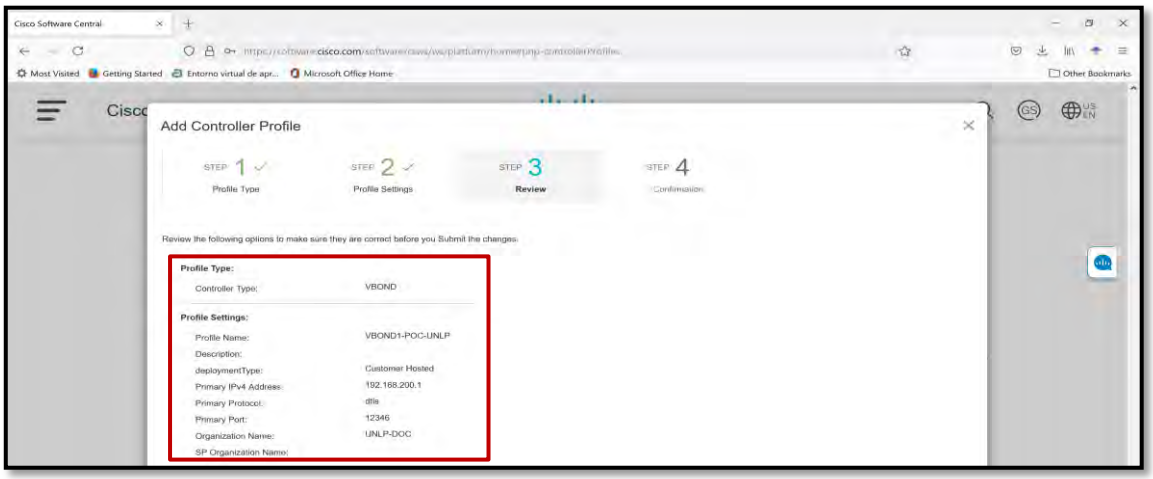


Figura 4-149 Virtual Account en Smart Account de Cisco Software Central - PoC SDWAN
Fuente: *Autor*

Luego, en la sección *Plug and Play Connect,* parte del proceso *Zero-Touch Deployment*, se añadirá el perfil del Controlador *vBond* identificando el **nombre de la Organización**, el cual debe ser único en todo el ecosistema Cisco (UNLP-DOC para el caso de este PoC) y se definirán la cantidad de *vEdges* y así generar el archivo con la lista de equipos.



[69] Cisco Software Central: https://software.cisco.com/#

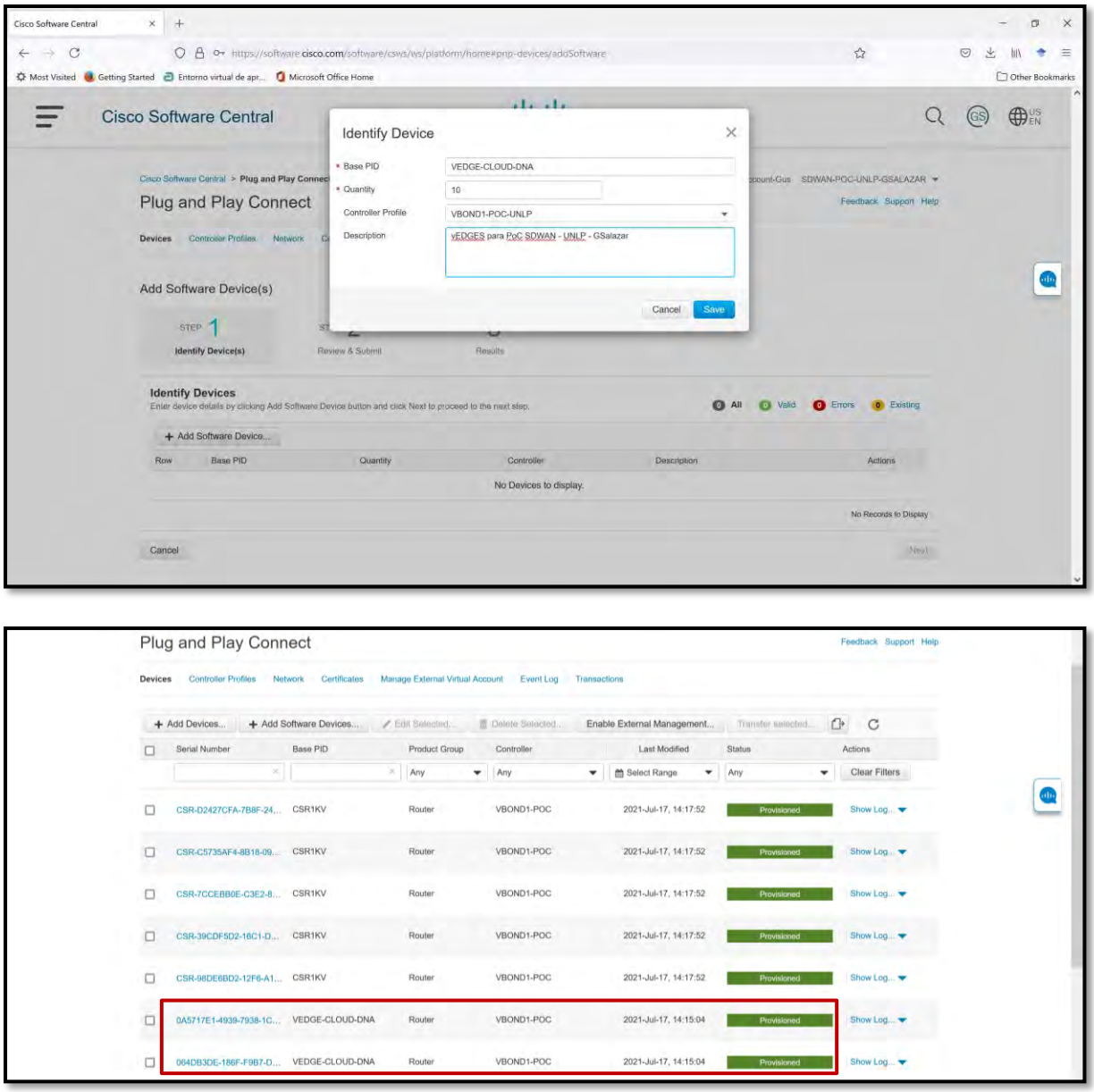
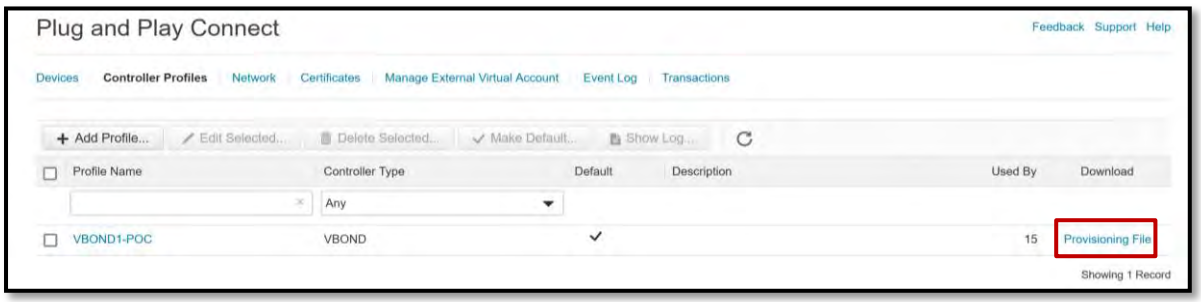


Figura 4-150 Aprovisionamiento de vEdges en Smart Account – Plug and Play Connect (PnP) - PoC SDWAN
Fuente: *Autor*

Si todo es correcto, será factible la descarga del archivo *Viptela* desde el perfil de controlador para incluirlo en el *vManage.* El proceso de La *Fig. 4-151* muestra dicha obtención del ***serialFile.viptela***.



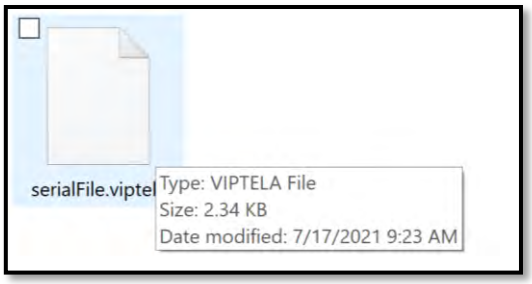
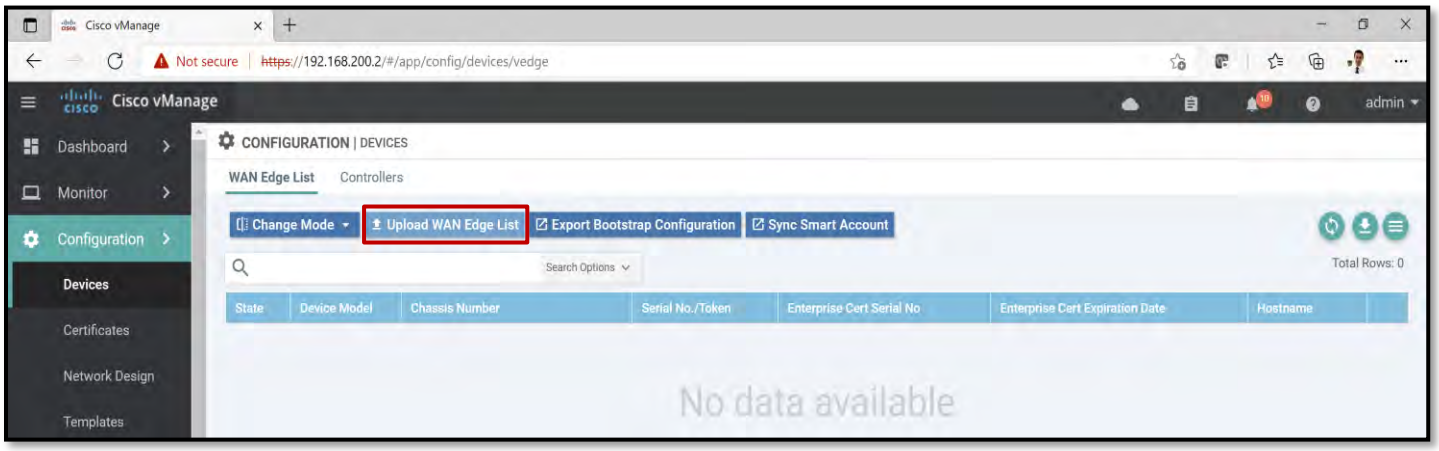


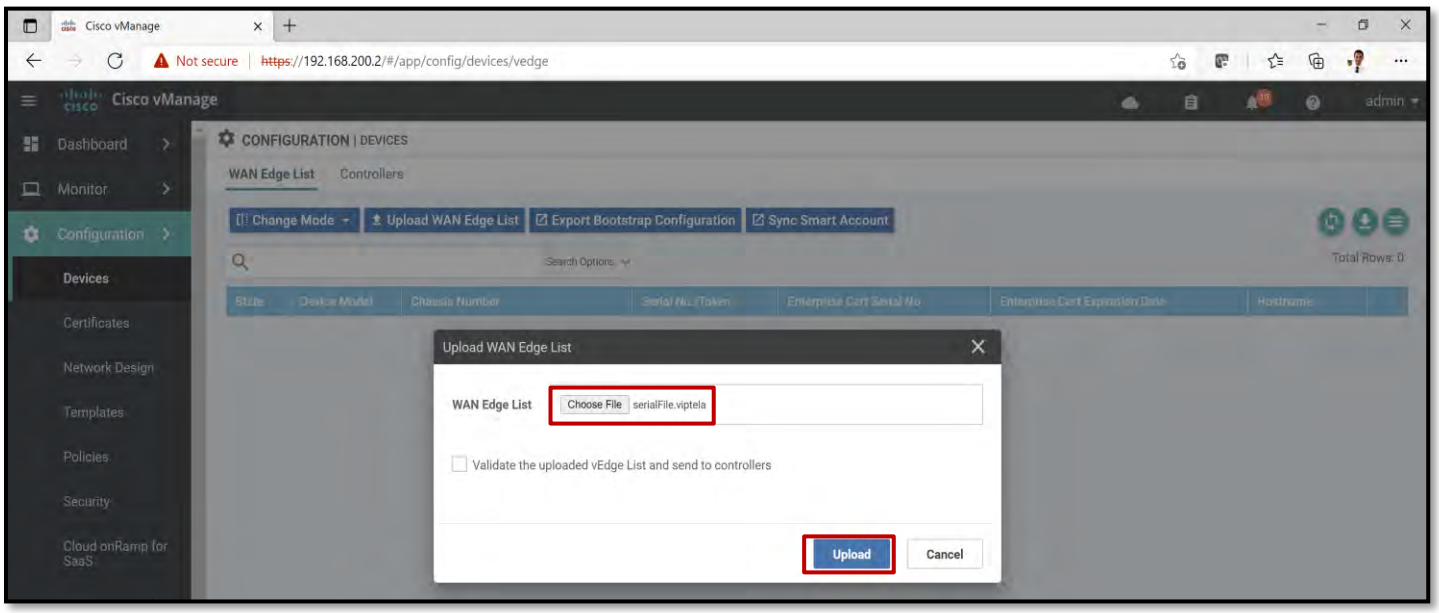
Figura 4-151 Obtención serialFile.viptela – Plug and Play Connect (PnP) - PoC SDWAN
Fuente: *Autor*

Es importante mencionar que no todo el proceso de obtención del *serialFile.viptela* fue descrito en la presente tesis, ya que existen credenciales que no pueden ser entregadas al público.

Ya con el archivo, ingrese al *vManage* y cárguelo de la siguiente manera:

Ingrese a *Configuration-Devices-WAN Edge List*. En ese lugar, dé clic en ***Upload WAN Edge List.*** **No hay que olvidar que para que funcione importar la lista de equipos, el *Organization Name* y la dir. IP de *vBond* debe coincidir** tanto del archivo Viptela como del PoC.





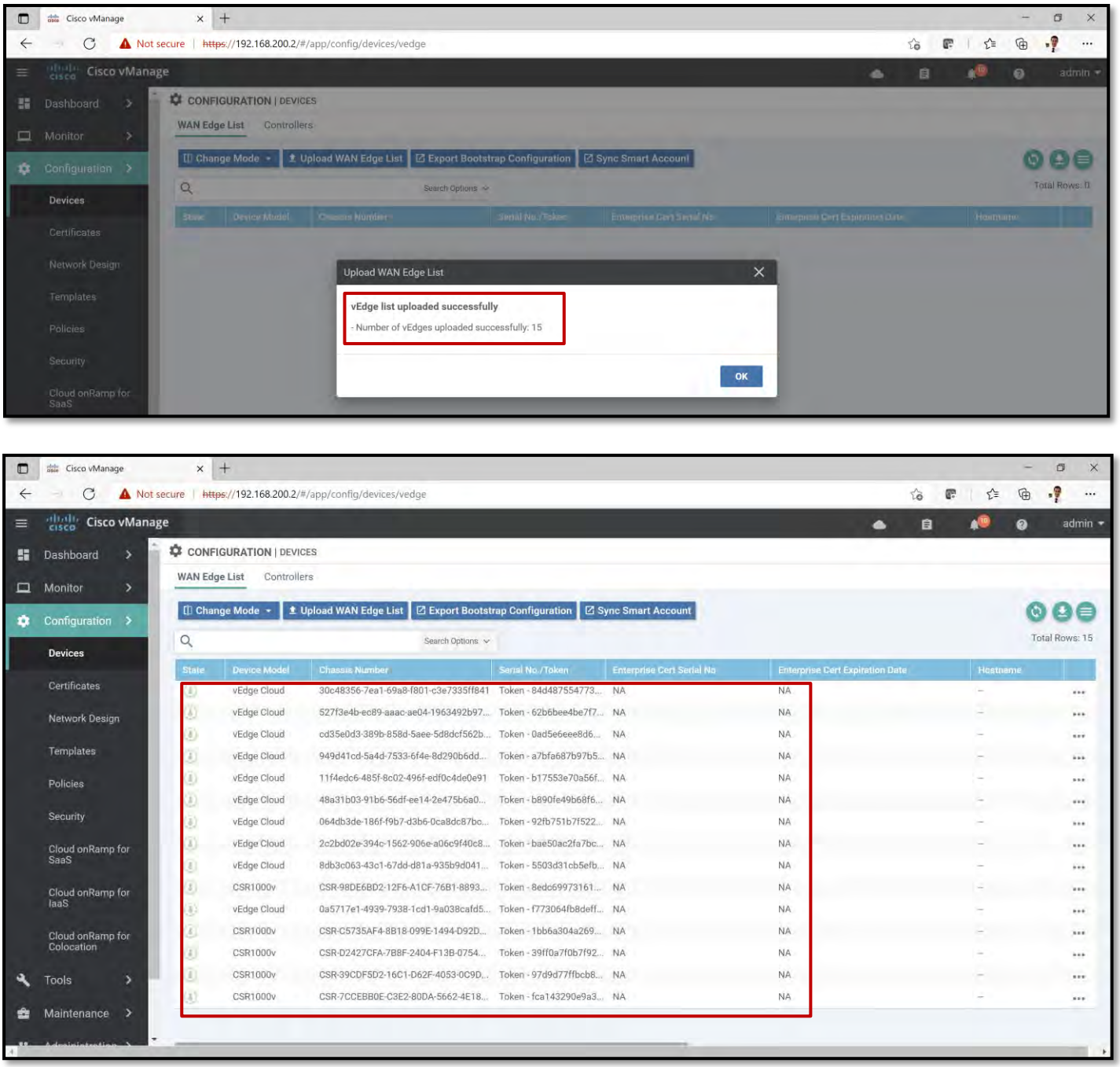


Figura 4-152 Carga de vEdge Clouds a vManage- PoC SDWAN
Fuente: *Autor*

Con el proceso mostrado en la *Fig. 4-152*, se cargó quince (15) *WAN Edges*, entre *vEdge-Cloud-DNA* y *csr1000v*.

Para vincular uno de los *vEdgeClouds* del listado anterior al equipo del PoC SDWAN, se selecciona aleatoriamente uno, en el caso de esta prueba de concepto, se seleccionó el primer *vEdgeCloud* del listado en *vManage*. Una vez seleccionado, se genera su configuración *Bootstrap* y de ahí se obtendrá el número de chasis y *token*. La *Fig. 4-153* muestra los pasos para generar el *Bootstrap Configuration* del *vEdgeCloud* seleccionado.

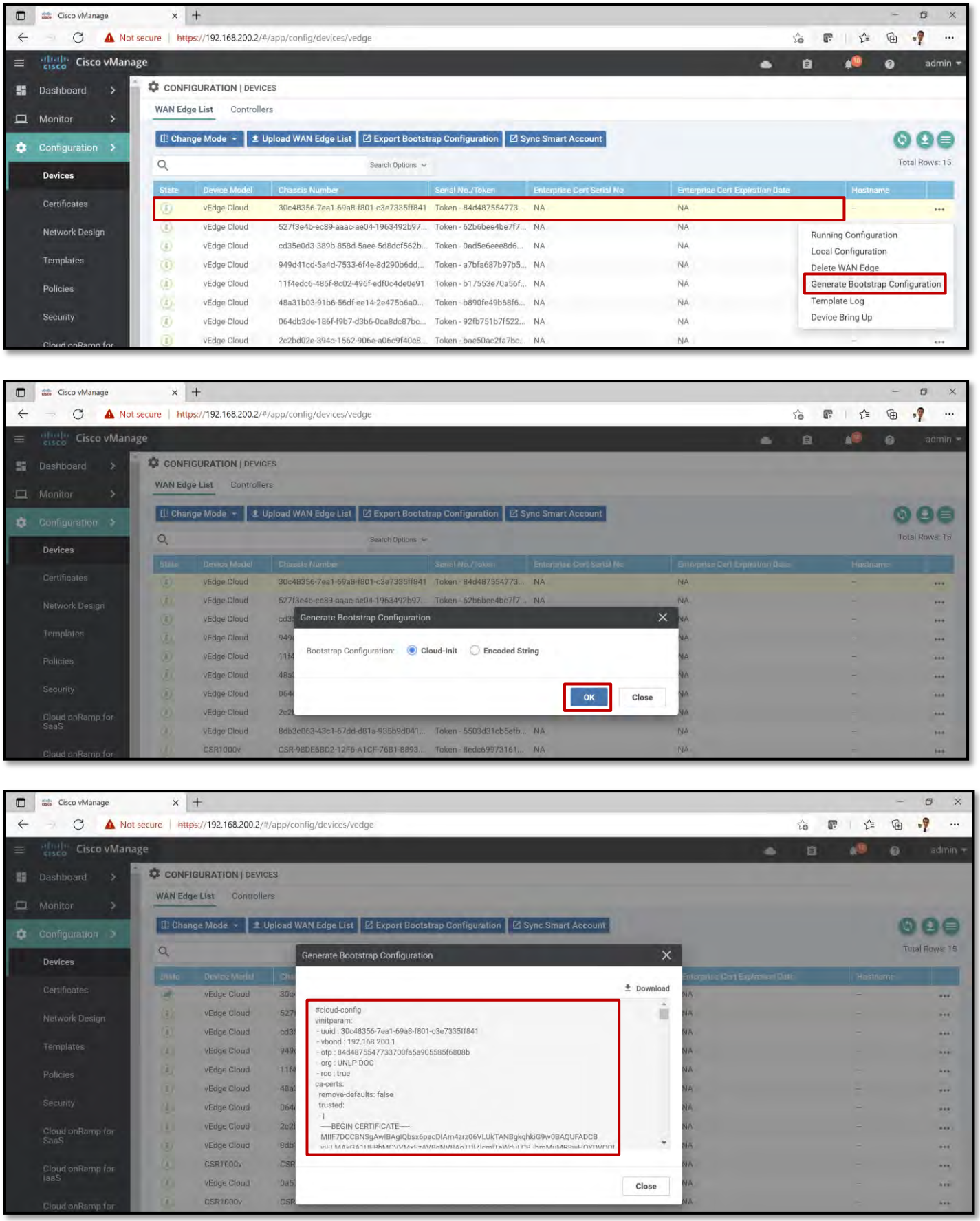


Figura 4-153 Generación de Bootstrap Configuration de vEdge Cloud en vManage- PoC SDWAN
Fuente: *Autor*

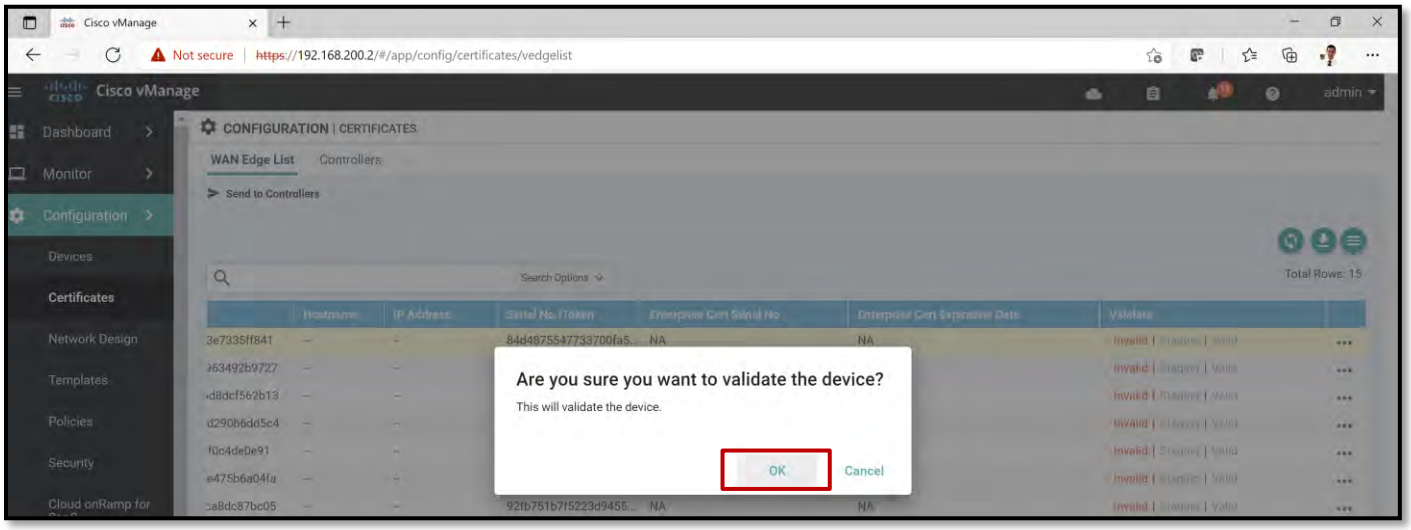
La última imagen de la *Fig. 4-153* muestra el **Número de Chasis (uuid)** y el ***Token* (otp)**, registros que se utilizarán vía comando para el registro del *vEdge* de este PoC.

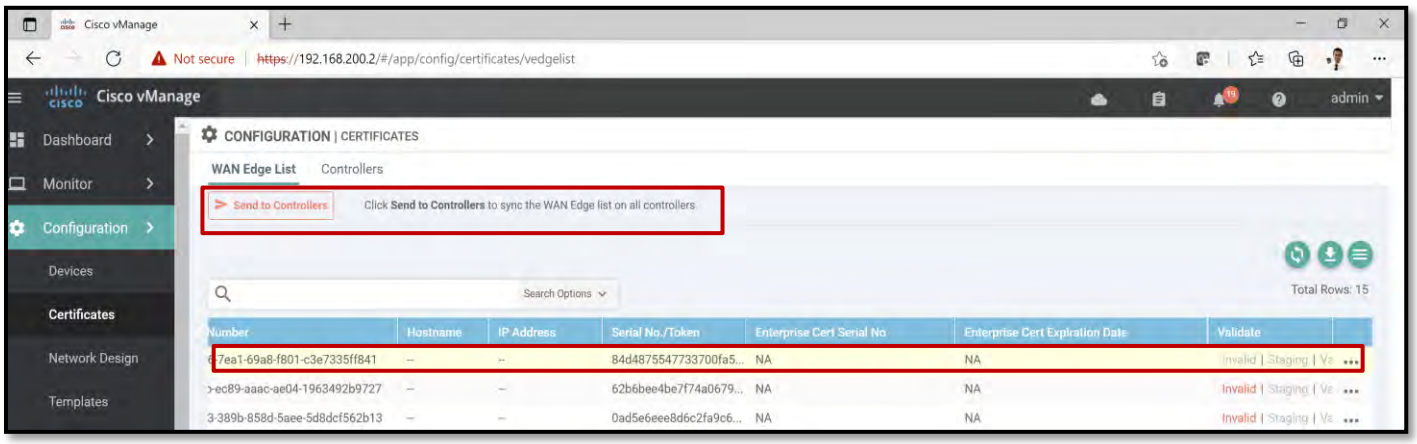
Se debe activar el túnel IPSec en *vEdge* para establecer la encapsulación TLOC correspondiente previo a la vinculación con *vManage* y exista una comunicación segura entre *vEdges* (sedes empresariales).

```
vEdge# config t
Entering configuration mode terminal
vEdge(config)# vpn 0
vEdge(config-vpn-0)# interface ge0/0
vEdge(config-interface-ge0/0)# tunnel-interface
vEdge(config-tunnel-interface)# encapsulation ipsec
vEdge(config-tunnel-interface)#
vEdge(config-tunnel-interface)# commit
```

Script 59 Generación de túnel IPSec-TLOC en vEdge - PoC SDWAN
Fuente: *Autor*

De igual forma, el certificado del equipo *vEdge* seleccionado y generado por el archivo *serialFile.viptela* debe validarse y enviarse a los controladores desde *Configuration-Certificates-WAN Edge List*. Al terminar el proceso, en *Configuration-Devices-WAN Edge List* se verá el Número de serial destinado al *vEdge* de este PoC. Las siguientes figuras muestran ello:





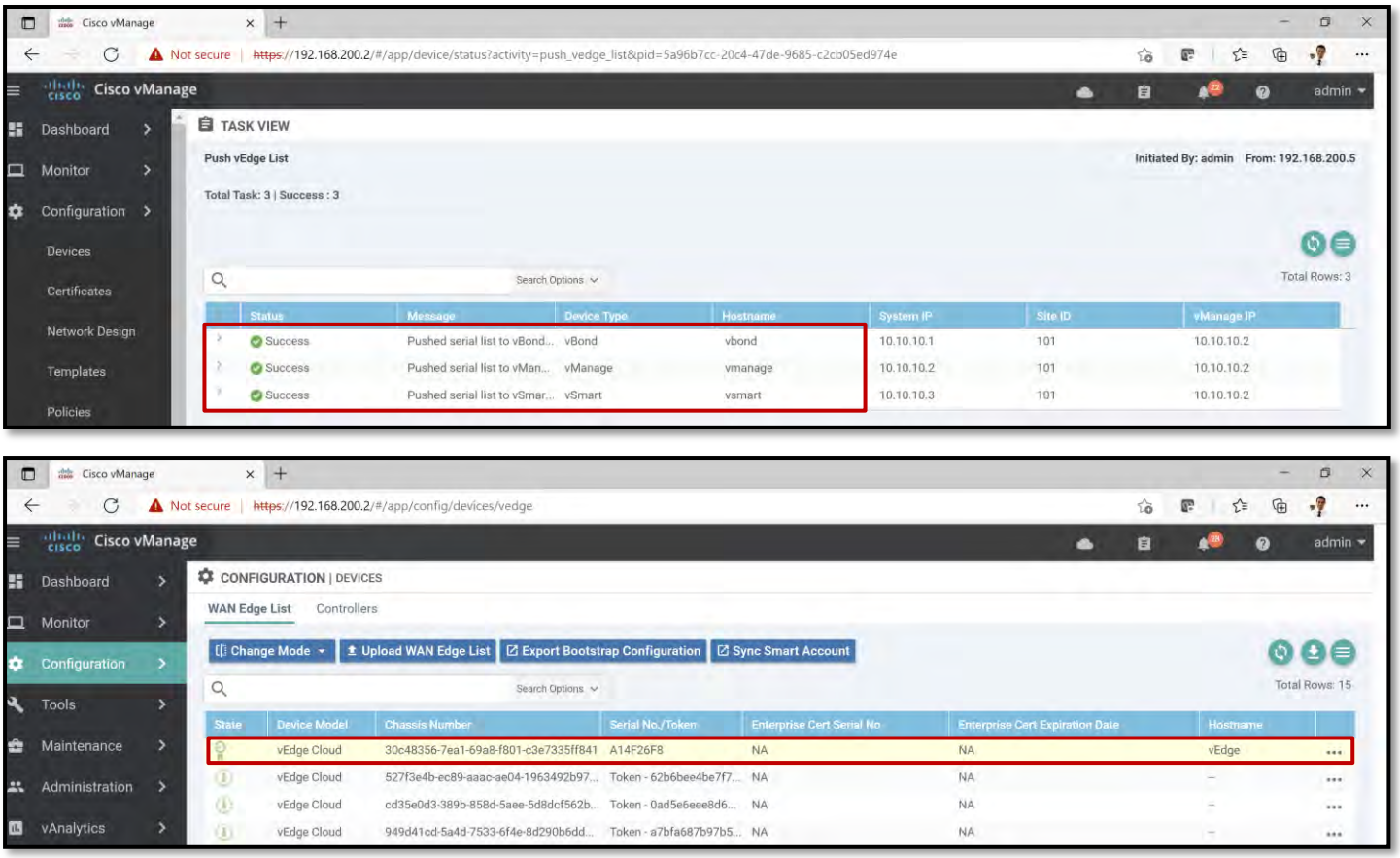


Figura 4-154 Proceso de activación de licencia y envío a Controladores de vEdge Cloud en vManage- PoC SDWAN
Fuente: *Autor*

En el *Script 58* se instaló el ROOTCA.pem en el *vEdge*, ahora, relacionaremos el *uuid* y *otp* de la *Fig. 4-151* a este equipo:

```
vEdge#
vEdge# request vedge-cloud activate chassis-number 30c48356-7ea1-69a8-f801-c3e73
35ff841 token 84d4875547733700fa5a905585f6808b
vEdge#
```

Script 60 Vinculación de número de chasis/Token para vEdge - PoC SDWAN
Fuente: *Autor*

En ese momento, el Plano de Datos (*vEdges*) está bajo el mando del Plano de Control (*vManage, vSmart* y *vBond*).

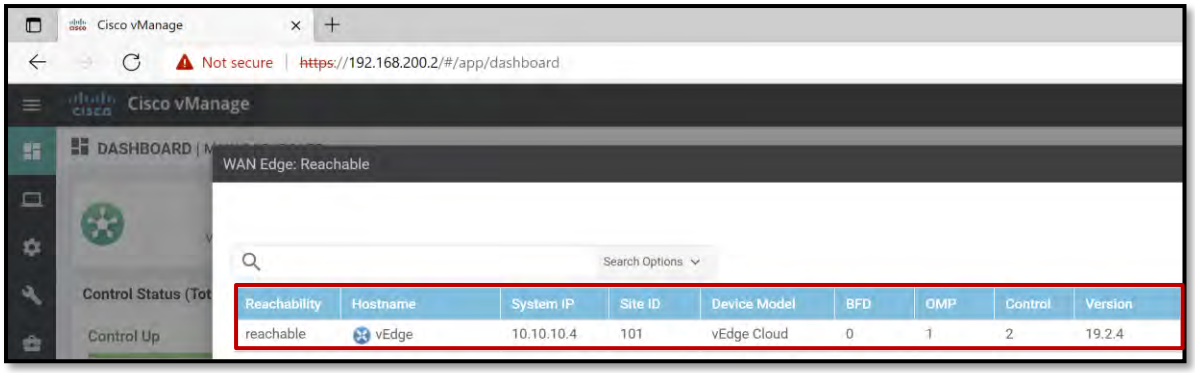


Figura 4-155 Conectividad exitosa de vEdge Cloud a vManage- PoC SDWAN
Fuente: *Autor*

4.5.5 Capacidad de Monitoreo en PoC de SDWAN Viptela

Durante las secciones anteriores se pudo evidenciar el paso a paso, así como verificar la factibilidad de uso de la solución SDWAN Viptela, sin embargo, una vez establecido el Plano de Control y de Datos, una característica fundamental de SDWAN es su capacidad de monitoreo de infraestructura (*insights*).

Las siguientes figuras mostrarán esa habilidad desde el *vManage*.

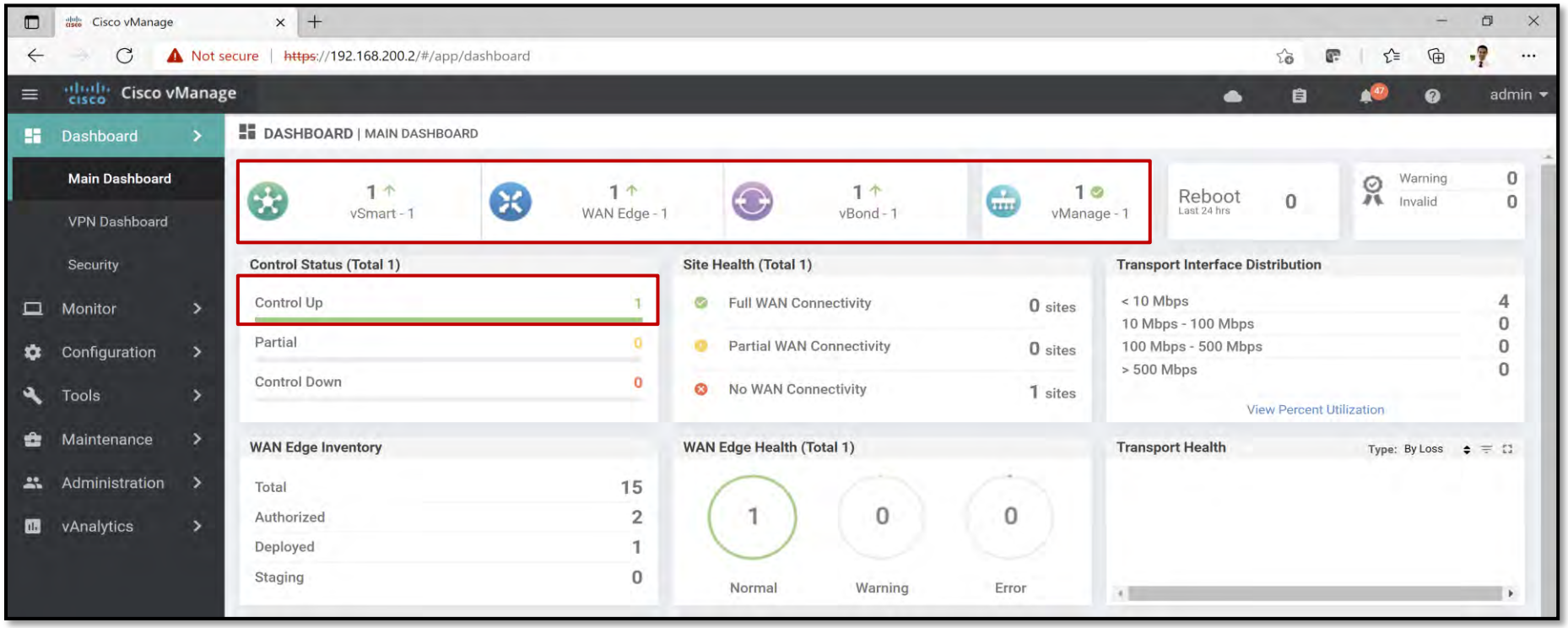
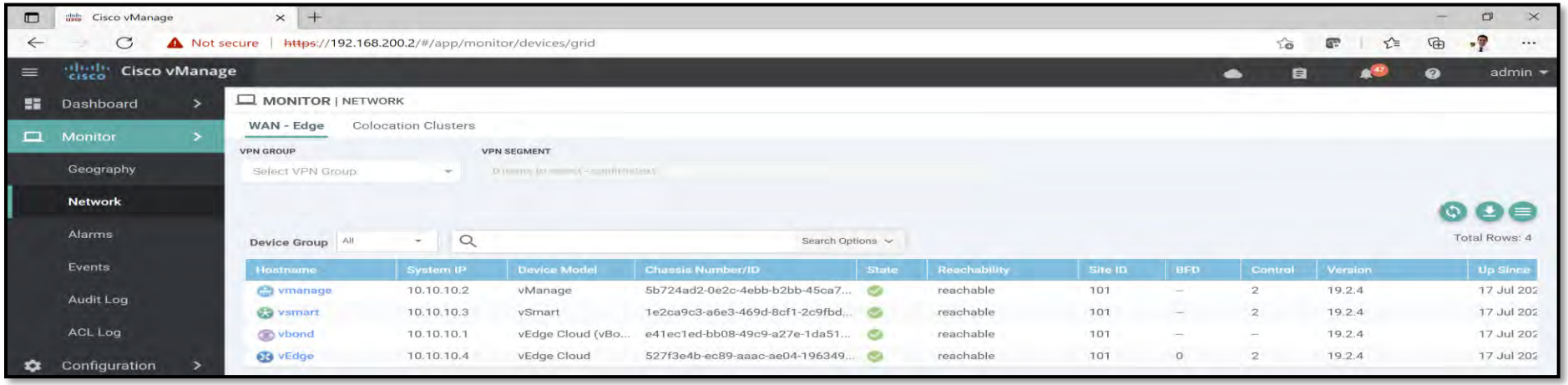
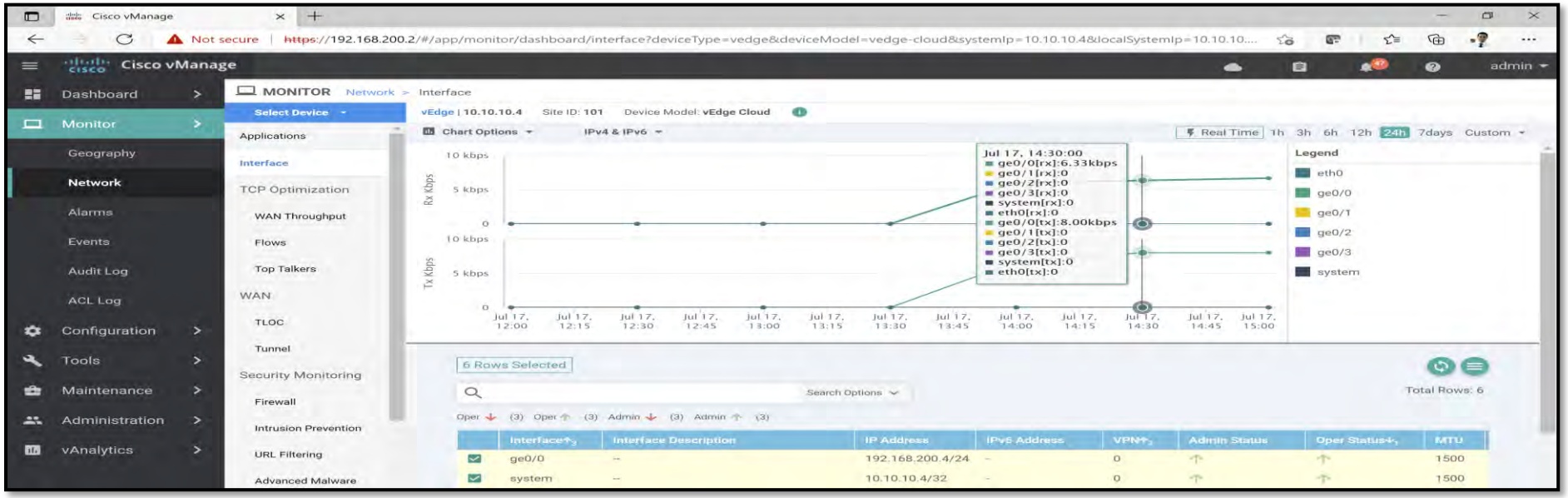


Figura 4-156 Main Dashboard de vManage- PoC SDWAN
Fuente: *Autor*

La *Fig. 4-156* evidencia la existencia de un Plano de Control (*vSmart, vBond* y *vManage*) y de un Plano de Datos (*WAN Edge*) activos y funcionales, sin error alguno, ni de autenticación, ni de certificados de encriptación.

La Sección *Monitor* tiene herramientas que permiten una visualización y control total de la infraestructura.





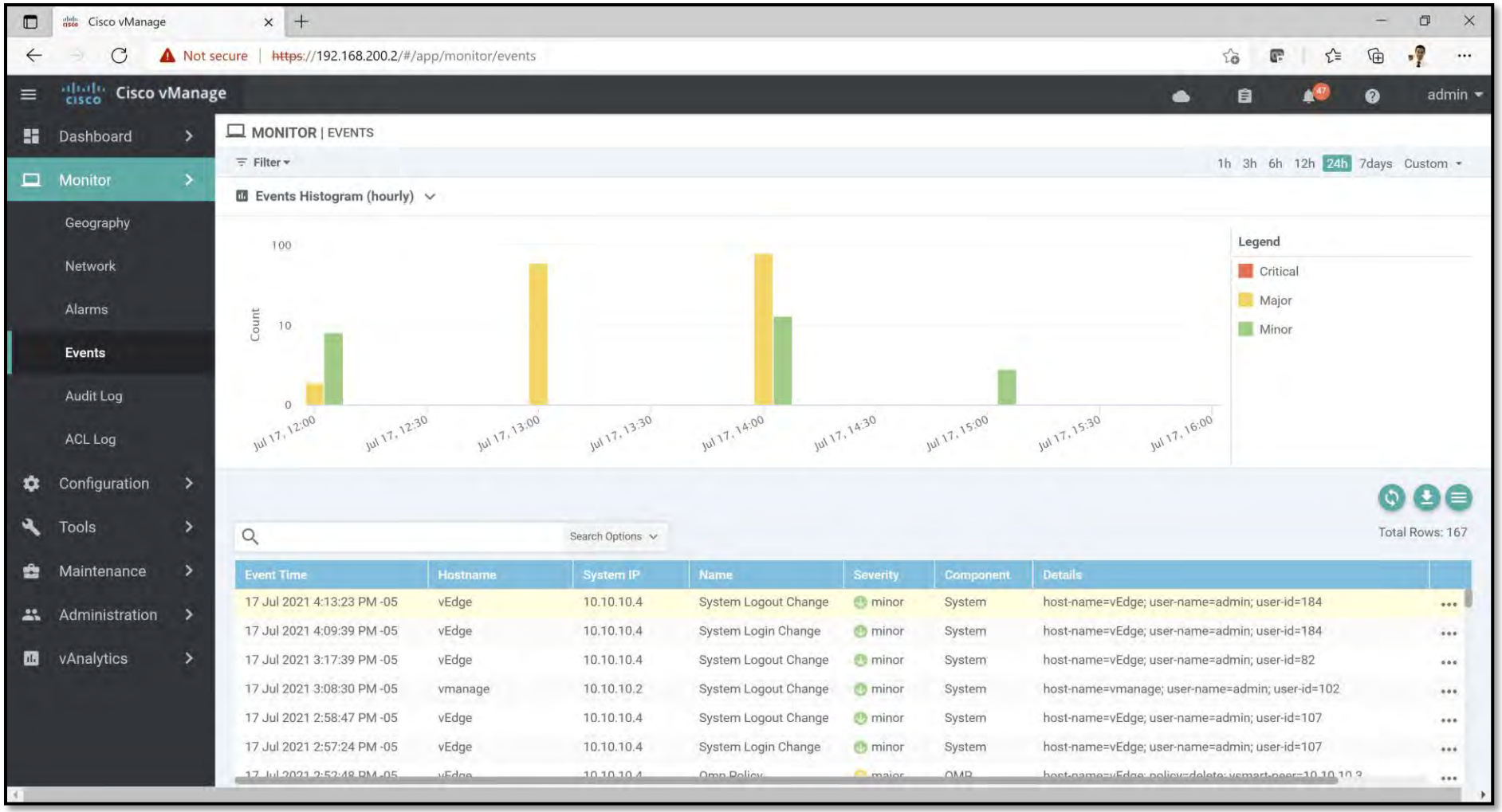


Figura 4-157 Monitoreo desde vManage- PoC SDWAN
Fuente: *Autor*

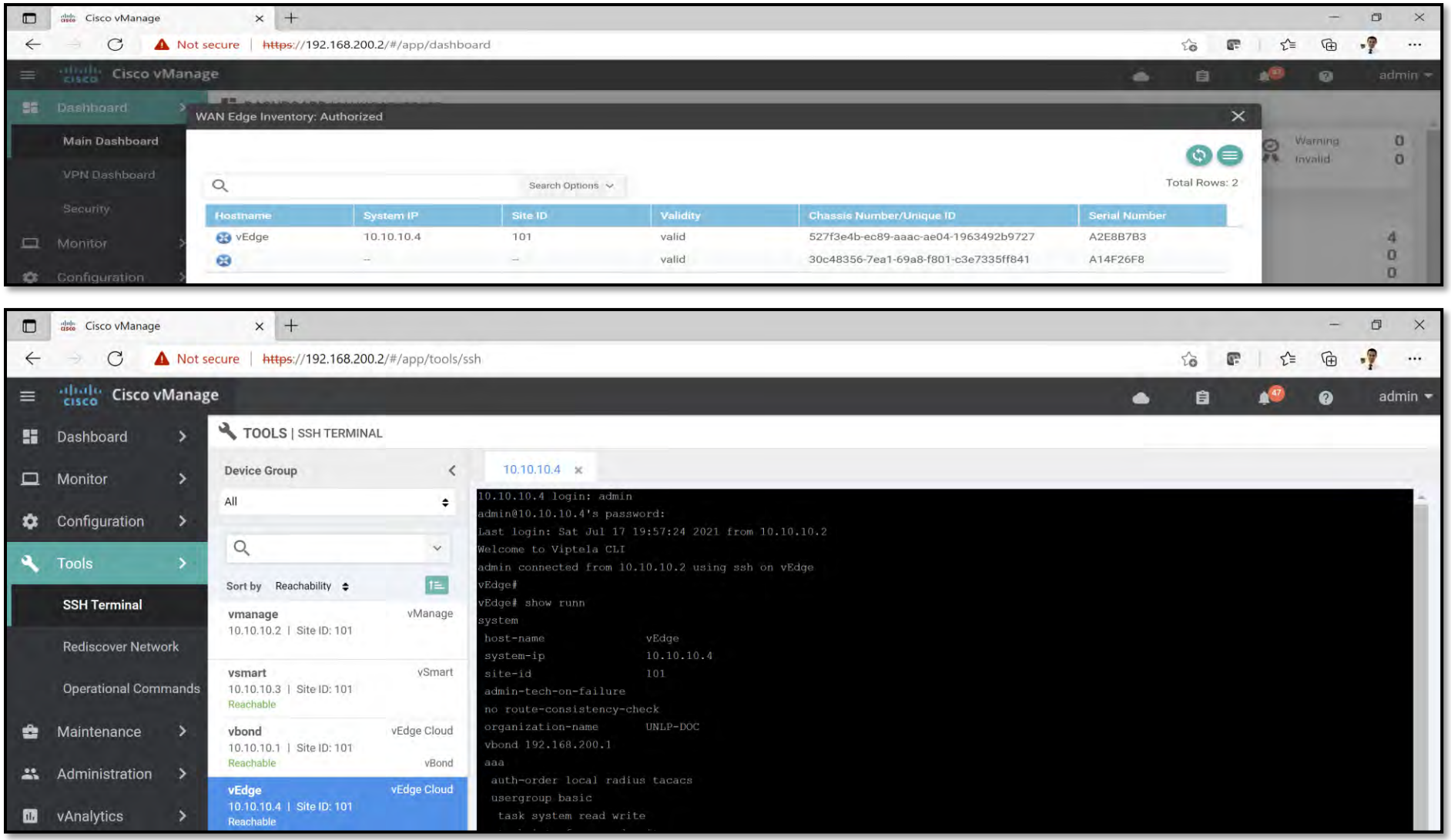


Figura 4-158 Control de Inventario y Acceso Remoto a vEdge desde vManage- PoC SDWAN
Fuente: *Autor*

```
vEdge# show omp peers
R -> routes received
I -> routes installed
S -> routes sent

                        DOMAIN     OVERLAY    SITE
PEER            TYPE    ID         ID         ID         STATE    UPTIME           R/I/S
-------------------------------------------------------------------------------------------
10.10.10.3      vsmart  1          1          101        up       0:02:06:42       0/0/0

vEdge#
```

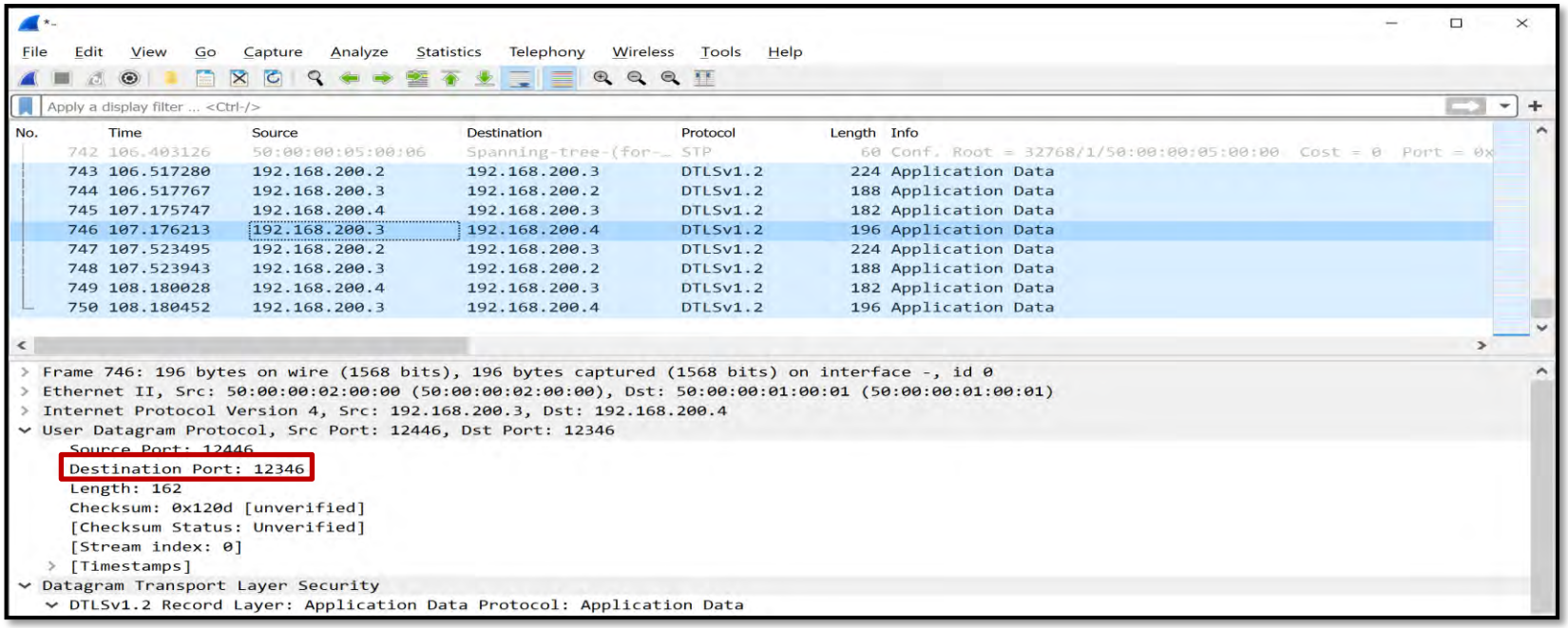


Figura 4-159 Establecimiento de paridad OMP entre vEdge-vSmart - PoC SDWAN

Fuente: *Autor*

La *Fig. 4-157* permite verificar el establecimiento de OMP (*Overlay Management Protocol*) una vez que los túneles DTLS entre *vSmart* y *vEdge* se establecieron. La captura de *Wireshark* toma los mensajes de OMP encriptados dentro de DTLS-UDP (**puerto 12346**).[70]

[70] Funcionamiento de OMP: https://www.lookingpoint.com/blog/cisco-sd-wan-omp

Algo adicional que se pudo determinar en el PoC, es el constante envío de *pings* (ICMP) entre *vSmart* y *vBond* a manera de *keepalives*.

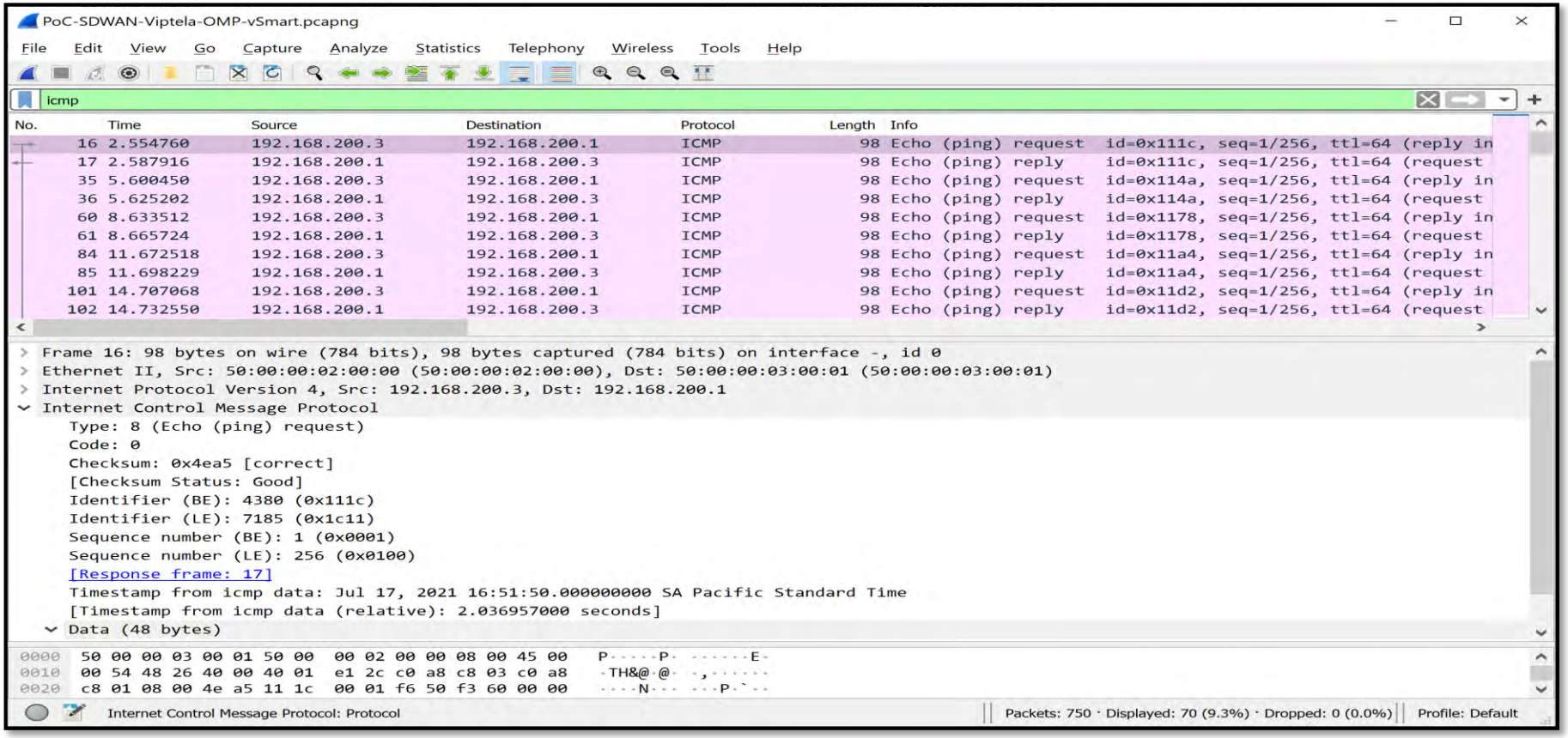


Figura 4-160 Keepalives (ICMP) entre vSmart-vBond - PoC SDWAN
Fuente: *Autor*

Al finalizar el presente PoC, se logró comprobar la capacidad general de SDWAN Viptela, su configuración básica, tanto de su Plano de Control como su Plano de Datos, además de las características de Monitoreo, Telemetría, inventario y control remoto sobre los *WAN Edges*.

SDWAN es también considerada una tecnología que permite unir redes tradicionales con conceptos de conexión de Redes Definidas por *Software*, ya que los *vEdges* pueden conectarse a cualquier *router* estableciendo enrutamiento mediante OSPF o BGP, pero en ese caso, OMP es necesario para que las rutas de esas sedes circulen desde los *vEdges/cEdges*[71] hacia el Plano de Control. OMP, como lo visto en la *Fig. 4-159*, se levanta entre los *WAN Edges* y *vSmart.*

El uso de APIs, *Templates* y Políticas facilitarán el manejo y control de equipos por parte del Plano de Control de SDWAN, convirtiendo a esta solución como una basada en políticas y contexto, lugar ideal para la inclusión de Inteligencia Artificial y programabilidad.

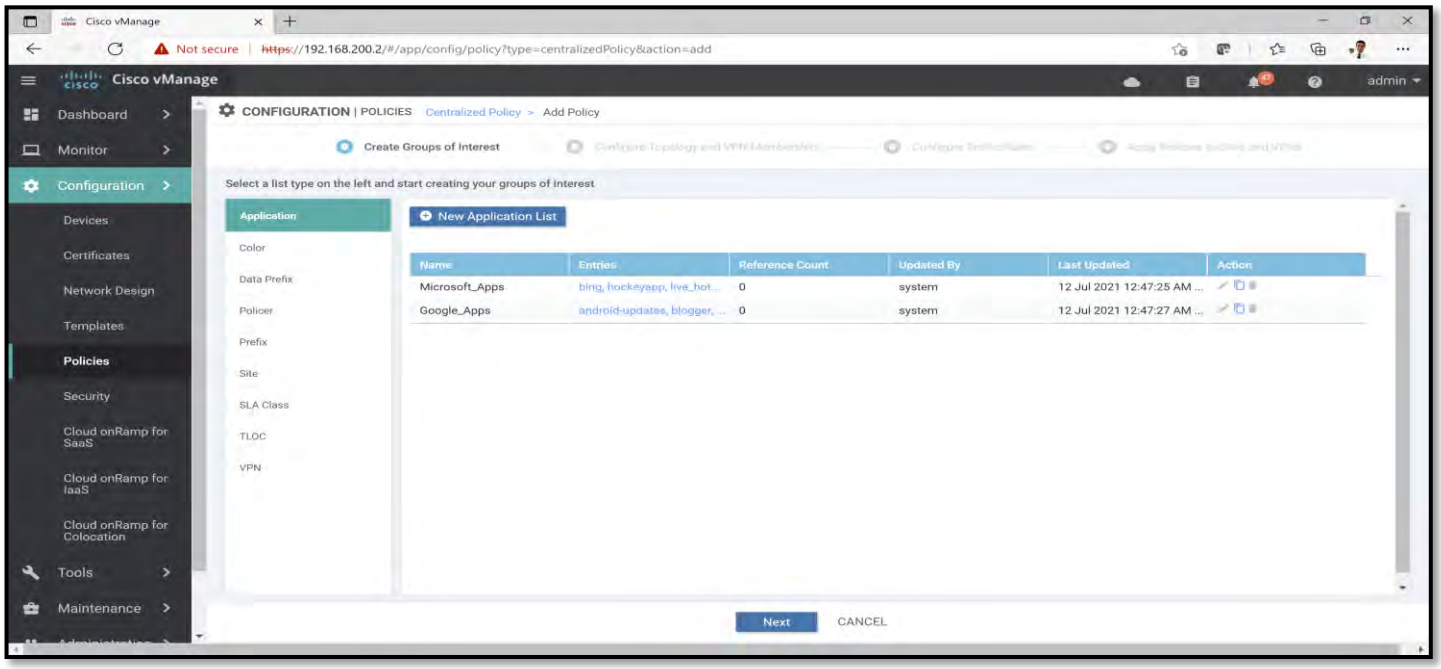


Figura 4-161 Políticas y Templates - PoC SDWAN
Fuente: *Autor*

Sin duda, SDWAN es una excelente tecnología para la comunicación empresarial de nueva generación que usa al Internet como *Fabric* de transporte, dándole así capacidad de conectividad ininterrumpida, segura y efectiva.

El concepto de SEN (*Secure Extensible Network*) fue comprobado a través del exitoso proceso de intercambio de certificados para autenticación, así como el uso de protocolos de encriptación robustos en los túneles de transporte de datos y de control.

[71] cEdge: Equipo tipo *router* con capacidad de soportar una imagen SDWAN (CSR1000v con IOS-XE SDWAN)

Capítulo 5

5. Implementación de Red Prototipo SDN

A lo largo de la presente tesis, se ha conceptualizado y puesto en práctica los diversos entornos donde el paradigma SDN puede ser implementado: *SD-Access/Branch*, *SD-WAN, SD-DC,* así como se realizó pruebas de concepto sobre una nueva generación de protocolos que facilitan que estas tecnologías puedan funcionar adecuadamente: LISP, VXLAN, *Segment-Routing*, además de protocolos que permiten la conectividad hacia los controladores SDN de entornos *OpenSource* y privados: *OpenFlow* y OMP, sin dejar de lado entornos de cambio cultural en la implementación rápida, segura y colaborativa de la transformación digital sustentada por SDN-*fabrics* enfocados a la automatización: *NetDevOps* en infraestructuras basadas en *OpenNetworking* y NG-SDN, permitiendo así una programabilidad al máximo nivel.

Los entornos SDN ofrecen un valor enorme a las organizaciones, principalmente por proveer de agilidad, simplicidad y reducción de costos en CAPEX y OPEX en las infraestructuras de comunicación a través del empleo de tecnologías basadas en *software*, virtualización, APIs, programabilidad y *Cloud Computing*, tomando mucha más importancia en época de pandemia COVID-19 en que la digitalización es parte de la nueva normalidad.

Por la importancia que tiene SDN, así como para completar el proceso de investigación científica, se realizará una implementación real de Red Prototipo SDN con equipos físicos habilitados para ese propósito.

El fabricante del *Hardware* seleccionado tiene de nombre *Northbound Networks*.

Figura 5-1 logo de Northbound Networks
Fuente: *Northbound Networks*

5.1 Hardware para redes SDN

Northbound Networks es una empresa australiana creada por Paul Zanna, quien también diseñó *Zodiac FX* y *Zodiac WX*, nombre del *Switch* y AP (*Access Point*) SDN a utilizar en este capítulo.

Según lo indicado por Zanna en (Northbound Networks, 2021), alrededor de los años 2013 empezó a indagar sobre SDN, pero en dicha época las únicas formas de experimentar con esta tecnología emergente eran mediante Mininet, un excelente emulador de redes de datos también utilizado en anteriores capítulos y equipos SDN costosos y propietarios diseñados exclusivamente para *Data Centers*, lo que imposibilitaba manipular paquetes reales generados por *hosts* reales, decidiendo hacer sus propios equipos con el fin de desarrollar pruebas de concepto y que otros investigadores puedan usarlos, es así como nace *Northbound Networks* y *Zodiac FX*, escribiendo su *firmware* desde cero y ensamblando alrededor de un lote de quinientos (500) equipos en una primera entrega, logrando que grandes multinacionales y centros universitarios de investigación de nuevas tecnologías soliciten estos dispositivos.

5.1.1 Zodiac FX SDN *Switch*

Este equipo es considerado uno de los mejores para pruebas de concepto físicas SDN a bajo costo y con capacidad de funcionamiento tipo *OpenSDN* (conexión a los controladores a través de *OpenFlow*).

Sus características físicas se muestran en la tabla siguiente:

Tabla 5-1 Características Zodiac FX – Switch SDN

Características	Descripción
Interfaces	Cuatro (4) Interfaces 10/100 (*Fast Ethernet*)
Accesso CLI-WebUI	A través de Puerto USB (COM)
Procesador	AMTEL ATSAM4E Cortex M4
Soporte OpenFlow	OpenFlow 1.0 y 1.3
Entradas en *Flow Table*	<512
Soporte VLANs/Trunks	IEEE 802.1Q - <4096 VIDs
QoS	IEEE 802.1Q Priority Tag
Autenticación	IEEE 802.1x
Especiales	<16 ACLs - 2KB Jumbo-Frames
Medidas	10x8 cm

Basado en (Kickstarter Projects, 2021)

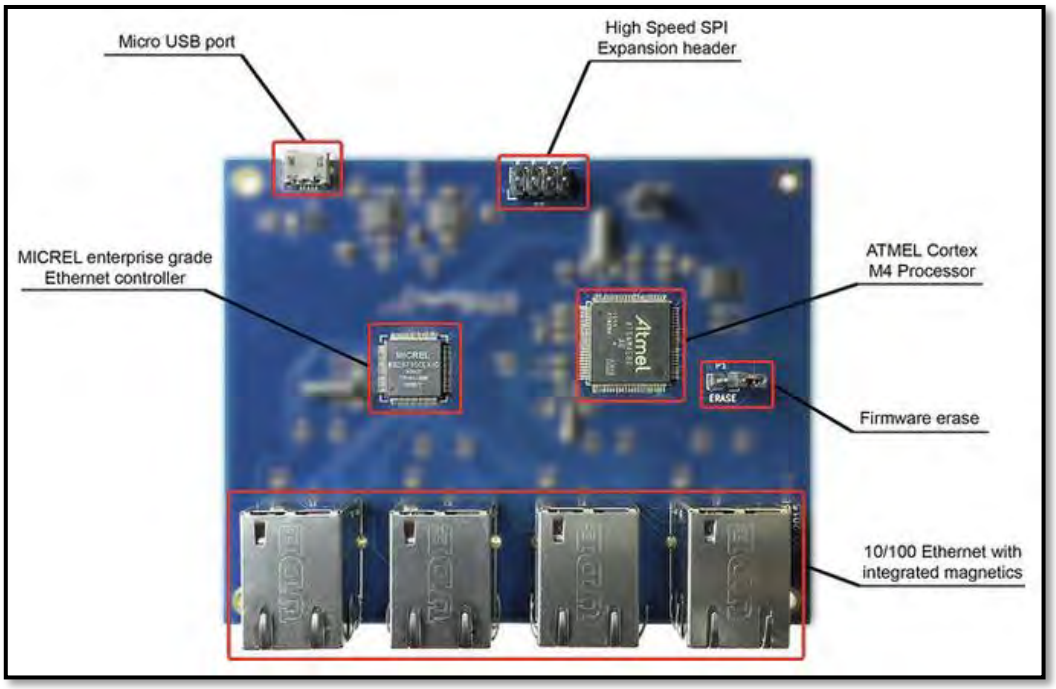


Figura 5-2 Zodiac FX – Características Físicas
Recuperado de (Kickstarter Projects, 2021)

Una de las grandes ventajas es que *Zodiac FX* está construido en base a un *Firmware* totalmente *OpenSource* desarrollado y actualizado por la comunidad de desarrollo SDN, incluso es posible personalizarlo.

Este equipo emplea ASF (*Atmel Software Framework*) para la conexión a periféricos externos como USB, SPI, etc, además hace uso de *FreeRTOS*[72] para administrar tareas y memoria de sus tres procesos principales: Acceso por CLI, *Switching* y SDN (*OpenFlow*).

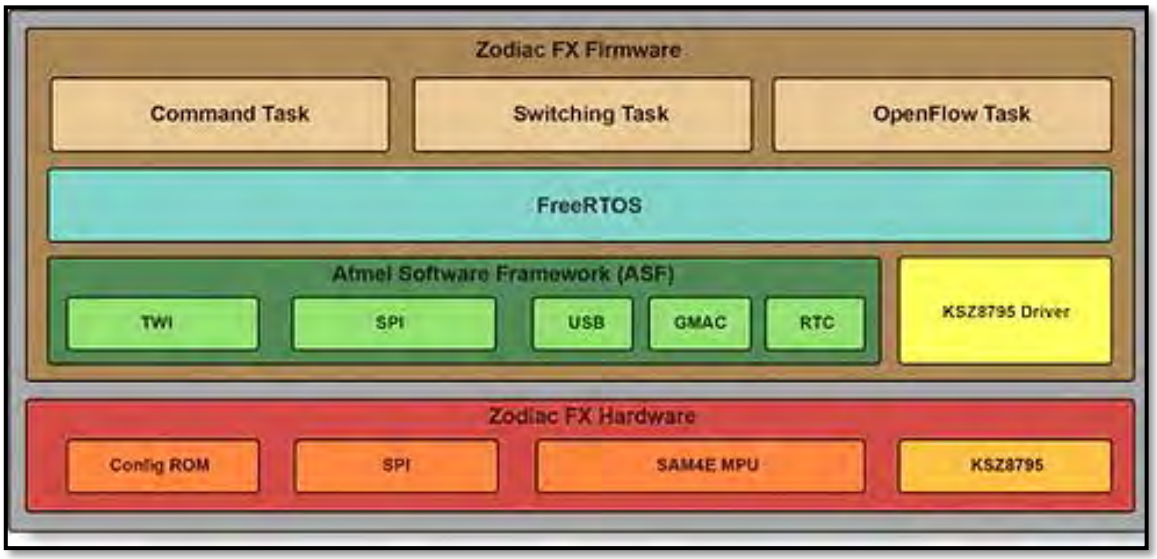


Figura 5-3 Esquema de Zodiac FX
Recuperado de (Kickstarter Projects, 2021)

Más información sobre este gran equipo se encuentra en:

https://github.com/NorthboundNetworks/ZodiacFX

https://forums.northboundnetworks.com/index.php?topic=1055.0

https://northboundnetworks.freshdesk.com/support/solutions

5.1.2 Zodiac WX SDN *Wireless AP*

Zodiac WX es un *Access Point* con capacidad de soportar SDN a través de *OpenFlow* a bajo costo creado por *Northbound Networks* basado en Zodiac FX; según *Northbound Networks*, es el primer AP con un *OpenFlow Engine* completo, dando una imagen a los controladores *OpenSDN* de que se trata de un *switch OpenFlow* cableado.

Así como APs tradicionales, este equipo puede energizarse a través de PoE (*Power over Ethernet*) o con un cargador de poder.

Su sistema operativo está basado en *OpenWRT* que *Northbound Networks* denominó *Zodiac LEDE*.

En cuanto al procesador, es un MIPS 74Kc con 128MB de RAM.

Cuenta con dos (2) conexiones para *Gigabit Ethernet* para la red cableada y el uso del estándar IEEE 802.11ac para la red *Wireless*.

[72] FreeRTOS – Real-Time Operating System for Microcontrollers - https://www.freertos.org/

Puede soportar un total de ocho (8) grupos, dieciséis (16) tablas y quinientos (512) entradas de flujo.

Figura 5-4 Zodiac WX
Recuperado de (Kickstarter Projects, 2021)

El proceso de envío en el SD-AP funciona a través de la asignación de puertos inalámbricos:

- Puerto cableado al puerto *OpenFlow 65*
- Clientes *Wireless* a 5GHz a los puertos *OpenFlow* 1-32
- Clientes *Wireless* a 2.4GHz a los puertos *OpenFlow* 33-64

El controlador SDN manipula los flujos de paquetes con la asignación de puertos previamente indicada a través de las Dir. MAC en el Controlador SDN.

La instalación, configuración básica y puesta en marcha tanto de *Zodiac FX* y *Zodiac WX* se encuentra en ***Anexo H: Entrada en Funcionamiento de Zodiac FX y Zodiac WX – SDN en infraestructura física con Aruba VAN SDN Controller y RYU-FlowManager***

5.2 Topología de la Red SDN

La topología de la red a probar su funcionamiento y realizar capturas de tráfico se puede apreciar en las *Fig. 5-5* y *Fig.5-6*.

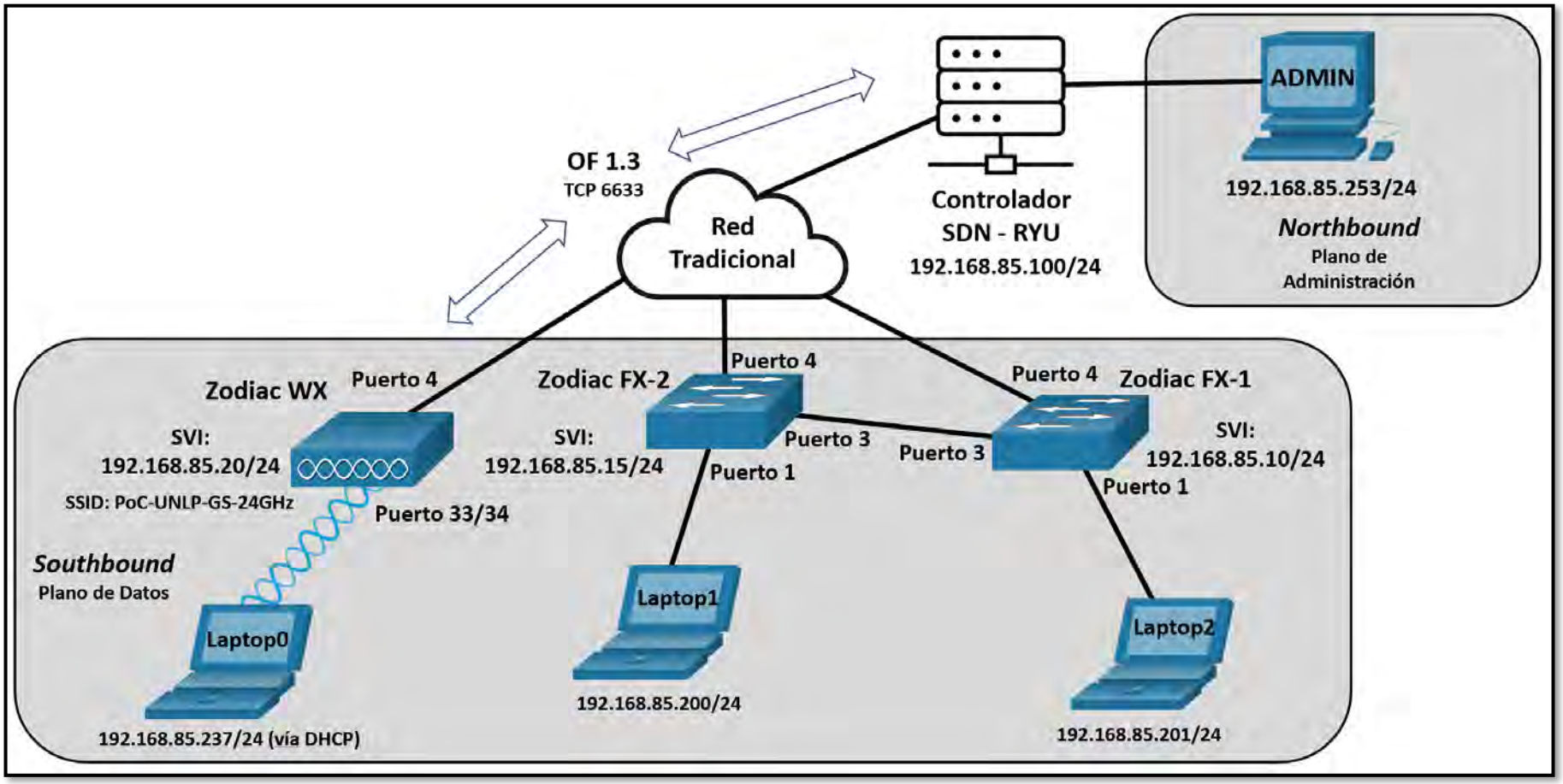


Figura 5-5 Topología PoC SDN en equipos físicos
Fuente: *Autor*

Figura 5-6 Red Física SDN implementada mediante Zodiac FX(2), Zodiac WX y Controlador RYU – Inicio de Cableado
Fuente: *Autor*

El proceso de configuración de los equipos se describe en el ***Anexo H: Entrada en Funcionamiento de Zodiac FX y Zodiac WX – SDN en infraestructura física con Aruba VAN SDN Controller y RYU-FlowManager*.**

Partiendo de ello, la configuración de los equipos del Plano de Datos es la siguiente:

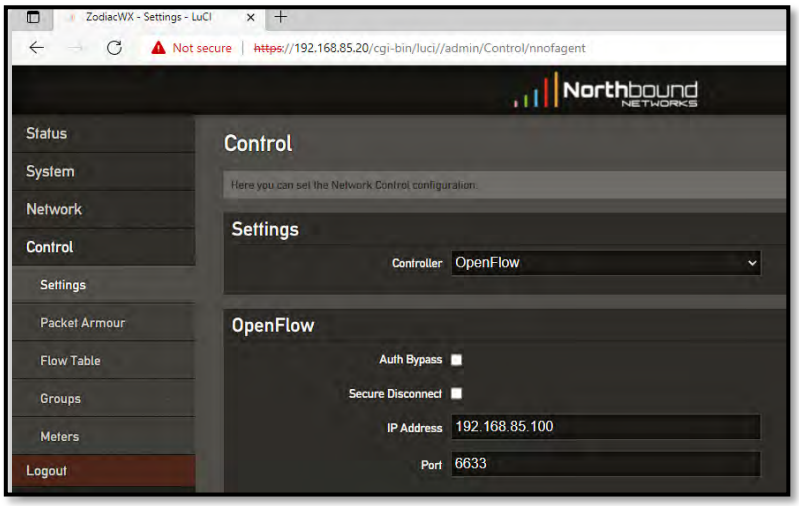
a)

```
Zodiac_FX# config
Zodiac_FX(config)# show config

-----------------------------------------
Configuration
 Name: Zodiac_FX
 MAC Address: 70:B3:D5:6C:DF:0D
 IP Address: 192.168.85.10
 Netmask: 255.255.255.0
 Gateway: 192.168.85.1
 OpenFlow Controller: 192.168.85.100
 OpenFlow Port: 6633
 Openflow Status: Enabled
 Failstate: Secure
 Force OpenFlow version: Disabled
 EtherType Filtering: Disabled
```

b)

```
Zodiac_FX# config
Zodiac_FX(config)# show config

-----------------------------------------
Configuration
 Name: Zodiac_FX
 MAC Address: 70:B3:D5:6C:DE:F6
 IP Address: 192.168.85.15
 Netmask: 255.255.255.0
 Gateway: 192.168.85.1
 OpenFlow Controller: 192.168.85.100
 OpenFlow Port: 6633
 Openflow Status: Enabled
 Failstate: Secure
 Force OpenFlow version: Disabled
 EtherType Filtering: Disabled
```

c)



d)

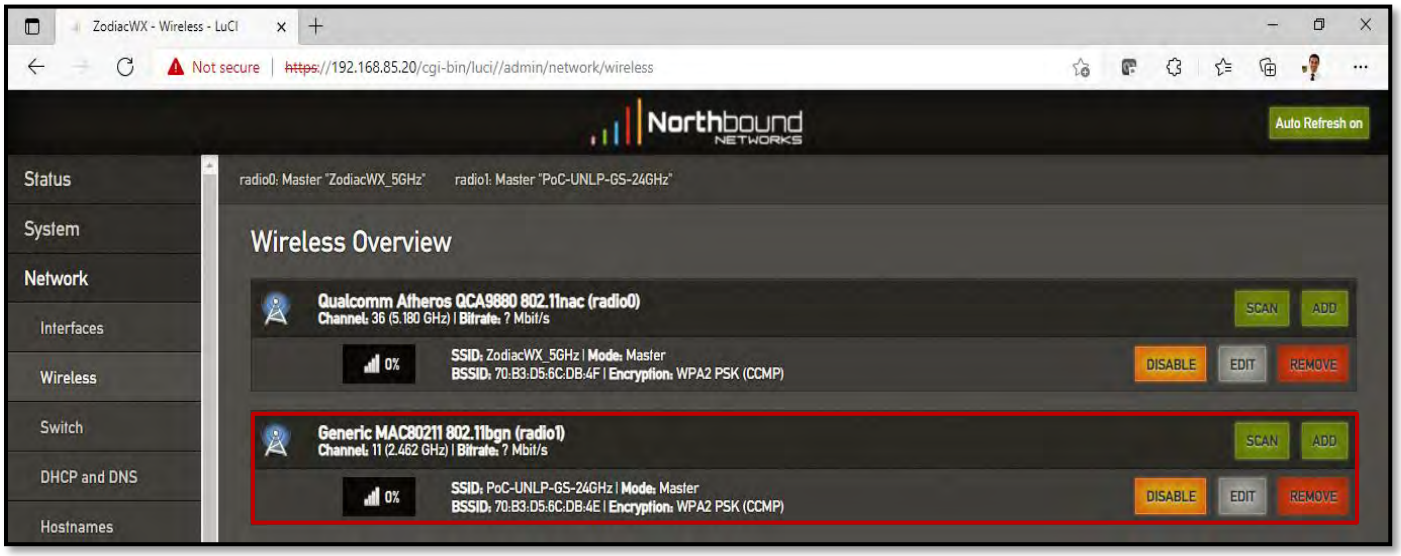


Figura 5-7 Configuración realizada en Zodiac FX-1(a), Zodiac FX-2(b) y Zodiac WX (c y d)
Fuente: *Autor*

Se debe verificar que exista comunicación entre el Plano de Control y el Plano de Datos. El Controlador tiene la dir. IP **192.168.85.100/24**

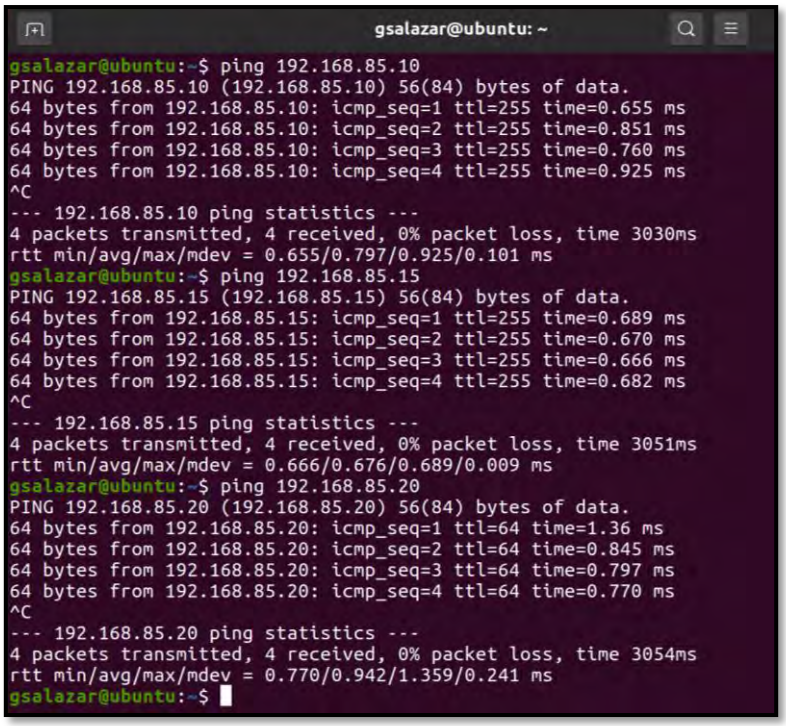


Figura 5-8 Conectividad entre Controlador RYU con Zodiac FX-1, Zodiac FX-2 y Zodiac WX
Fuente: *Autor*

El Controlador *RYU* está funcional y con *FlowManager* activo para el control de flujos de los *hosts* finales.

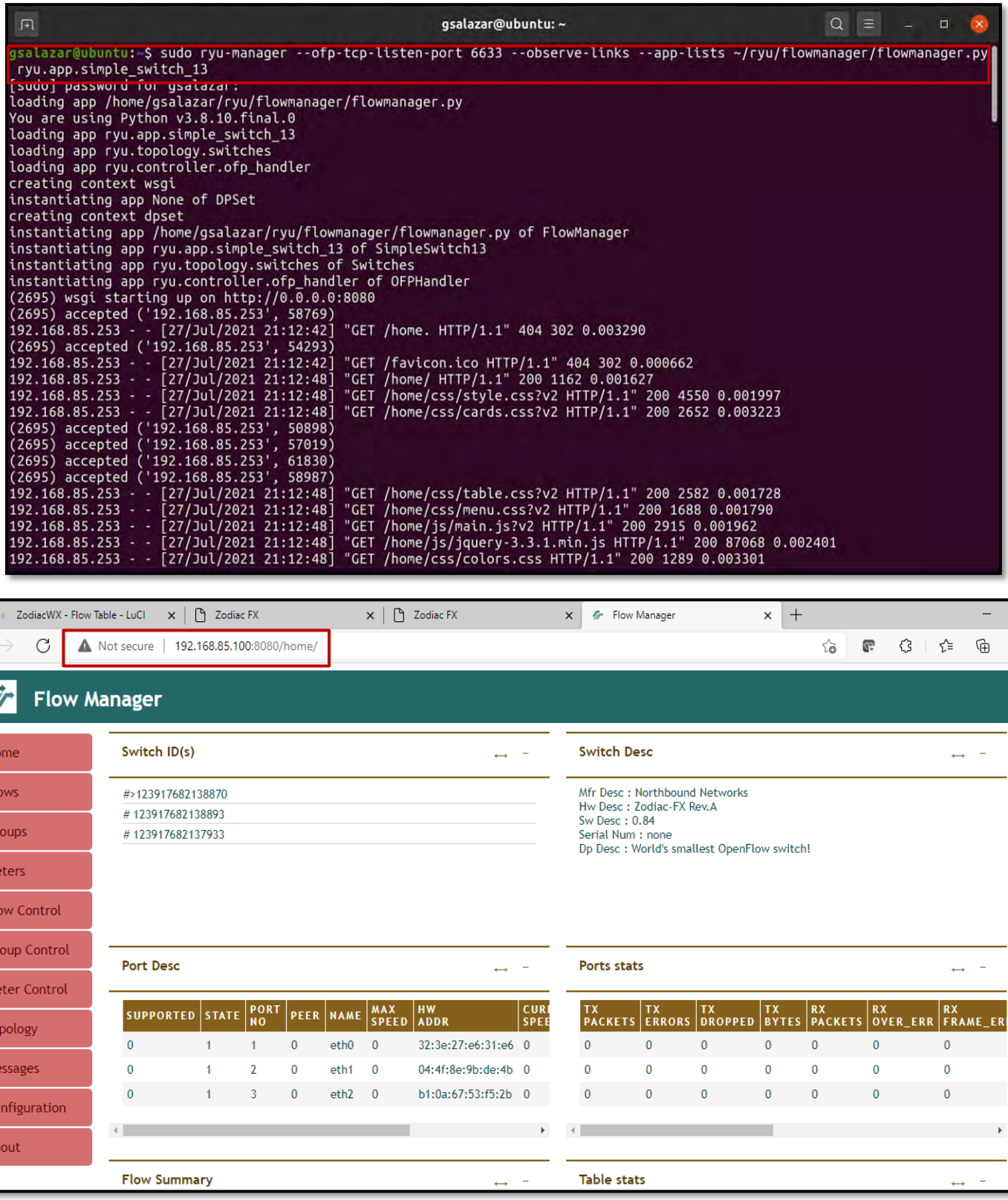


Figura 5-9 Activación de Controlador RYU y FlowManager para gestionar Zodiac FX-1, Zodiac FX-2 y Zodiac WX
Fuente: *Autor*

Al observar la topología y flujos en *FlowManager*, se determina el éxito de la implementación de la red física SDN

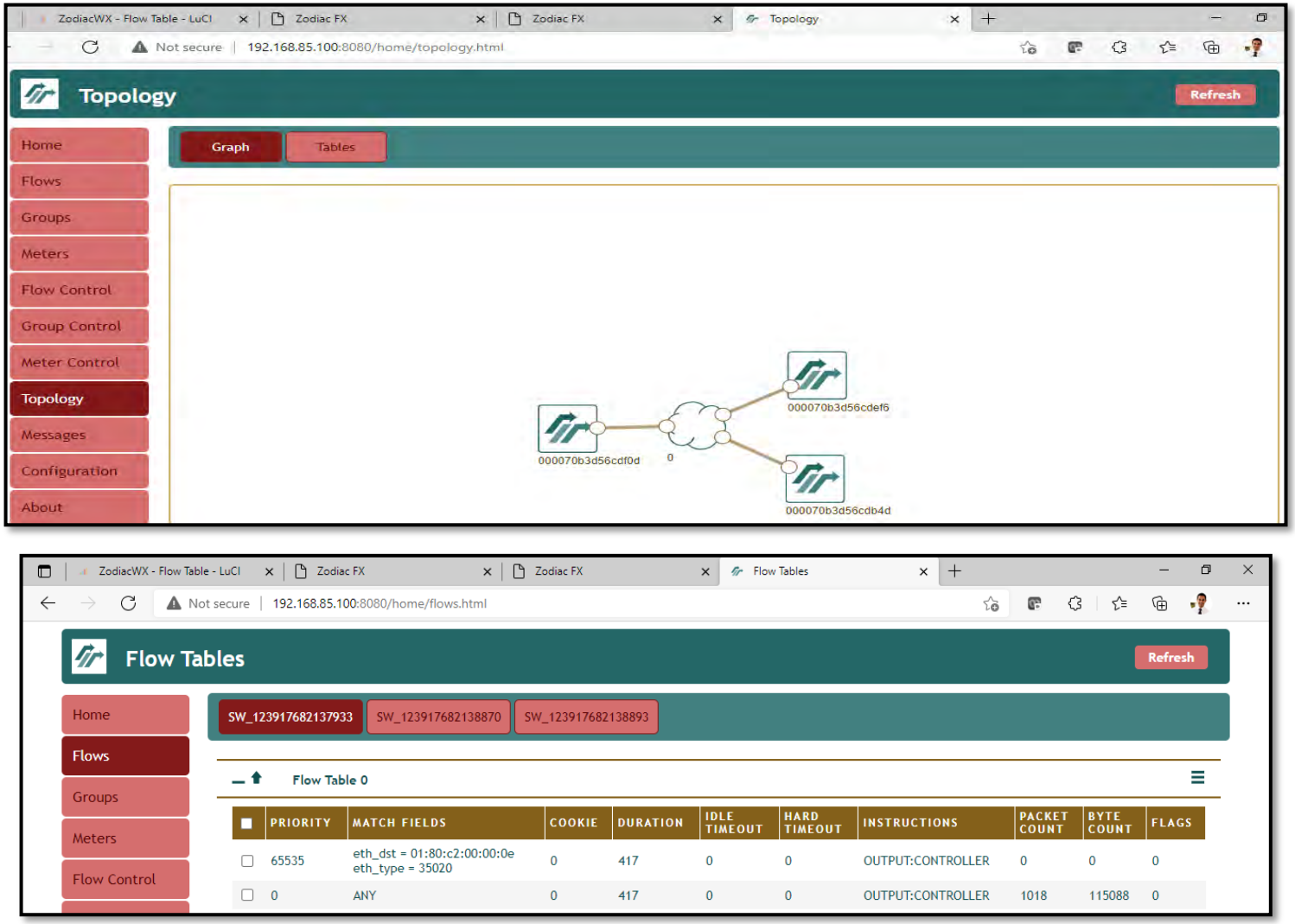
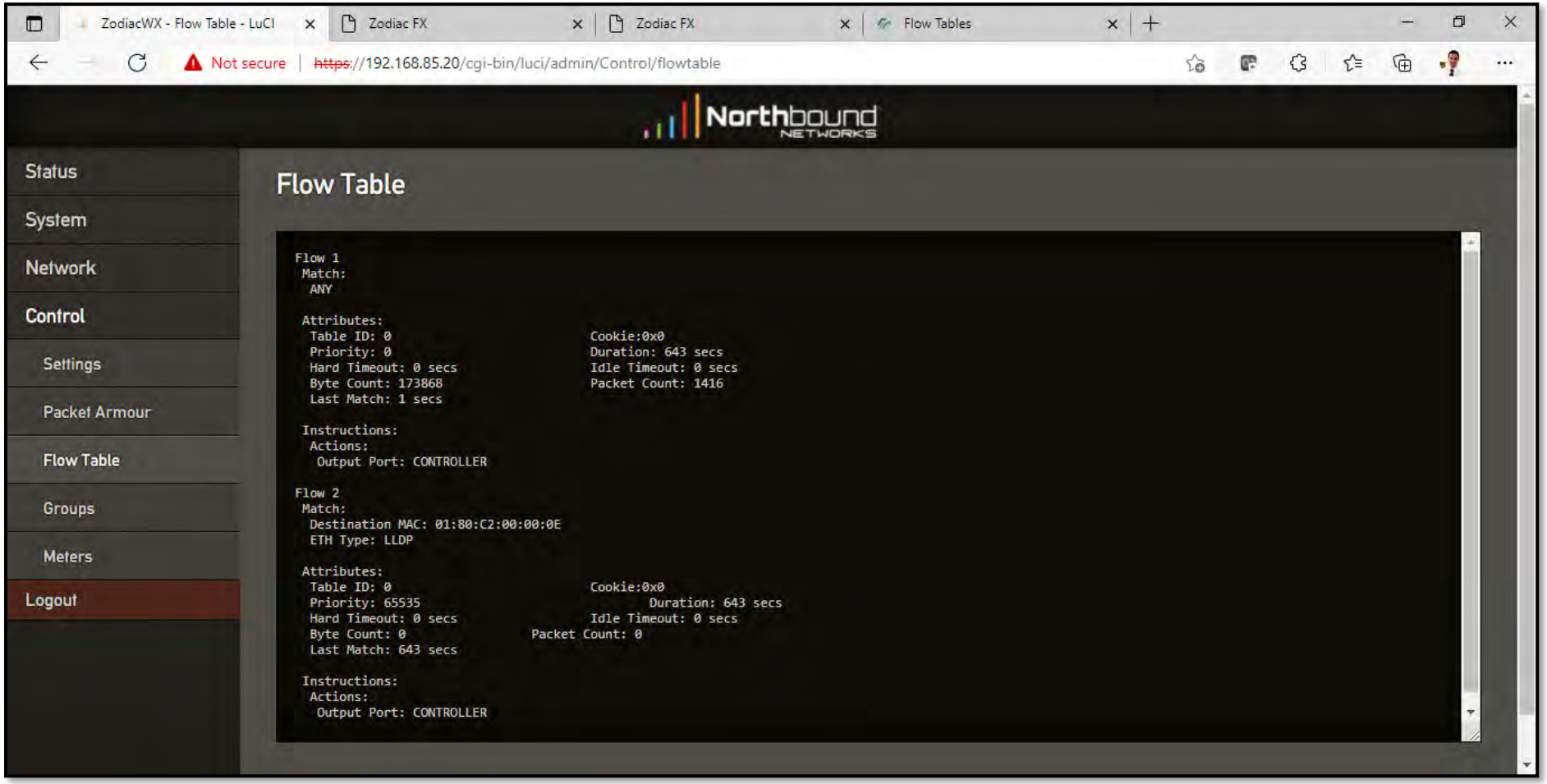


Figura 5-10 Topología y Flujos generados en Controlador RYU-FlowManager para Zodiac FX-1, Zodiac FX-2 y Zodiac WX
Fuente: *Autor*

Los flujos de la última imagen de la *Fig. 5-10* fueron cargados correctamente en los equipos del plano de datos tal como se aprecia en la siguiente imagen:



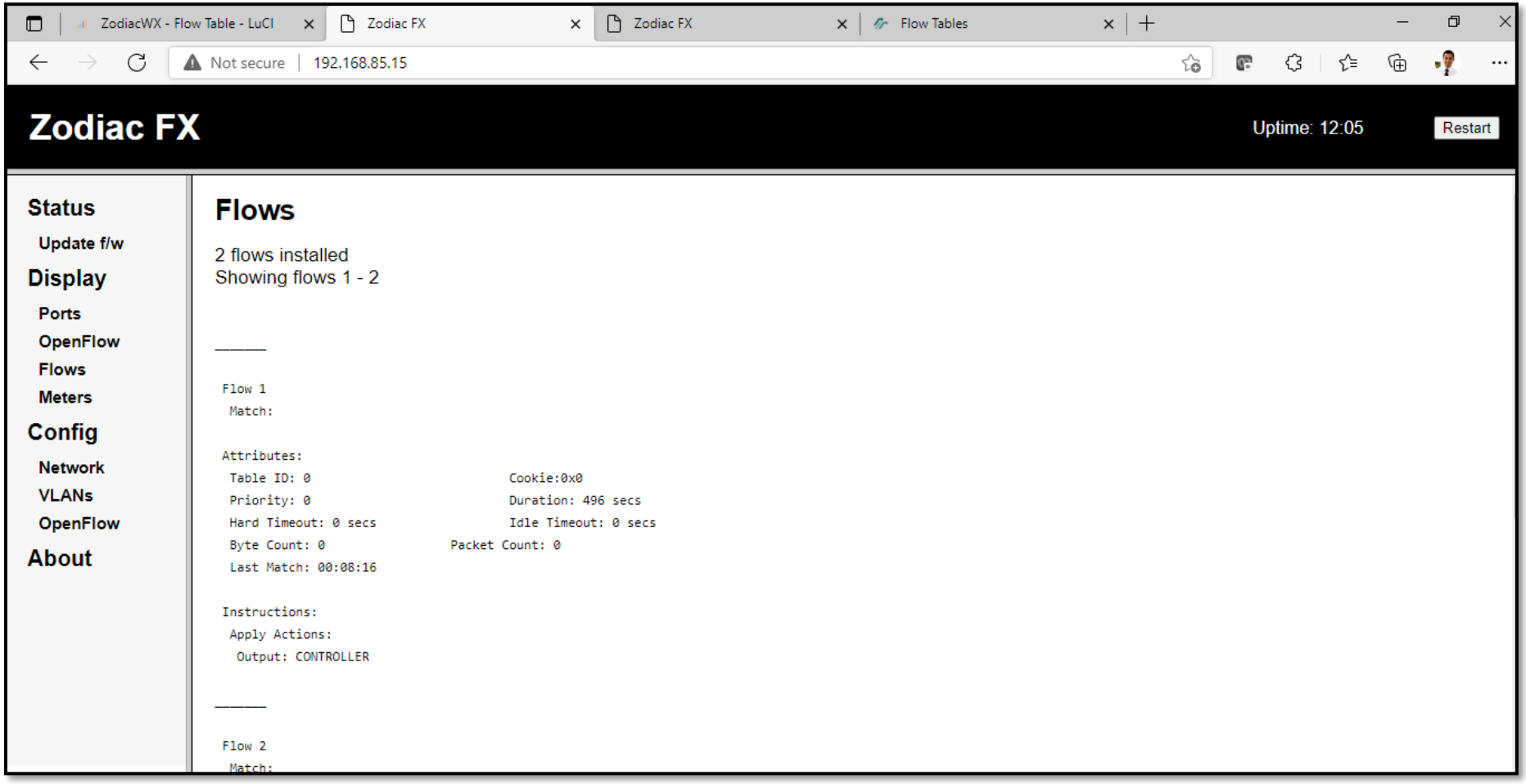


Figura 5-11 Flujos cargados desde el Controlador RYU-FlowManager a Zodiac FX-1, Zodiac FX-2 y Zodiac WX
Fuente: *Autor*

Analizando los dos flujos instalados al momento de inicializar el Controlador *RYU*, se determina los siguiente:

- Flujo 1: Envío de cualquier mensaje recibido por el dispositivo (sea este *Zodiac FX* o *Zodia WX*) al Controlador:

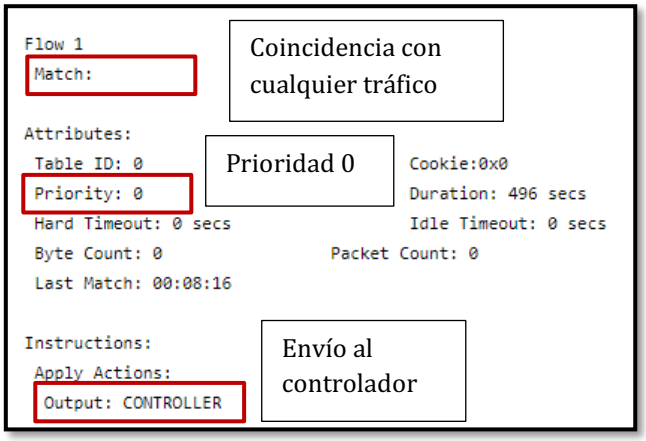


Figura 5-12 Análisis Flujo 1 cargado a Plano de Datos – RYU
Fuente: *Autor*

- Flujo 2: Protocolo LLDP (*Logical-Link Discovery Protocol*) para el descubrimiento de dispositivos conectados directamente. Se información de este protocolo al controlador para funciones de inventario, descubrimiento de equipos, generación de topología y telemetría.

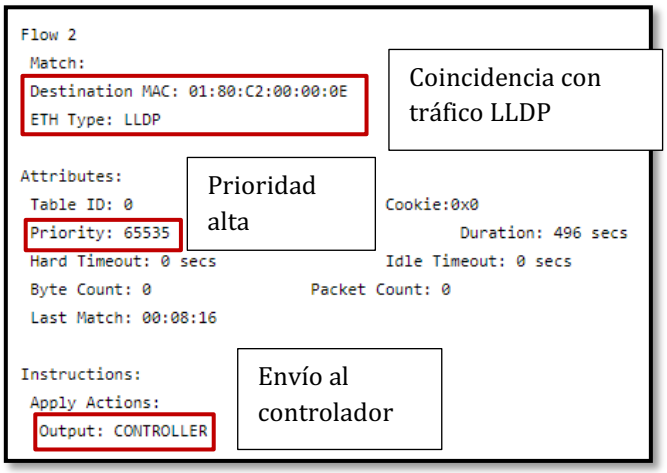


Figura 5-13 Análisis Flujo 2 cargado a Plano de Datos – RYU
Fuente: *Autor*

5.3 Resultados de Conectividad en Infraestructura SDN Física

Con la red totalmente funcional, se conectan dispositivos finales al Plano de Datos y se prueba su conectividad:

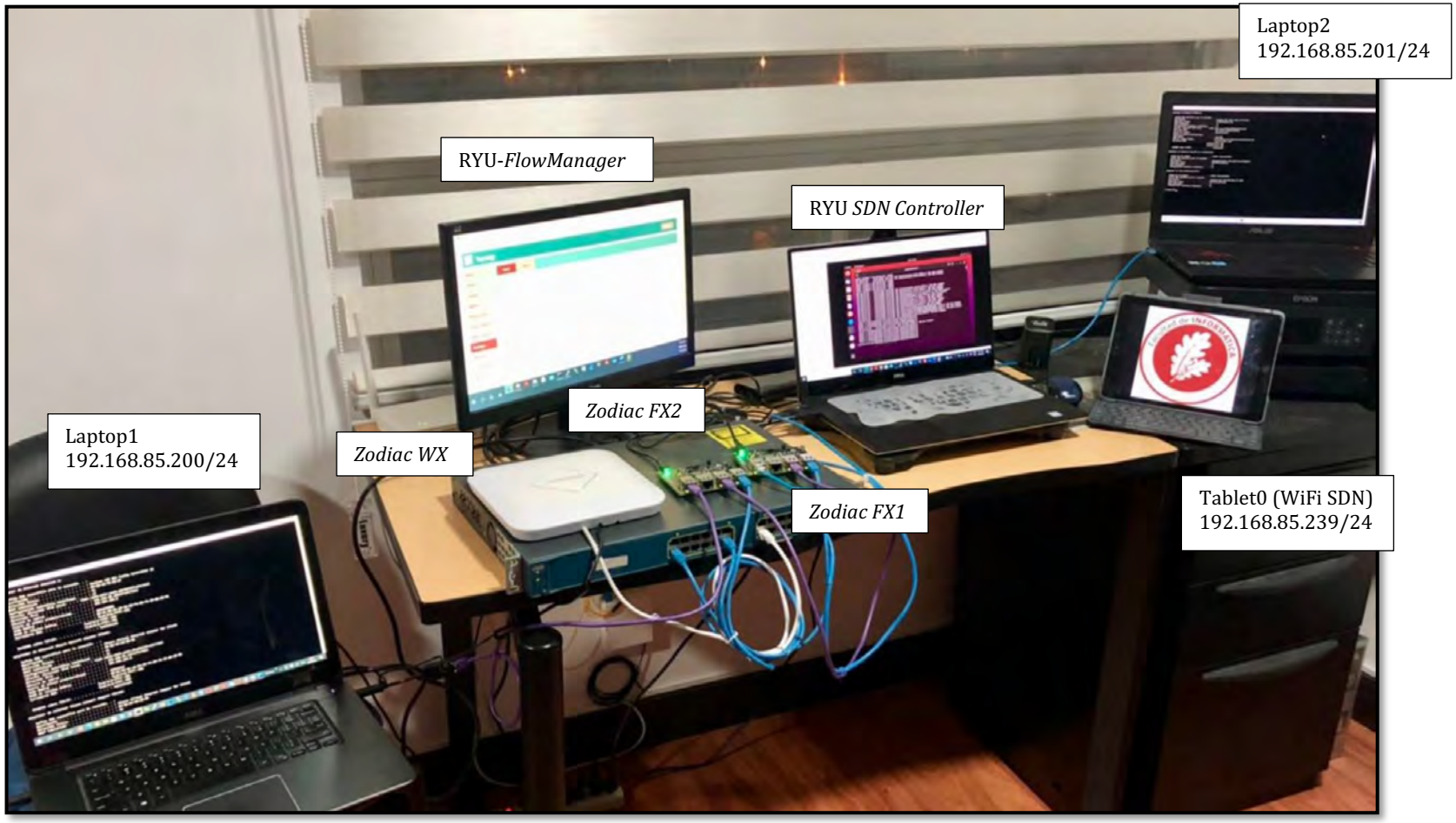


Figura 5-14 Red física SDN – Prueba de Conectividad
Fuente: *Autor*

Para que la conectividad pueda darse en dentro del Plano de Datos, se requiere configurar nuevos flujos en el Controlador *RYU* y permitir dicha comunicación bidireccional.

La primera prueba será entre las dos Laptops cableadas.

Según la *Fig. 5-5* y *Fig. 5-14,* las dir. IP de los equipos son:

- Laptop 1: 192.168.85.200/24

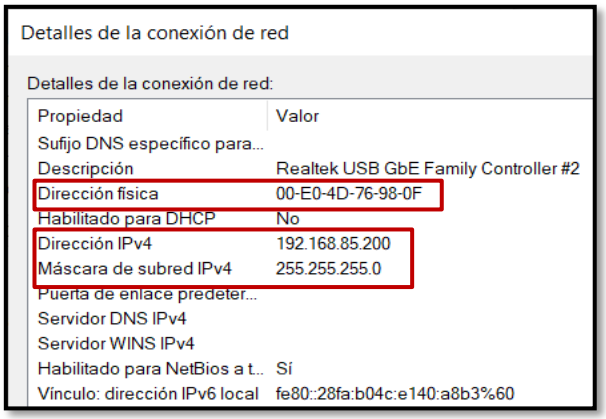


Figura 5-15 Dir. IP Laptop1 - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

- Laptop 2: 192.168.85.201/24

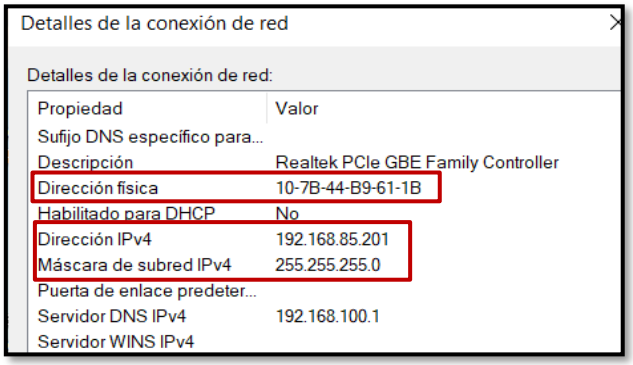


Figura 5-16 Dir. IP Laptop2 - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

Al intentar realizar un *ping* entre los dos equipos, la conectividad no es posible.

```
C:\Users\PC>ping 192.168.85.200

Haciendo ping a 192.168.85.200 con 32 bytes de datos:
Respuesta desde 192.168.85.201: Host de destino inaccesible.
Respuesta desde 192.168.85.201: Host de destino inaccesible.
Respuesta desde 192.168.85.201: Host de destino inaccesible.
Respuesta desde 192.168.85.201: Host de destino inaccesible.

Estadísticas de ping para 192.168.85.200:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
```

Figura 5-17 Ping fallido debido a falta de flujos - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

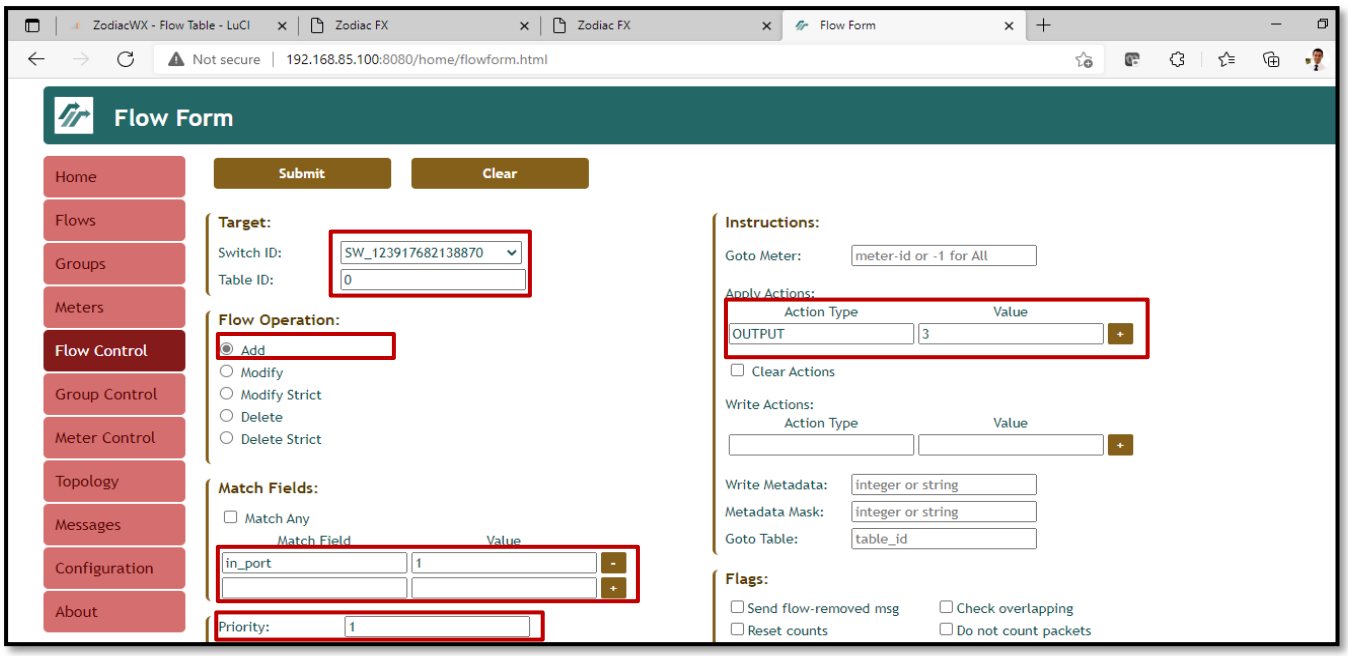
Como se observa en la figura anterior, no se pueden conectar los *hosts* ya que no existen flujos establecidos en el Controlador, los cuales se envían (*push*) a los *switches* que lo permitan.

Para instanciar dichos flujos, se ingresa a *FlowManager* del controlador *RYU*, específicamente a la sección *Flow Control* del menú lateral.

Los flujos para establecer una comunicación bidireccional son los siguientes:

- *Switch Zodiac FX-2* (Dir. IP: 192.168.85.15 – *SwitchID:* 123917682138870)

  Flujo para el tráfico de Ida hacia el destino (desde Laptop1 – conectado al puerto 1 hacia Laptop2 conectado a través de *Switch Zodiac FX-1* en el puerto 3)



  Flujo para el tráfico de Regreso hacia el origen (desde Laptop2 conectado a través de *Switch Zodiac FX-1* en el puerto 3 hacia Laptop1 – conectado al puerto 1)

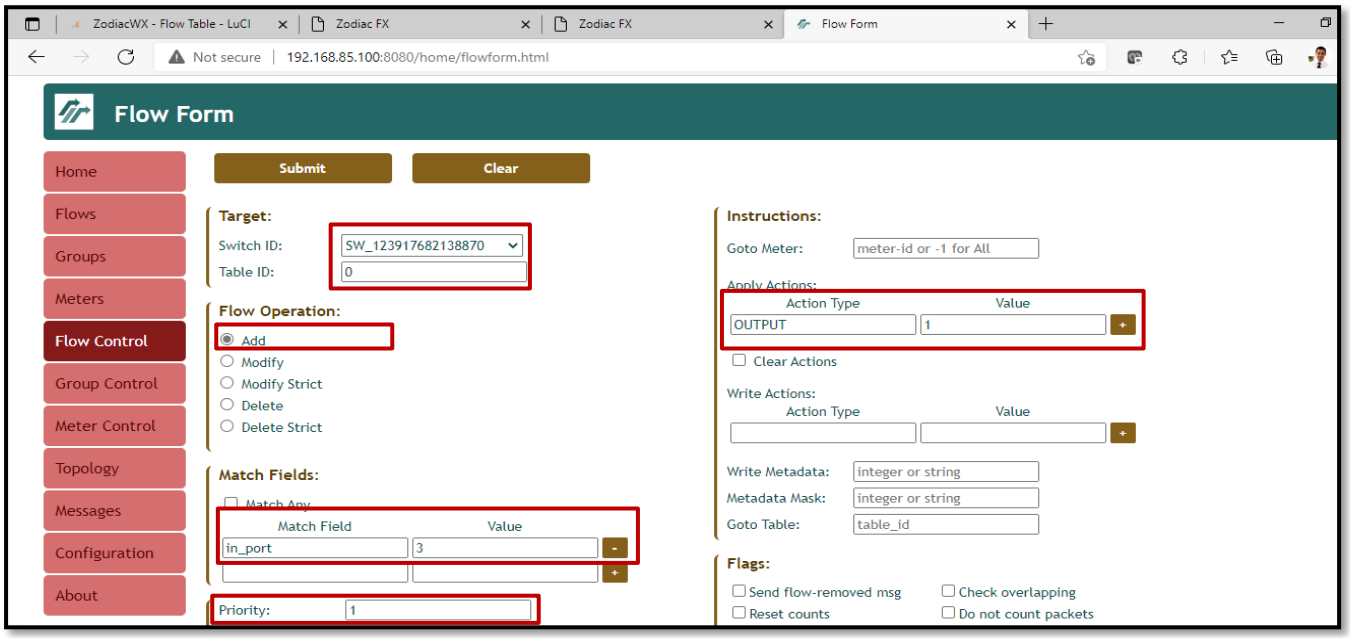
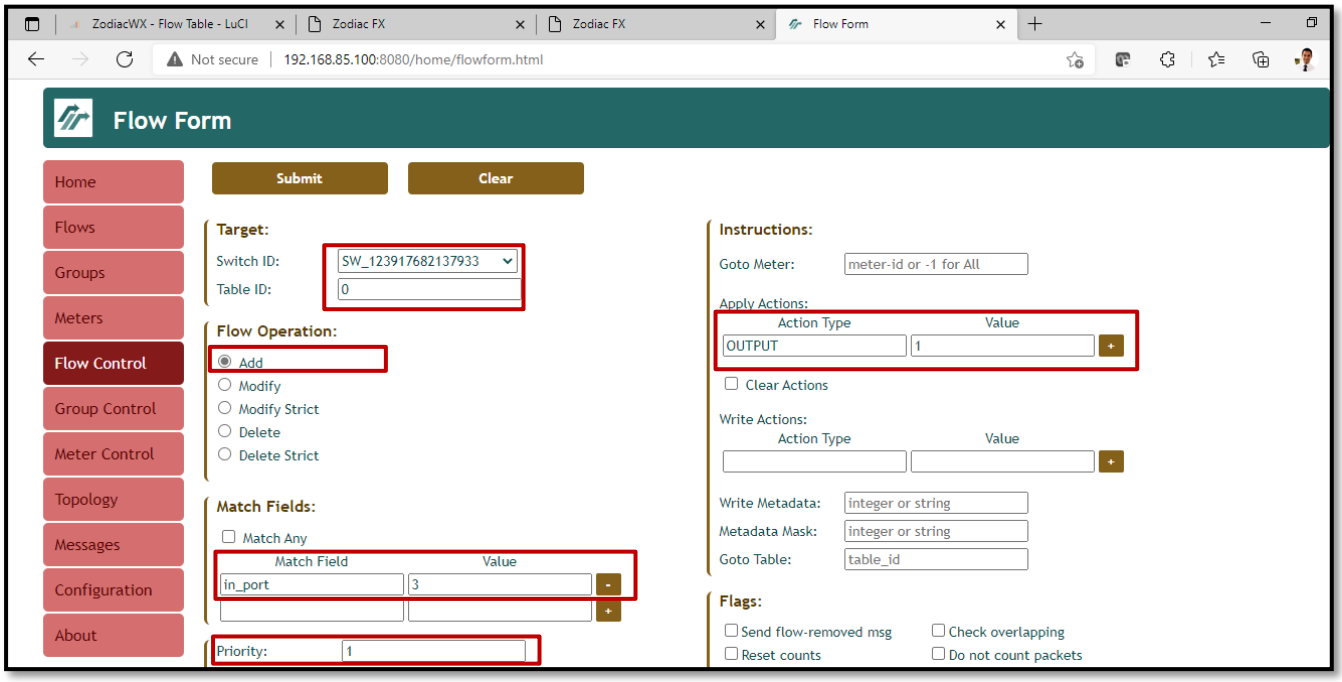


Figura 5-18 Flujos para Switch Zodiac FX-2 - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

- *Switch Zodiac FX-1* (Dir. IP: 192.168.85.10 – *SwitchID:* 123917682137933)

  Flujo para el tráfico de Ida hacia el origen (desde Laptop2 – conectado al puerto 1 hacia Laptop1 conectado a través de *Switch Zodiac FX-2* en el puerto 3)



Flujo para el tráfico de Regreso hacia el destino (desde Laptop1 conectado a través de *Switch Zodiac FX-2* en el puerto 3 hacia Laptop2 – conectado al puerto 1)

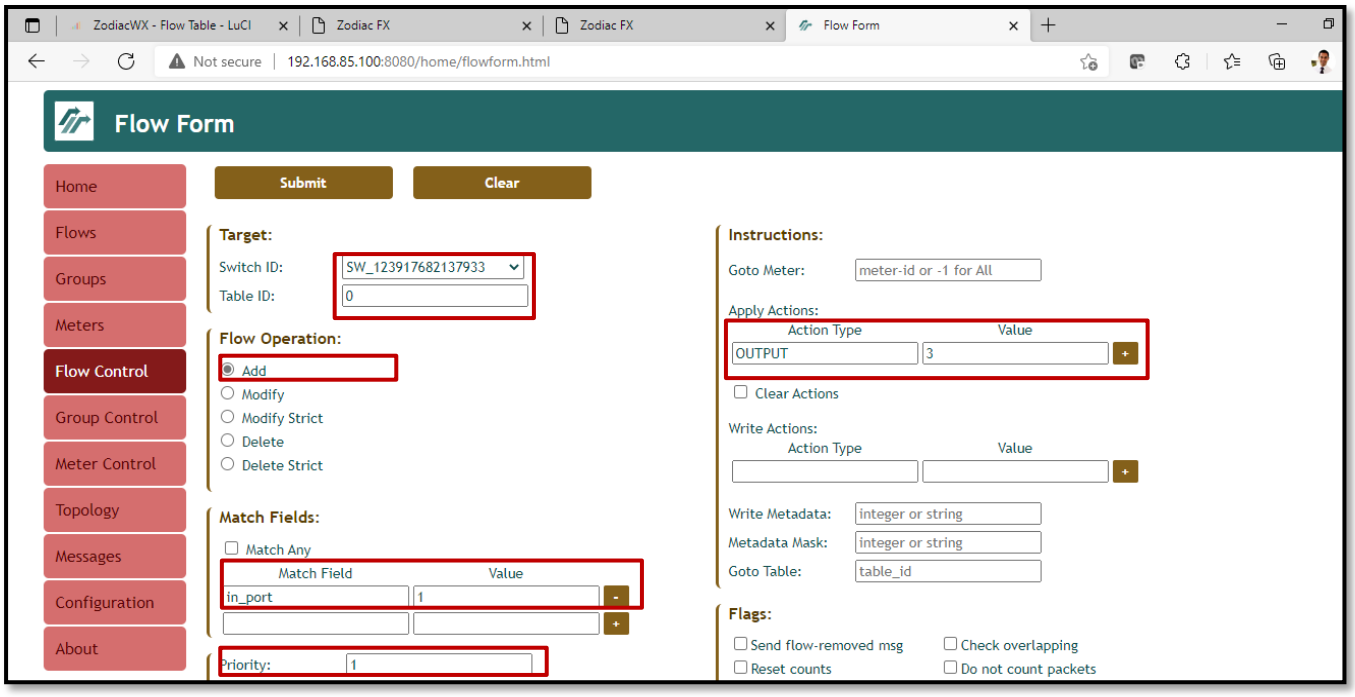


Figura 5-19 Flujos para Switch Zodiac FX-1 - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

Con la lógica diseñada y configurada en flujos SDN, es suficiente para lograr conectividad total.

El resumen de los flujos (*Flow Summary*) se observa a continuación:

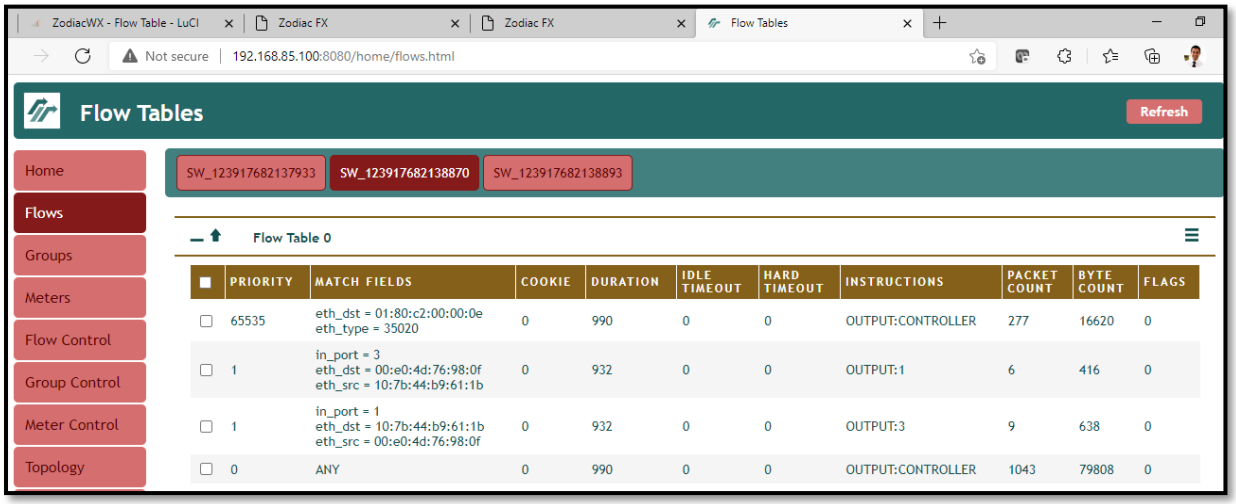


Figura 5-20 Resumen de Flujos para Switch Zodiac FX-2 - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

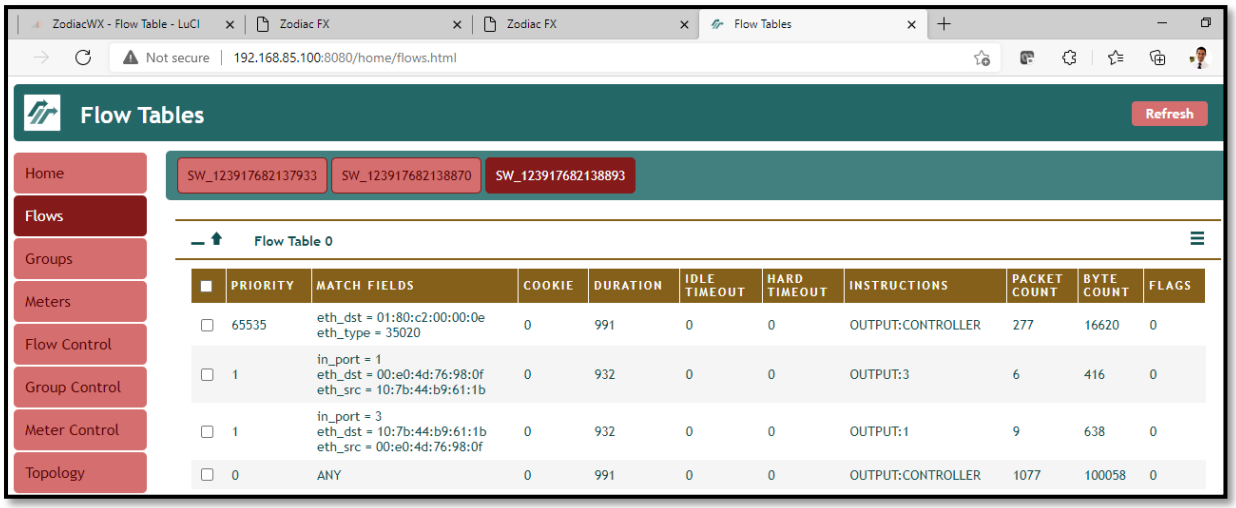


Figura 5-21 Resumen de Flujos para Switch Zodiac FX-1 - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

En la *Fig. 5-21* se observa el aprendizaje de direccionamiento MAC confirmándolo con las *Fig. 5-15* y *Fig. 5-16*. El *ping* es exitoso.

```
C:\Users\GustavoDavid>ping 192.168.85.201

Haciendo ping a 192.168.85.201 con 32 bytes de datos:
Respuesta desde 192.168.85.201: bytes=32 tiempo=7ms TTL=128
Respuesta desde 192.168.85.201: bytes=32 tiempo=8ms TTL=128
Respuesta desde 192.168.85.201: bytes=32 tiempo=8ms TTL=128
Respuesta desde 192.168.85.201: bytes=32 tiempo=7ms TTL=128

Estadísticas de ping para 192.168.85.201:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 7ms, Máximo = 8ms, Media = 7ms
```

Figura 5-22 Ping exitoso entre hosts - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

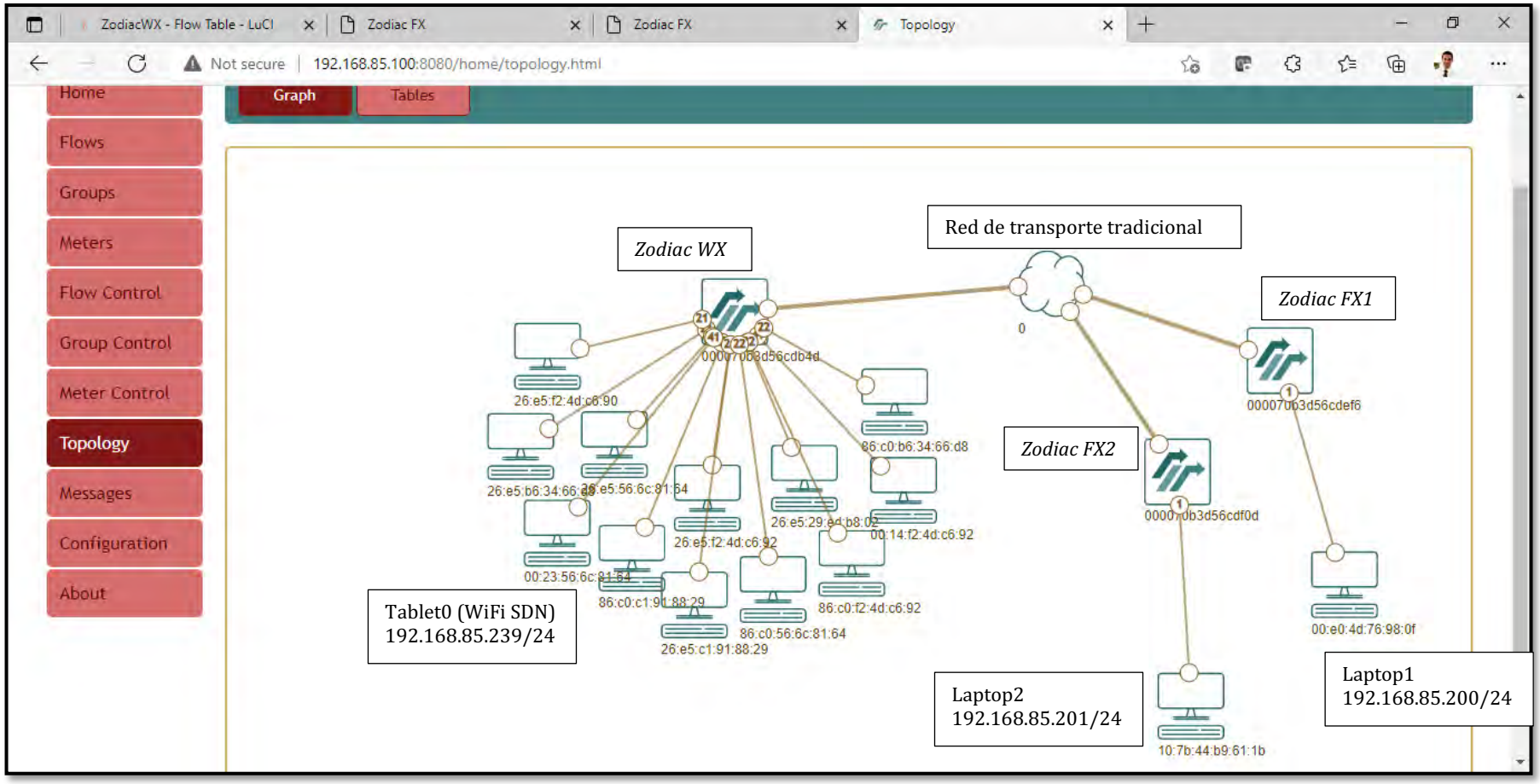


Figura 5-23 Topología visualizada en RYU FlowManager - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

En cuanto a la red *Wireless*, el equipo *Zodiac WX* tiene la capacidad de crear flujos de forma dinámica para permitir conectividad rápida en un ambiente inalámbrico. Los *hosts* inalámbricos recibirán dirección IP por DHCP también configurado en *Zodiac WX*.

En la *Fig. 5-23* se observa varios equipos inalámbricos que se conectaron en el PoC; la *Fig. 5-24* muestra la asociación de dos dispositivos, un celular y la Tablet0.

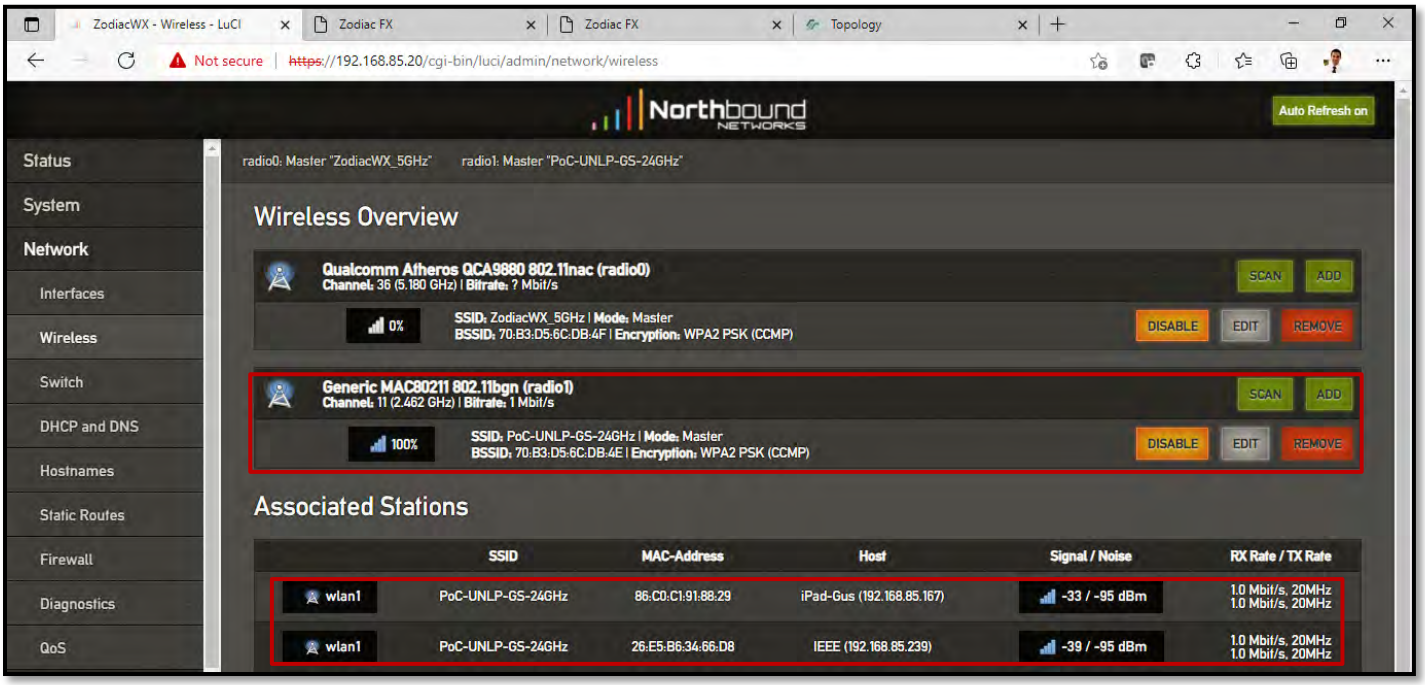


Figura 5-24 Asociación exitosa de equipos Inalámbricos - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

Para finalizar este análisis de resultados, se muestra cómo se registra en los equipos del Plano de Datos la cantidad de paquetes recibidos y transmitidos, además del reporte del estado de las interfaces y VLANs.

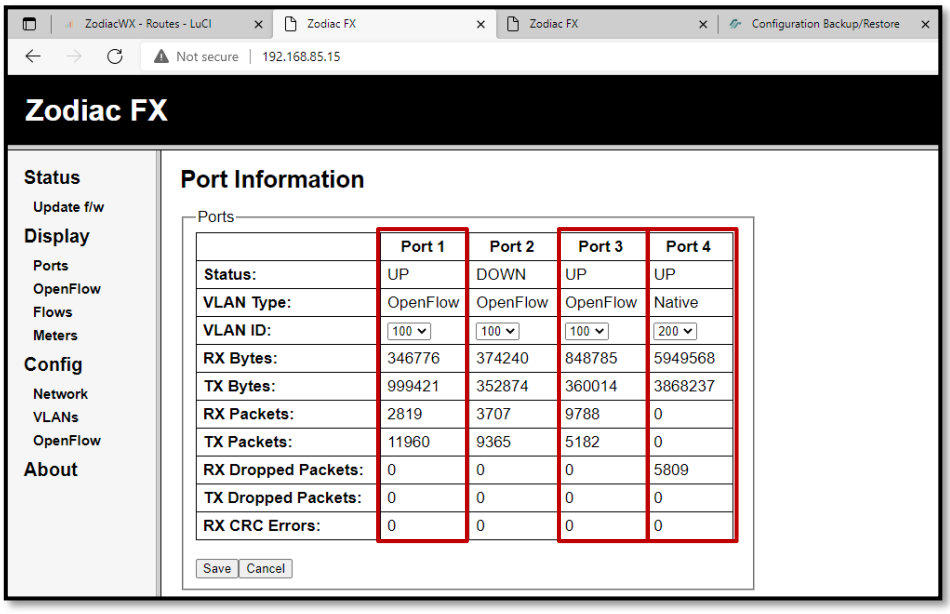


Figura 5-25 Estado de Puerto y Paquetes transmitidos - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

Capturando los mensajes enviados entre el Plano de Datos y el Controlador, se observa el intercambio de información *OpenFlow*, además de negociación TCP (puerto 6633). Los pings, así como los mensajes LLDP se encapsulan dentro de los mensajes OF_*PacketIn*.

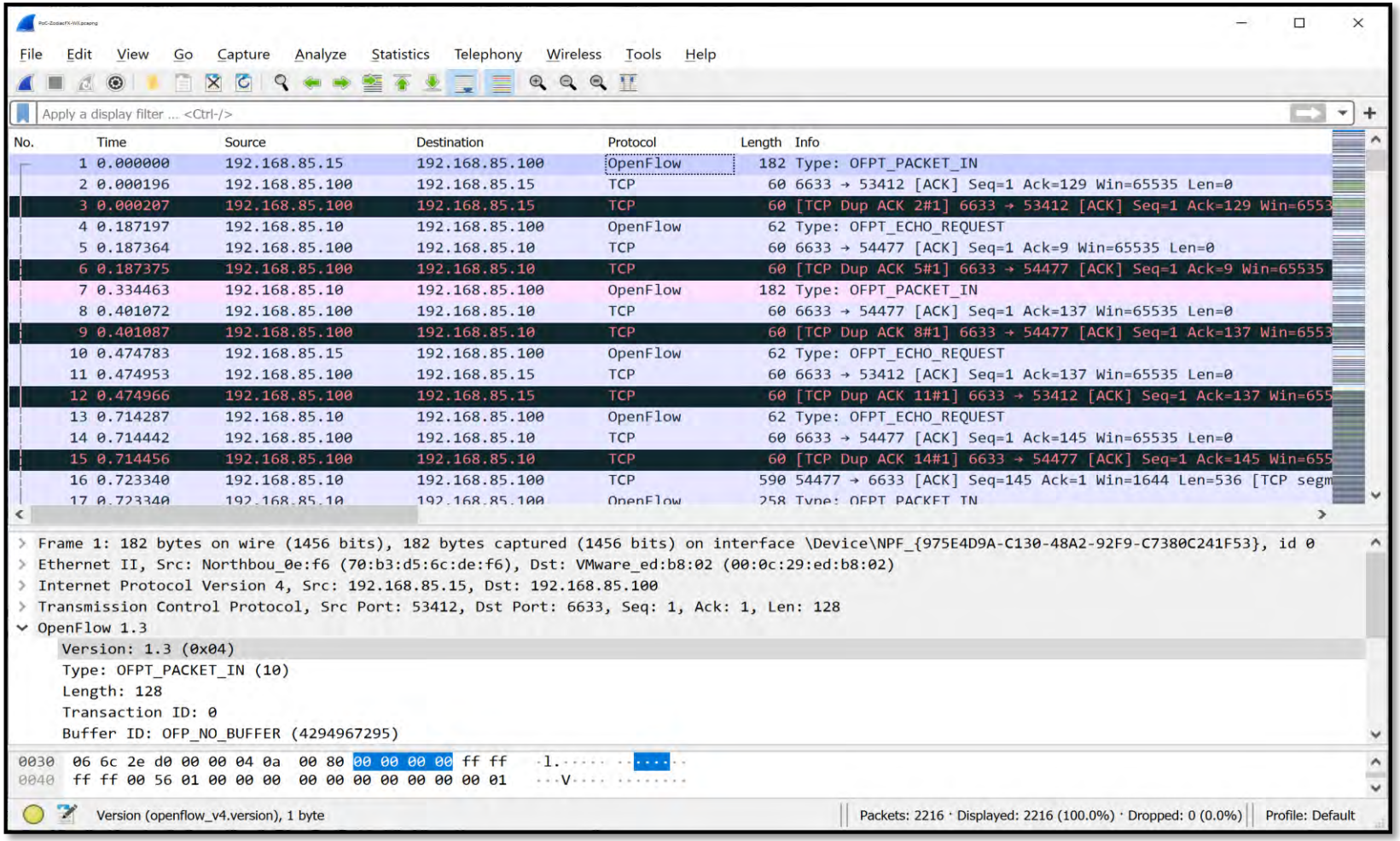


Figura 5-26 Captura de mensajes OpenFlow - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

Los flujos especificados anteriormente se muestran en el Plano de Datos dentro de *Table 0*. La segmentación en redes SDN se puede dar gracias a flujos, tablas y métricas.

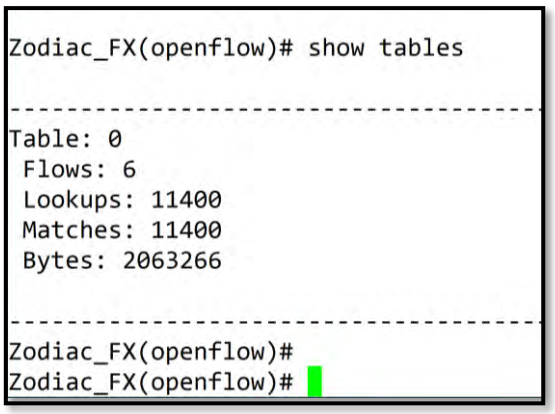


Figura 5-27 Tablas y Flujos - Red física SDN – Prueba de Conectividad
Fuente: *Autor*

Al finalizar este PoC, se comprobó la factibilidad de implementación de una red SDN en un entorno físico real, empleando protocolos estandarizados y realizando un análisis de una comunicación entre *hosts* tanto en entornos inalámbricos (IEEE 802.11n/802.11ac) como cableados (IEEE 802.3/802.1q).

Conclusiones de la Investigación

Al término de la presente tesis doctoral y en base a la experimentación realizada mediante emuladores y equipo físico, abordando tanto la parte conceptual, como empírica relacionada a la evolución de las redes de datos a través de SDN, compaginando dichos avances tecnológicos con la Transformación Digital empresarial requerida en época de pandemia COVID-19 momento en que la información se convirtió en el bien intangible más importante del siglo XXI, fue posible llegar a conclusiones relevantes que se plantean a continuación.

Durante los Capítulos 1 y 2, se estableció el estado del arte de las redes de datos, así como los requerimientos empresariales y de usuarios finales, visualizando además las falencias que como región tienen los países latinoamericanos en cuanto a conectividad a Internet se refiere, problemas acrecentados por la pandemia que impulsaron una digitalización desordenada e incompleta. Es justamente por esos motivos que contar con una red segura, inteligente, flexible, ágil y programática se convierte en un imperativo para sustentar los negocios y las actividades humanas relacionadas con tecnologías de la información de forma efectiva; sin embargo, implica mayores retos para el talento humano en el área de TI demostrado a lo largo de esta tesis, por ejemplo, necesidad de fortalecer habilidades de programación tanto en formato de datos (JSON, YAML, XML), REST-APIs, modelos tipo YANG y protocolos de administración de nodos de red modernos como NETCONF/RESTCONF, *OpenConfig*, todos siguiendo un ciclo de vida de desarrollo tipo SDLC (*Systems/Software Development Life* Cycle) basados en colaboración, integración continua, control de versiones, criterios de seguridad, agilidad y alta disponibilidad para controlar-gestionar la infraestructura subyacente, convirtiéndola en IaC (*Infrastructure-as-Code*) con código idempotente, limpio (*clean-code*), modular, reusable y sencillo.

Luego de un proceso de investigación y posterior realización de pruebas de concepto bajo el principio de Desarrollo basado en Pruebas (*Test-Driven Development*), se identificaron los elementos clave para adaptar las redes tradicionales a la Era de la Programabilidad mediante un modelo de innovación propuesto:

- Redes seguras de extremo a extremo bajo el paradigma *Underlay-Overlay*, utilizando mecanismos de cifrado y autenticación robustos, con capacidad de conexión *multicloud* y entrega de servicios virtualizados. Este concepto crea la red híbrida IP-SDN, en el que las redes tradicionales y SDN coexistirán de forma armónica e interoperable.
- Capacidad de orquestación, analítica y monitoreo (*insights)* a través de APIs en infraestructura abierta y *OpenNetworking* que reduzcan costos en CAPEX, OPEX y fomenten un pronto retorno de inversión (efectividad en las redes de datos).
- Uso de *frameworks* multiplataforma que faciliten la automatización de los procesos de redes, configuración y resolución de problemas (Programabilidad basada en Modelos), dando control de versiones de código y agilidad de implementación. El uso de patrones de diseño claros y estandarizados solucionan problemas comunes siempre presentes en el desarrollo del *software* controlador de la red.
- Cambio cultural, empresarial, laboral, económico y tecnológico enfocado al uso de metodologías ágiles, *NetDevOps* y Transformación Digital coherente con la época de pandemia COVID-19.

En los Capítulos 2 y 3, se sentaron las bases teóricas de SDN/*OpenSDN* según el modelo anteriormente planteado y bajo la premisa de *Separation of Concerns* (SoC) según el modelo MVC *(Model-View-Controller*), estudiando la estandarización, encapsulación y mecanismos de comunicación de *OpenFlow,* además comparándolo con tecnologías como NFV, concluyendo que, a pesar de tener diferentes alcances y casos de uso, funcionan bien integrándolos en un ecosistema SDN de tipi CI/CD (Integración Continua / Implementación Continua).

Se analizó de igual manera a los principales controladores SDN al momento de la escritura de esta tesis con el fin de determinar a los mejores, llegando a la conclusión que ODL, RYU y ONOS tienen el mejor desempeño según los KPIs planteados (*throughput*, latencia, *jitter*, modularidad, soporte y capacidad de monitoreo) y pruebas de estrés (tasa de ráfaga). Cabe recalcar que los controladores desarrollados en C/Java como Mul, LibFluid y Maestro son más reactivos en entornos de estrés, pero menos adaptables a mejoras futuras, mientras los más modernos como *OpenDayLight,* su continua evolución y cambios en paqueterías, han complicado su integración con infraestructuras tradicionales.

Al finalizar el Capítulo 3, se estudió las distintas formas que puede tomar SDN en una infraestructura real corporativa: *SD-Access*, *SD-DC* y *SD-WAN*, manteniendo el precepto de desacoplamiento del Plano de Control del de Datos con el fin de mejorar la respuesta de redes de datos masivos y larga distancia a través del uso de protocolos de transporte/enrutamiento de próxima generación como LISP, VXLAN, *Segment-Routing* y OMP. Su factibilidad de implementación e interoperabilidad con redes tradicionales fue comprobada con éxito en los PoCs establecidos en el Capítulo 4, donde a través de la realización de pruebas de concepto se reafirmó la efectividad del paradigma *Underlay-Overlay* y de la programabilidad extrema incluso en el *hardware* (plano de datos) con equipos abiertos tipo cajas blancas (*White boxes*) que pueden funcionar como distintos equipos bajo demanda en un concepto denominado NG-SDN.

La evolución hacia el ecosistema SDN será paulatino, aún nos encontramos en un momento de adaptación, es así que, el éxito de migrar a un entorno SD-WAN se medirá en su nivel de integración con redes tradicionales. OMP en SD-WAN Viptela es capaz de ser el puente entre las redes LAN y el *fabric* de SD-WAN, lo cual fue comprobado en los PoCs respectivos.

Finalmente, la factibilidad de implementación de *OpenSDN* (*OpenFlow*) en entornos reales mediante controlador RYU-*FlowManager* y Aruba VAN SDN, se corroboró con resultados prometedores en el Capítulo 5, tanto en desempeño como facilidad de uso, lo que sin duda, deja un precedente, SDN no sólo sirve en entornos masivos de gran escala, también en entornos empresariales PyME-SMB donde los recursos de conectividad y procesamiento de datos son escasos, pero se requiere las mismas características básicas de una red bien diseñada empresarial: Escalabilidad, Seguridad, Calidad de Servicio y Resiliencia, sumando a ellas las ventajas de las Redes Definidas por *Software*: mejor desempeño, programabilidad y posibilidad de automatización para disminuir los tiempos de configuración y solución de problemas, además de reducir considerablemente el gasto operativo y de gestión de la red.

Estudios Futuros

Las soluciones *multicloud* son el presente y futuro del *networking.* Ese concepto se planteó un par de años atrás mediante implementaciones con *clouds* híbridas, donde los procesos y servicios migran a la nube no solo de un proveedor en particular, sino con una mezcla de ellos (AWS, *Microsoft Azure, Google Cloud Platform, iCloud,* etc.), soluciones que provocaron la aparición de *Shadow IT*, término que hace referencia a la consumerización de aplicaciones, servicios y *hardware/software* (infraestructura) por departamentos que no necesariamente son los de Tecnologías de la Información empresarial gracias a la rápida adopción de la Transformación Digital en todo el contexto corporativo.

La consecuencia de ello se plasma en problemas de seguridad, malas inversiones e ineficiencias en términos de colaboración. Las organizaciones pierden visibilidad total y capacidad de control. SDN, es la tecnología ideal para cubrir esas deficiencias desde el lado de la infraestructura, por lo que es un tópico en evolución constante.

La primera propuesta de estudio futuro es el análisis e inclusión de aplicaciones basadas en Inteligencia Artificial y programabilidad intuitiva en infraestructuras SDN. Grandes empresas de telecomunicaciones están empezando este desarrollo, nombrando a *ThousandEyes*[73] como referente, permitiendo contar con *insights*, monitoreo y control sobre todo lo que ocurre en la red, que además apalanca la posibilidad de visibilidad desde la LAN, WAN y trabajo remoto. Otra plataforma abierta, creada en 2017 para procesos de solución de problemas se denominó *PyATS* (*Python Automated Test System*)[74], la cual recolecta información y lleva a la telemetría de red a otro nivel, pues dentro de sus casos de uso, ha sido empleada en ambientes TAC (*Technical Assistance Center*) para resolver errores de configuración de forma rápida.

De la mano de esta solución, está otorgar un acceso seguro a la red, más aún por el IoT masivo y dispositivos inalámbricos, parte de la hiperconectividad. El siguiente punto planteado es la necesidad de mantener entornos con niveles de riesgo de ciberseguridad bajo. Una solución por estudiar es la denominada SASE[75], siglas de *Secure Access Service Edge*, servicio que permite mediante funcionalidades de *Gateway* seguro (SWG), agentes de seguridad de acceso a la nube (CASB), empleo de *Zero Trust* y equipos como *Firewalls-as-a-Service,* DNS-*Layer* Security y protección anti-*malware*, lograr una integración a SD-WAN y así obtener un entorno adecuado para la conectividad empresarial. Cisco implementó esta solución con la premisa de Conectar, Controlar y Converger. Muchos fabricantes de telecomunicaciones nombran estos sistemas como *Network Security-as-a-Service*.

El punto faltante como análisis a futuro, es la famosa colocación de infraestructura (*CoLocation*). Esta característica permitirá dotar de flexibilidad y simplicidad a SD-WAN para que conecte múltiples sedes distribuidas sobre varias regiones mediante un concepto denominado *Virtual Hubs* según ubicación geográfica, nuevos PoPs (*Points-of-Presence*) para SD-WAN. Los beneficios de esta tecnología son:

[73] ThousandEyes: https://www.thousandeyes.com/
[74] Pyats: https://xrdocs.io/programmability/tutorials/pyats-series-collecting-many-show-commands/
[75] SASE: https://www.cloudflare.com/lp/ppc/sase-x/?&_bt=477215087405&_bk=sase&_bm=e&_bn=g&_bg=113413743980&_placement=&_target=&_loc=9069516&_dv=c&awsearchcpc=1&gclid=Cj0KCQjw6s2IBhCnARIsAP8RfAgvCTRfeR1lyT2ifugM3Lf2sbdTd2qTZch9m7Mk_eGzer5VPSNh_WkaAmmFEALw_wcB&gclsrc=aw.ds

- Creación de políticas que mejoran el QoE (*Quality of Experience*) con seguridad granular.
- Interoperabilidad con redes tradicionales que no conocen el concepto de SD-WAN.
- Multinacionales con capacidad de gestión de acceso a Internet/Servicios en la nube con criterios de privacidad, autenticación y autorización.
- Teletrabajadores que requieren de conexión a través de VPNs inteligentes sobre enlaces de acceso a Internet de bajo costo.

Soluciones como *Cloud-onRamp* propuesto por fabricantes y *vendors* de tecnología, compagina con soluciones tipo SD-WAN, equipos virtualizados y VNFs (*Virtual Network Functions*) al crear túneles seguros hacia aplicaciones SaaS, servicios *multicloud* y DCs.

Los estudios futuros permiten entender las soluciones tecnológicas del presente y de los próximos años, pero para llegar a ese punto, es necesario clarificar los requerimientos solicitados en el *networking* corporativo moderno:

- Red flexible, creada bajo demanda, aprovisionando clientes y usuarios en una red confiable, segura, resiliente, de buenas prestaciones y que apalanque los objetivos empresariales, generando telemetría y diagnóstico a distancia.
- Disminuir la complejidad intrínseca de las redes hacia un camino automatizado, programable y abierto, donde las redes híbridas IP-SDN coexistirán.
- Contar con una seguridad robusta e inteligente de extremo a extremo, la cual integra control de la red, capas de visibilidad, identificación de roles y monitoreo.

Acrónimos

A

AAA	Authentication, Authorization and Accounting
ACK	Acknowledgement
AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol
AP	Access Point (Wireless)
API	Application Programming Interface
AR	Augmented Reality. Realidad Aumentada
ASIC	Application-Specific Integrated Circuit
ASF	Atmel Software Framework (Zodiac FX)
AWS	Amazon Web Services

B

BASH	Bourne Again Shell
BDF	Bidirectional Forwarding Detection
BGP	Border Gateway Protocol
BGP-LS	BGP – Link State
BoD	Board of Directors. Alta Gerencia. Mesa Directiva
BSS (ONF)	Business Support System
BYOX	Bring Your Own Everything. Trae tu propio cualquier elemento (y conéctalo a la red)

C

CAPEX	Capital Expenditure. Gasto de Capital
CAPWAP	Control and Provisioning of Wireless Access Points
CD	Continuous Delivery
CEF	Cisco Express Forwarding
CEO	Chief Executive Officer.
CI	Continuous Integration
CLI	Command Line Interface
CoAP	Constrained Application Protocol
CSP	Communications Service Provider
cURL	Cliente URL

D

D2D	Device to Device. Comunicación Dispositivo a Dispositivo
DC	Data Center
DevOps	Development Operations
DMVPN	Dynamic Multipoint Virtual Private Network
DNA (Cisco)	Cisco Digital Network Architecture
DSL (Puppet)	Domain Specific Language (Puppet DSL)
DTLS	Datagram Transport Layer Security

E

ECMP	Equal-Cost MultiPath
EID	Endpoint Identifier (LISP)
ETSI	European Telecommunications Standards Institute
EVPN	Ethernet Virtual Private Network

F

FCAPS	Fault, Configuration, Accounting, Performance and Security
FcoE	Fibre Channel over Ethernet
FIB	Forwarding Information Base
FRR	Free Range Routing

G

GIT	Software de control de versiones creado por Linus Torvalds
GUI	Graphical User Interface

H

HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol

I

IaaC/IaC	Infrastructure as a Code
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IKE	Internet Key Exchange
Inet	Intelligent Network
IoT	Internet of Things. Internet de las Cosas
IP	Internet Protocol 300ocker300300 4 y 6 (Ipv4 – Ipv6)
ISAKMP	Internet Security Association and Key Management Protocol
ISOC	Internet Society
ISP	Internet Service Provider
ITIL	Information Technology Infrastructure Library
ITU	International Telecommunication Union
iWAN	Intelligent Wide Area Network

J

J2	Jinja 2
JSON	Javascript Object Notation

L

LAN	Local Area Network.

LDAP	Lightweight Directory Access Protocol
LFP	The Linux Foundation Projects
LFN	The Linux Foundation Networking
LISP	Locator ID Separation Protocol
LSRR	Loose Source and Record Route
LTE	Long Term Evolution
LWAPP	Lightweight Access Point Protocol

M

M2M	Machine to Machine. Conectividad Máquina a Máquina.
MD-SAL	Model-Driven Service Abstraction Layer
ML	Machine Learning
MPLS	Multiprotocol Label Switching
MQTT	Message Queuing Telemetry Transport

N

NAPALM	Network Automation and Programmability Abstraction Layer with Multivendor Support
NDP	Neighbor Discovery Protocol
NETCONF	Network Configuration Protocol
NFV	Network Functions Virtualization
NGFW	Next-Generation Firewall
NOS	Network Operation System
NPB	Network Packet Broker

O

OCP	Open Compute Project
ODL	OpenDayLight
OMP	Overlay Management Protocol
ONF	Open Networking Foundation
ONOS	Open Network Operating System
ONU	Organización de las Naciones Unidas
OOP	Object-Oriented Programming
OPEX	Operational Expenditures. Gasto Operacional
OPNFV	Open Platform for Network Functions Virtualization
OSI	Open System Interconnection
OSS	Operation Support System
OVN	Open Virtual Network
OVS	Open Virtual Switch (Nicira – VMWare)

P

P4	Programming Protocol-independent Packet Processors
PaaS	Platform as a Service
PCEP	Path Computation Element Protocol

PIP	Preferred Installer Program – Pip Install Python
PISA	Protocol Independent Switch Architecture
PPDIOO	Prepare, Plan, Design, Implement, Operate and Optimize Lifecicle
PoCs	Proof-of-Concepts
POM	Project Object Model
PyATS	Python Automated Test Systems

Q

QoS	Quality of Service. Calidad de Servicio

R

RADIUS	Remote Authentication Dial-In User Service
REST	Representational State Transfer
RFID	Radio Frequency Identifier
RIB	Routing Information Base
RLOC	Routing/Record Locator (LISP)
ROI	Return on Investment. Retorno de Inversión
RPC	Remote Procedure Call

S

S3P	Security, Scalability and Performance (ODL)
SASE	Secure Access Service Edge
SaaS	Software as a Service
SDN	Software Defined Networking. Redes Definidas por Software
SD-Access	Software Defined Access
SD-DC	Software Defined Data Center
SDK	Software Development Kit
SD-WAN	Software Defined Wide Area Network
SecOps	Security Operations
SEN	Secure Extensible Network – SDWAN Viptela
SGT	Scalable Group Tags
SID	Segment Identifier (Segment Routing)
SLA	Service Level Agreement
SMB	Small-Medium Business
SNMP	Simple Network Management Protocol
SNV	Service Node Functionality for VXLAN
SOAP	Simple Object Access Protocol
SoC	Separation of Concerns
SRGB	Segment Routing Global Block
SRH	Segment Routing Header
SSH	Secure Shell
SSID	Service Set Identifier
SSRR	Strict Source and Record Route

T

TAC	Technical Assistance Center
TCP	Transmission Control Protocol
TCO	Total Cost of Ownership. Costo total de Propiedad
TDD	Test-Driven Development
TDM	Time Division Multiplexing
TLOC	Transport Locations Routes (OMP SD-WAN)
TLS	Transport Layer Security

U

UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name

V

VCS	Version Control System
VPN	Virtual Private Network
VR	Virtual Reality. Realidad Virtual
VXLAN	Virtual Extensible Local Area Network
VXRD	VXLAN Registration Daemon

W

WAN	Wide Area Network
WEF	World Economic Forum
WiFi6 – WiFi6E	Wireless Fidelity 6. IEEE 802.11ax – WiFi6E (6GHz)
WLC	Wireless LAN Controller
WSGI	Web Server Gateway Interface (RYU SDN Controller)

X

XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Y

YAML	Yet Another Markup Language – Ain't Markup Language
YANG	Yet Another Next Generation

Z

ZB	Zettabyte
ZTP	Zero-Touch Provisioning

Bibliografía

ABI Research. (2013, Mayo 09). *More Than 30 Billion Devices Will Wirelessly Connect to the Internet of Everything in 2020.* Retrieved from https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne/

Afaq, M. (2018, Junio 18). *Understanding and Building an end-to-end NetDevOps Pipeline*. Retrieved from https://www.fullstacknetworker.com/understanding-and-building-an-end-to-end-netdevops-cicd-pipeline/

Álvarez, M. (2020, Agosto 5). *NAPALM Network Automation Python: Introduction e Installation*. Retrieved from https://codingnetworks.blog/napalm-network-automation-python-introduction-e-installation/

Andrade-Salinas, G., Salazar-Chacon, G., & Vintimilla, L.M. "Integration of IoT Equipment as Transactional Endorsing Peers over a Hyperledger-Fabric Blockchain Network: Feasibility Study", *International Conference on Applied Technologies*, pp. 95-109, 2019, December

Apostolopoulos, J. (2018, Junio 1). *Why is intent-based networking good news for software-defined networking?* Retrieved from https://blogs.cisco.com/analytics-automation/why-is-intent-based-networking-good-news-for-software-defined-networking

Arista Networks. (2021, Abril 4). *MPLS Segment Routing*. Retrieved from https://www.arista.com/en/solutions/mpls-segment-routing

Aruba - A Hewlett Packard Enterprise Company. (n.d.). *¿Qué es la arquitectura "spine-leaf"?* Retrieved from https://www.arubanetworks.com/es/arquitectura-spine-leaf/

Aruba Networks. (2017). *Aruba VAN SDN Controller Software - Datasheet.* Retrieved from https://tdhpe.techdata.eu/Documents/Aruba/DS_VAN_SDN.pdf?epslanguage=en

Bachar, Y. -F. (2015, Febrero 11). *Introducing "6-pack": the first open hardware modular switch*. Retrieved from https://code.fb.com/production-engineering/introducing-6-pack-the-first-open-hardware-modular-switch/

Badotra, S., & Narayan Panda, S. (2019, Septiembre 30). Evaluation and comparison of OpenDayLight and open networking operating system in software-defined networking. *Cluster Computing - Springer Science+Business*. doi:10.1007/s10586-019-02996-0

Barton, B. (2017, Noviembre). *Understanding Cisco's Next Generation SD-WAN Technology.* Retrieved from https://www.slideshare.net/CiscoCanada/cisco-connect-vancouver-2017-understanding-cisco-next-gen-sdwan

Bednarz, A. (2016, Noviembre 18). *Network outages linked to human error, incompatible changes, greater complexity*. Retrieved from https://www.networkworld.com/article/3142838/top-reasons-for-network-downtime.html

Bitcoin. (2020). *Bitcoin*. Retrieved from https://bitcoin.org/es/

Boorsma, B. (2016, Septiembre 26). *Digitization: Drivers, Trends and the Internet of Things.* Retrieved from https://www.slideshare.net/BigDataExpo/cisco-bas-boorsma

Bruno, A., & Jordan, S. (2011). *Cisco CCDA Cert Guida: Data Center Design.* Indianapolis, USA: Cisco Press.

CAF - Banco de Desarrollo de América Latina. (2020, abril 07). *COVID-19: ¿Cuál es el estado de la digitalización de América Latina para la resiliencia social, económica y productiva?* Retrieved from https://www.caf.com/es/actualidad/noticias/2020/04/covid-19-cual-es-el-estado-de-la-digitalizacion-de-america-latina-para-la-resiliencia-social-economica-y-productiva/

Capone, A., & Cascone, C. (2015, Abril 13). *From dumb to smarter switches in software defined networks: An overview of data plane evolution.* Retrieved from https://www.slideshare.net/commtechpolimi/tutorial-on-sdn-data-plane-evolution

Cascone, C. (2019, Septiembre 8). *Open Networking Foundation - Topology NG-SDN Tutorial.* Retrieved from https://github.com/opennetworkinglab/ngsdn-tutorial/blob/master/mininet/topo.py

Cascone, C., & Condon, S. (2020, Mayo 5). *NG-SDN Tutorial - Docker-Compose.yml.* Retrieved from https://github.com/opennetworkinglab/ngsdn-tutorial/blob/master/docker-compose.yml

Chaloupka, J. (2017, Agosto). *Network Architecture with Software Programmability - Segment Routing.* Retrieved from https://slideplayer.com/slide/13426422/

Cisco DevNet - Programming Fundamentals. (2020). *Data Formats: Understanding and using JSON, XML and YAML*.

Cisco Meraki SD-WAN. (2019). *Discover Meraki SD-WAN*. Retrieved from The world's most trusted SD-WAN provider: https://meraki.cisco.com/products/appliances#sd-wan

Cisco Netacad. (2020, julio). *Redes empresariales, Seguridad y Automatización*.

Cisco NetAcad DevNet Associate Course. (2020, Julio). *Automating Infrastructure with Cisco*. Retrieved from www.netacad.com

Cisco Systems - Cisco Umbrella. (n.d.). *Point DNS to Cisco Umbrella.* Retrieved from https://docs.umbrella.com/mssp-deployment/docs/point-dns-to-cisco-umbrella

Cisco Systems - DevNet Learning Labs. (2021, junio 1). *Introduction to SDWAN REST APIs.* Retrieved from https://developer.cisco.com/learning/tracks/sd-wan_programmability/sd-wan/intro-sd-wan-rest-api/step/2

Cisco Systems - Senior VicePresident Liz Centoni. (2019). *CiscoLive Barcelona 2019*. Retrieved from IoT Keynote: https://video.cisco.com/detail/videos/events/video/6009932373001/ciscolive-barcelona-2019-iot-keynote?autoStart=true

Cisco Systems - User Defined Network. (2020, junio). *Cisco User Defined Network: Provide control and security to your users*. Retrieved from https://www.cisco.com/c/en/us/solutions/enterprise-networks/user-defined-network.html

Cisco Systems. (2018, Junio 15). *Cisco Wireless Controller Configuration Guide, Release 8.3.* Retrieved from https://www.cisco.com/c/en/us/td/docs/wireless/controller/8-3/config-guide/b_cg83/b_cg83_chapter_0100111.html

Cisco Systems. (2018, Enero 20). *IP Routing: LISP Configuration Guide, Cisco IOS XE Release 3S.* Retrieved from https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_lisp/configuration/xe-3s/irl-xe-3s-book/irl-overview.html

Cisco Systems. (2018). *Transforming Businesses with Artificial Intelligence.* Retrieved from https://www.cisco.com/c/dam/en/us/solutions/collateral/digital-transformation/ai-whitepaper.pdf?ccid=cc000098&dtid=esofbk000262&CAMPAIGN=NB-06+Network+Buyer+Program&Country_Site=us&POSITION=Social%2BMedia&REFERRING_SITE=Facebook&CREATIVE=Cisco+Enterprise+

Cisco Systems. (2019, mayo 21). *Cisco Academy Conference Latam 2019*. Retrieved from http://www.academyconferencelatam.com/

Cisco Systems. (2019, mayo 9). *Cisco SD-WAN Deployment Guide.* Retrieved from https://www.cisco.com/c/en/us/td/docs/solutions/CVD/SDWAN/CVD-SD-WAN-Deployment-2019APR.html?ccid=cc000954&dtid=esofbk000262&CAMPAIGN=NB-06+Network+Buyer+Program&Country_Site=us&POSITION=Social%2BMedia&REFERRING_SITE=Facebook&CREATIVE=Cisco+Enterprise+Net

Cisco Systems. (2019, febrero 27). *Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper*. Retrieved from https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

Cisco Systems. (2019, agosto). *Evolución Digital: Desde Mainframe al Multicloud y más allá.* Retrieved from https://engage2demand.cisco.com/lp_servidor_17617?CCID=cc001060&DTID=odicdc000016&OID=pordc018464

Cisco Systems. (2019, julio 7). *Intent-Based Networking*. Retrieved from The network platform for IT and business innovation: https://www.cisco.com/c/en_be/solutions/intent-based-networking.html

Cisco Systems. (2020). *Software-Defined Networking - Automate and program your network faster*. Retrieved from https://www.cisco.com/c/en/us/solutions/software-defined-networking/overview.html

Cisco Systems. (n.d.). *Cisco SD-WAN Dashboard.* Retrieved from https://sdwan-docs.cisco.com/Product_Documentation/vManage_Help/Release_18.2/Dashboard/Dashboard

Citakovic, R. (2019, Marzo 19). *Introduction to Ansible.* Retrieved from https://www.cisco.com/c/dam/m/sr_rs/events/2019/cisco-connect/pdf/using_ansible_in_dc_automation_radenko_citakovic.pdf

Citrix. (2019). *What is software-defined networking (SDN)?* Retrieved from https://lac.citrix.com/glossary/what-is-software-defined-networking.html

Clark, S. (2019, Julio 16). *Network Automation Using Unified API – Napalm*. Retrieved from https://blogs.cisco.com/developer/network-automation-using-napalm

Cloudfare. (2021, abril 14). *Zero Trust Network Access (ZTNA)*. Retrieved from https://www.cloudflare.com/es-la/teams/zero-trust-network-access/

CodiLime. (2020, Septiembre 20). *Introduction to network programming with P4*. Retrieved from https://www.youtube.com/watch?v=UEMAvXXNWsY

Coppola, F. (2019, junio 30). *The Real Threat From Facebook's Libra Coin - Forbes*. Retrieved from https://www.forbes.com/sites/francescoppola/2019/06/30/the-real-threat-from-facebooks-libra-coin/#138867171dc5

Curran, D. (2012, Diciembre 4). *Under the Hood: Open vSwitch & OpenFlow in XCP & XenServer - CloudStack Collaboration Conference.* Retrieved from https://www.slideshare.net/xen_com_mgr/under-the-hood-open-vswitch-openflow-in-xcp-xenserver

Data Center Frontier. (2018, Abril 16). *Evolution IT Infrastructures*. Retrieved from https://datacenterfrontier.com/the-top-5-it-strategies-for-the-data-center/evolution-it-infrastructure-2/

Davidson, J. (2017, Septiembre 27). *Simplifying Networks through Segment Routing*. Retrieved from https://blogs.cisco.com/news/simplifying-networks-through-segment-routing

Debois, P. (2008). Agile infrastructure and operations: how infra-gile are you? *IEEE Agile 2008 Conference*, 202-207. doi:10.1109/Agile.2008.42

Dell'Oro Group. (2020, Marzo 3). *Cisco, Silver Peak, Versa, VMWare, and Fortinet Were 2019 Top Five Vendors - SDWAN*. Retrieved from https://www.delloro.com/news/sd-wan-market-increased-64-percent-and-surged-over-1-billion-in-2019/

Erdem, K. (2019, Abril 17). *DevOps Turkiye*. Retrieved from https://medium.com/devopsturkiye/teknolojileri-ile-hayat-kurtaran-32-devops-arac%C4%B1-4eb35b234c88

Filsfils, C. (2016, Junio 23). *Segment Routing Overview with Clarence Filsfils*. Retrieved from https://www.youtube.com/watch?v=ZpUnsYjMBEA

Filsfils, C., Michielsen, K., & Talaulikar, K. (2017). *Segment Routing Part I.* California: CreateSpace Independent Publishing Platform; 1er edición (17 Enero 2017).

Forbes. (2017, Diciembre 28). *3 Key Machine Learning Trends To Watch Out For In 2018*. Retrieved from https://www.forbes.com/sites/janakirammsv/2017/12/28/3-key-machine-learning-trends-to-watch-out-for-in-2018/#1d2429801280

Fortinet. (2021, marzo). *SD-WAN Vendors List: Comparison of Top Providers*. Retrieved from https://www.fortinet.com/lat/products/sd-wan-providers

Gaurav, S. (2017, Noviembre 12). *Is DevOps Agile? - DZone*. Retrieved from https://dzone.com/articles/is-devops-agile?platform=hootsuite

Global Market Insights. (2020, Mayo). *Software-Defined Wide Area Network (SD-WAN) Market Size*. Retrieved from https://www.gminsights.com/industry-analysis/software-defined-wide-area-network-sdwan-market

GNS3 Marketplace. (2017, Junio 29). *Network Automation Appliance - Download*. Retrieved from https://gns3.com/initiatives/network-automation

Gómez, O. (2017, Junio 27). *Transformación Digital, ¿Evolución o Mutación?* Retrieved from https://gblogs.cisco.com/la/dg-oscargomez-transformacion-digital-evolucion-o-mutacion/

Gomolski, B. (2019, enero 18). *Driving Cost Optimization Across the Enterprise: An Executive Perspective.* Retrieved from Gartnet: https://www.gartner.com/en/doc/3898566-driving-cost-optimization-across-the-enterprise-an-executive-perspective

Gooley, J., Hasan, R., & Vemula, S. (2021). *Cisco Software-Defined Access.* Hoboken, NJ: Cisco Press.

Gooley, J., Yanch, D., Schuemann, D., & Curran, J. (2021). *Software-Defined Wide-Area Networks.* Cisco Press.

Guillermo, J. (2017, Septiembre 5). *Demystifying Software-defined Networks Part I: Open SDN Approach - Dell Technologies*. Retrieved from https://infocus.delltechnologies.com/javier_guillermo/demystifying-software-defined-networking-open-sdn-approach/

Guis, I. (2012, Noviembre 15). *The SDN Gold Rush To The Northbound API - SDX Central.* Retrieved from https://www.sdxcentral.com/articles/contributed/the-sdn-gold-rush-to-the-northbound-api/2012/11/

H3C. (2019). *The Next Generation Network Architecture with H3C's SDN, NFV & Overlay*. Retrieved from http://www.h3c.com.hk/Products___Solutions/Home/SDN_NFV_Overlay/

HabitatIII. (2016, Octubre 17). *Reporte Conferencia HabitatIII*. Retrieved from http://habitat3.org/documents-and-archive/final-reports/the-conference-report/

Hanes et al. (2017). *IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things.* Indianapolis, IN 46240 USA: Cisco Press.

Hill, C., Miller, D., Zacks, D., Suhr, J., Kumar, K., Karmarkar, K., . . . otros. (2019, Abril). *Software-Defined Access - Enabling Intent-Based Networking.* Retrieved from https://www.cisco.com/c/dam/en/us/products/se/2018/1/Collateral/nb-06-software-defined-access-ebook-en.pdf

Hinchcliffe, D. (2018, Diciembre 19). *The SAP platform and digital transformation*. Retrieved from https://www.zdnet.com/article/the-sap-platform-and-digital-transformation/

Huang, W. (2020, Marzo 4). *Ansible 101 Getting Started*. Retrieved from https://medium.com/@wintonjkt/ansible-101-getting-started-1daaff872b64

Huawei. (2019). *SDN Concept and Benefits*. Retrieved from https://developer.huawei.com/ict/en/site-sdn/article/01

IBM. (2020). *How much would a data breach cost your business?* Retrieved from https://www.ibm.com/security/data-breach

IBM. (2021, Abril 7). *What is a Software-Defined Data Center (SDDC)?* Retrieved from https://www.ibm.com/services/network/software-defined-data-center

IBM Developer . (2019, Febrero 20). *YAML basics in Kubernetes*. Retrieved from https://developer.ibm.com/technologies/containers/tutorials/yaml-basics-and-usage-in-kubernetes/

IETF RFC 8040. (2017, Enero). *IETF - RESTCONF Protocol.* Retrieved from https://tools.ietf.org/html/rfc8040

Infobae. (2019, julio 4). *El Congreso de EEUU pidió a Facebook que detenga el desarrollo de la criptomoneda Libra*. Retrieved from https://www.infobae.com/america/tecno/2019/07/03/el-congreso-de-eeuu-pidio-a-facebook-que-detenga-el-desarrollo-de-la-criptomoneda-libra/

Ingram Micro. (2017, Mayo 05). *5 Differences Between SDN and Network Functions Virtualization*. Retrieved from https://imaginenext.ingrammicro.com/data-center/5-differences-between-sdn-and-network-functions-virtualization

Internet Engineering Task Force - IETF. (2013, Enero). *The Locator/ID Separation Protocol (LISP) - RFC 6830*. Retrieved from https://tools.ietf.org/html/rfc6830

Internet Society. (2020). *World IPv6 Lauch Day*. Retrieved from https://www.worldipv6launch.org/

Jackson, C., Gooley, J., Iliesiu, A., & Malegaonkar, A. (2020). *Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide.* Ciscopress.

Jager, T. (2013, Julio 9). *Siemens S7-1200 Web Server Tutorial - From Getting Started to HTML5 User Defined Pages*. Retrieved from https://www.dmcinfo.com/latest-thinking/blog/id/8567/siemens-s7-1200-web-server-tutorial--from-getting-started-to-html5-user-defined-pages

Juniper Networks. (2019). *Software-Defined Networking*. Retrieved from https://www.juniper.net/us/en/products-services/sdn/

Juniper Networks. (2021, Abril 4). *What is segment routing?* Retrieved from https://www.juniper.net/us/en/products-services/what-is/segment-routing/

Kickstarter Projects. (2021, Marzo 1). *Zodiac FX: The world's smallest OpenFlow SDN switch*. Retrieved from https://www.kickstarter.com/projects/northboundnetworks/zodiac-fx-the-worlds-smallest-openflow-sdn-switch?lang=es

Logan, P. (2018, Septiembre 21). *REST, SOAP, GraphQL — Gesundheit!* Retrieved from https://medium.com/better-practices/rest-soap-graphql-gesundheit-6544053f65cf

Manville, J., Woolwine, J., & Benny, V. d. (2019). *Evolution of the Data Center: Global Cloud Strategy and Tetration.* Retrieved from Cisco Systems: https://www.cisco.com/c/dam/en_us/about/downloads/evolution-of-the-data-center-john-manville-dc-day.pdf

McCormick, J. (2016, Noviembre 2). *Predictions 2017: Artificial Intelligence Will Drive The Insights Revolution*. Retrieved from https://www.forrester.com/report/Predictions+2017+%20Artificial+Intelligence+Will+Drive+The+Insights+Revolution/-/E-RES133325

MeritStep. (2019, Febrero 12). *DevOps*. Retrieved from https://www.meritstep.com/courses/devops

Nadeau, T., & Gray, K. (2013). *SDN: Software Defined Networks.* O'Reilly.

Nefkens, P.-J. (2020). *Transforming Campus Networks to Intent-Based Networking.* Hoboken: Cisco Press.

Northbound Networks. (2017, Mayo). *Zodiac FX User Guide.* Retrieved from https://forums.northboundnetworks.com/downloads/zodiac_fx/guides/ZodiacFX_UserGuide_0517.pdf

Northbound Networks. (2017). *Zodiac WX - User Guide.* Retrieved from https://forums.northboundnetworks.com/downloads/zodiac_wx/guides/ZodiacWX_Quick_Start_Guide_1017.pdf

Northbound Networks. (2021, Julio 23). *Acerca de Northbound Networks*. Retrieved from https://northboundnetworks.com/pages/about-us

nSolve - Excellence in software design. (2019, Julio 5). *What is the nCall REST API Server module for?* Retrieved from https://www.nsolve.com/what-is-the-ncall-api/

O'Connor, B. (2018, Junio 5). *Enabling the Next Generation of SDN.* Retrieved from https://p4.org/assets/P4WS_2018/Stratum_Brian.pdf

O'Connor, B., & Cascone, C. (2019, Septiembre 10). *Tutorial Next-Gen SDN - main.p4.* Retrieved from https://github.com/opennetworkinglab/ngsdn-tutorial/blob/master/p4src/main.p4

Okasha, K. (2017, Mayo 18). *Network Automation and the Rise of NETCONF*. Retrieved from https://medium.com/@k.okasha/network-automation-and-the-rise-of-netconf-e96cc33fe28

ONF - Open Networking Foundation. (2020, Febrero 5). *2020 State of the ONF*. Retrieved from https://opennetworking.org/news-and-events/blog/2020-state-of-the-onf/

ONF - Open Networking Foundation. (2021). *NG-SDN*. Retrieved from https://opennetworking.org/ng-sdn/

ONF - Open Networking Foundation. (2021). *P4*. Retrieved from https://opennetworking.org/p4/

ONF - Open Networking Foundation. (2021). *What is Mininet?* Retrieved from https://opennetworking.org/mininet/

Open Networking Foundation - ONF. (2020). *Software-Defined Networking (SDN) Definition*. Retrieved from https://www.opennetworking.org/sdn-definition/

Open Networking Foundation - OpenFlow Switch Spec. (2013, Abril 25). *OpenFlow Switch Specification. Version 1.3.2 (Wire Protocol 0x04) .* Retrieved from https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.2.pdf

Open Networking Foundation - OpenFlow Switch Spec. (2015, Marzo 26). *OpenFlow Switch Specification Version 1.5.1.* Retrieved from https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

Open Networking Foundation. (2016, Septiembre 08). *ONF SDN Evolution.* Retrieved from http://www.opennetworking.org/wp-content/uploads/2013/05/TR-535_ONF_SDN_Evolution.pdf

OpenDayLight - Fluorine Release. (2018, Septiembre). *OpenDayLight - Fluorine Release*. Retrieved from https://www.opendaylight.org/wp-content/uploads/sites/14/2018/09/Screen-Shot-2018-09-13-at-3.15.43-PM.png

OpenDaylight - The Linux Foundation Projects. (2021). *OpenDaylight - The Linux Foundation Projects*. Retrieved from https://www.opendaylight.org/

OpenDayLight. (2018). *Use Case | Cloud and NFV*. Retrieved from https://www.opendaylight.org/use-cases-and-users/by-function/cloud-and-nfv

OpenLisp. (n.d.). *The OpenLISP Project*. Retrieved from http://www.openlisp.org/

Oracle. (2017). *Can Virtual Experiences Replace Reality?* Retrieved from The future role for humans in delivering: https://www.oracle.com/webfolder/s/delivery_production/docs/FY16h1/doc35/CXResearchVirtualExperiences.pdf

Parker, B., & Shawn, F. (2018, Noviembre 7). *IDC FutureScape: Worldwide Digital Transformation (DX).* Retrieved from https://www.ibm.com/downloads/cas/RGOQMOY1

Peñaloza, D. (2018, Mayo 1). *Introducción a Segment Routing*. Retrieved from https://learningnetwork.cisco.com/s/blogs/a0D3i000002SKD4EAO/introducci%C3%B3n-a-segment-routing

Pepelnjak, I. (2013, Agosto 13). *Management, Control and Data Planes in Network Devices and Systems*. Retrieved from https://blog.ipspace.net/2013/08/management-control-and-data-planes-in.html

Perrin, S. (2017, Enero). *Making Networks SDN-Ready With Segment Routing.* Retrieved from https://www.segment-routing.net/images/lightreading_report.pdf

Platón. (369 ac). *La Replública.*

Prabhu, V. (2019). *What is Cisco SDWAN - Part 1.* Retrieved from https://developer.cisco.com/sdwan/video/

Preston, H. (2017, Octubre 25). *Intent Networks - How to be a Network Engineer in a Programmable Age.* Retrieved from https://www.netacad.com/sites/default/files/images/careers/Webinars/DevNet/devnet_session_3_intent_networks.pdf

ProgrammerSought. (2021). *Ubuntu install RYU and Mininet steps (using Python3.6 version).* Retrieved from https://www.programmersought.com/article/20364436241/

Puppet. (2021, Enero 1). *Puppet System Requirements*. Retrieved from https://puppet.com/docs/puppet/7.1/system_requirements.html

Ransbotham, S., Kiron, D., & Gerbert, P. (2017). Reshaping bussiness with Artificial Intelligence. *MIT Sloan Management Review*.

Rivenes, L. (2016, Septiembre 2). *Why DevOps Should Care About SDN*. Retrieved from https://devops.com/devops-care-sdn/

Rodriguez-Natal, A., Portoles-Comeras, M., Ermagan, V., Farinacci, D., Maino, F., & Cabellos-Aparicio, A. (2015). LISP: a southbound SDN protocol? *IEEE Communications Magazine*, 201-207. doi:10.1109/MCOM.2015.7158286

Roman, T., & Bryan, T. (2018, Enero 29). *Model Driven Network Automation with IOS-XE.* Retrieved from https://www.ciscolive.com/c/dam/r/ciscolive/emea/docs/2018/pdf/LTRCRT-2700.pdf

Safris, S. (2019). *A Deep Look at JSON vs. XML, Part 1: The History of Each Standard*. Retrieved from https://www.toptal.com/web/json-vs-xml-part-1

Salazar Ch, G. D., & Naranjo, E. F. (2017). Underlay and overlay networks: The approach to solve addressing and segmentation problems in the new networking era: VXLAN encapsulation with Cisco and open source networks. *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*, 1-6. doi:10.1109/ETCM.2017.8247505

Salazar Ch., G. D., Naranjo, E. F., & Marrone, L. (2018). SDN-Ready WAN networks: Segment Routing in MPLS-Based Environments. *2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 173-178. doi:10.1109/UEMCON.2018.8796613

Salazar Chacón, G., & Chafla Altamirano, G. (2015). Empleo de Path-Control Tools en una red empresarial moderna mediante políticas de enrutamiento. *3C Tecnología*, 1-18. Recuperado de http://ojs.3ciencias.com/index.php/3c-tecnologia/article/view/233

Salazar, G., & Solano, J. P. (2018, Junio 14). *Fundamentos de Cisco Intelligent WAN.* (C. C. Support, Ed.) Retrieved from https://community.cisco.com/t5/videos-routing-y-switching/webcast-video-fundamentos-de-cisco-intelligent-wan/ba-p/3399530

Salazar, G., Venegas, C., Baca, M., & et-al. (2018). Open Middleware proposal for IoT focused on Industry 4.0. 2018 *IEEE 2nd Colombian Conference on Robotics and Automation (CCRA)* (pp. 1-6). doi: 10.1109/CCRA.2018.8588117

Salazar-Chacón, G. (2019, Noviembre 26). *Ansible y SDN en acción: Los pilares de la era de la Programabilidad*. Retrieved from https://www.youtube.com/watch?v=m0bSM8Xv10g

Salazar-Chacón, G. D., & Reinoso García, A. R. (2021). Segment-Routing Analysis: Proof-of-Concept Emulation in IPv4 and IPv6 Service Provider Infrastructures. *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, 1-7. doi:10.1109/IEMTRONICS52119.2021.9422559

Salazar-Chacón, G., & Marrone, L. (2020). OpenSDN Southbound Traffic Characterization: Proof-of-Concept Virtualized SDN-Infrastructure. *11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 0282-0287. doi:10.1109/IEMCON51383.2020.9284938

Salazar-Chacón, G. D, & Vaca, J. E. (2020). VXLAN-IPSec Dual-Overlay as a Security Technique in Virtualized Datacenter Environments. *2020 IEEE ANDESCON*, 1-6, doi: 10.1109/ANDESCON50619.2020.9272160.

Salazar-Chacón, G., Naranjo, E., & Marrone, L. (2020). Open networking programmability for VXLAN Data Centre infrastructures - Ansible and Cumulus Linux feasibility study. *Revista Ibérica de Sistemas e Tecnologias de Informação*(32), 469-482.

Salazar, G. (2016). Fundamentos de QoS-Calidad de Servicio en Capa 2 y Capa 3, 2016, [online] Available: https://community.cisco.com/t5/blogsrouting-y-switching/fundamentos-de-qos-calidad-de-servicio-en-capa-2-ycapa-3lba-p/3103715.

Salazar Ch, G. D., Venegas, C. & Marrone, L. (2019). MQTT-Based Prototype Rover with Vision-As-A-Service (VAAS)in an IoT Dual-Stack Scenario. *2019 Sixth International Conference on eDemocracy & eGovernment (ICEDEG)*, 2019, pp. 344-349, doi: 10.1109/ICEDEG.2019.8734341

Salazar Ch., G.D., Hervas, C, Estevez E., & Marrone, L. High-Level IoT Governance Model Proposal for Digitized Ecosystems, *2019 International Conference on Information Systems and Software Technologies (ICI2ST)*, 2019, pp. 79-84, doi: 10.1109/ICI2ST.2019.00018.

Salazar, G. *Direccionamiento IPv6 - Bases y Fundamentos*. Cisco, 02 Febrero 2016. [online]. Disponible: https://supportforums.cisco.com/blog/12914981/direccionamiento-ipv6-bases-y-fundamentos

Salazar, G. *DMVPN Fase1 y 2 en IPv4 Fundamentos y Configuración básica enfocado al CCIE RS*, 2017, [online] Disponible: https://community.cisco.com/t5/videos-routing-y-switching/dmvpn-fase1-y-2-en-ipv4-fundamentos-y-configuraci%C3%B3n-b%C3%A1sica/ba-p/3104173

Salman, O., Elhajj, I., Kayssi, A., & Chehab, A. (2016). SDN controllers: A comparative study. *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, 1-6. doi:10.1109/MELCON.2016.7495430

SDX Central. (2019). *SDN Automation, Programmability, and Programmable Networks*. Retrieved from https://www.sdxcentral.com/automation/definitions/programmability-network-automation-sdn-networks/

SDX Central. (2019). *Understanding the SDN Architecture – SDN Control Plane & SDN Data Plane*. Retrieved from https://www.sdxcentral.com/networking/sdn/definitions/inside-sdn-architecture/

SDxCentral. (2015, Mayo 7). *What's a Software Defined Data Center?* Retrieved from https://www.sdxcentral.com/data-center/definitions/software-defined-data-center/

Segment Routing. (2021, Abril). *Segment Routing Main Page*. Retrieved from https://www.segment-routing.net/

Tamilselvan, G., Froehlich, J., & Raghunathan, Y. (2019, Febrero 1). *Cisco Live 2019 - Barcelona - Network Automation with Ansible*.

Teare, D., Vachon, B., & Graziani, R. (2015). *Implementing Cisco IP Routing (Route). Foundation Learning Guide.* Indianapolis, IN, USA: Cisco Press.

TeleGeography. (2020). *Submarine Cable Map*. Retrieved from https://www.submarinecablemap.com/

Tittel, E. (2018). *SDN vs. NFV: What's the difference?* Retrieved from https://www.cisco.com/c/en/us/solutions/software-defined-networking/sdn-vs-nfv.html

Vaca, J., & Salazar-Chacón, G. (2020). VXLAN-IPSec Dual-Overlay as a Security Technique in Virtualized Datacenter Environments. *IEEE ANDESCON*, 1-6.

Velrajan, S. (2019, Abril 23). *List of OpenFlow Controllers for SDN* . Retrieved from https://www.thetech.in/2012/12/list-of-openflow-controllers-for-sdn.html

VMWare SDX-Central. (2018). *What is VMware Network Virtualization?* Retrieved from https://www.sdxcentral.com/networking/virtualization/definitions/what-is-vmware-network-virtualization/

Wexler, S. (2016, octubre 31). *Digital Transformation Changes Everything — Are you Ready?* Retrieved from https://www.cio.com/article/3132004/digital-transformation-changes-everything-are-you-ready.html

World Economic Forum. (2017). *Realizing the Internet of Things: A Framework for Collective Action*. Retrieved from World Economic Forum Annual Meeting 2017: https://www.weforum.org/whitepapers/realizing-the-internet-of-things-a-framework-for-collective-action

Xu, Z., Liu, F., Wang, T., & Xu, H. (2016). Demystifying the energy efficiency of Network Function Virtualization. *IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)* (pp. 1-6). Beijing: IEEE Xplore.

Yigal, A. (2017, Enero 16). *Chef vs. Puppet: Methodologies, Concepts, and Support*. Retrieved from https://logz.io/blog/chef-vs-puppet/

Zoher Bholebawa, I., & Dalal, U. (2016, Abril 18). *Design and Performance Analysis of OpenFlow-Enabled Network Topologies Using Mininet.* Retrieved from http://www.ijcce.org/vol5/469-CT013.pdf

Anexos

Anexo A: Formatos y Estructuras de Datos para *NetDevOps*

EL término automatización, según (Cisco Netacad, 2020), "es cualquier proceso impulsado de forma automática que reduce y, a la larga, elimina, la necesidad de intervención humana", proceso que en sus inicios estuvo limitado a la fabricación industrial en masa debido al comportamiento altamente repetitivo por ejemplo, en el ensamblaje automotriz. Las máquinas son excelentes en la repetición de tareas y siempre que lo hagan de una manera acorde a lo planificado, los errores disminuyen al mínimo.

Al traer la automatización a la Informática y luego a las Tecnologías de la Información mediante el concepto de *NetDevOps*, la hiperconectividad juega un papel de suma importancia, ya que es la era donde múltiples y variados dispositivos se interconectan. Esos dispositivos deben compartir una estructura comunicacional para entenderse. Con el fin de que la estandarización avance más rápido, los entornos *NetDevOps*, así como las *Infrastructure-as-Code* y SDN, establecieron que lo ideal es contar con formatos de datos claros y efectivos para así la información compartida sea empleada por todos los elementos de red (*Observer Pattern* con *Separation of Concerns*), tanto finales como intermediarios; es así como los formatos de datos especifican una manera de almacenar, intercambiar y procesar datos estructuradamente. Un ejemplo sencillo y ampliamente utilizado es el Lenguaje de Marcas de Hipertexto (HTML por sus siglas en inglés), el cual describe la estructura de páginas Web.

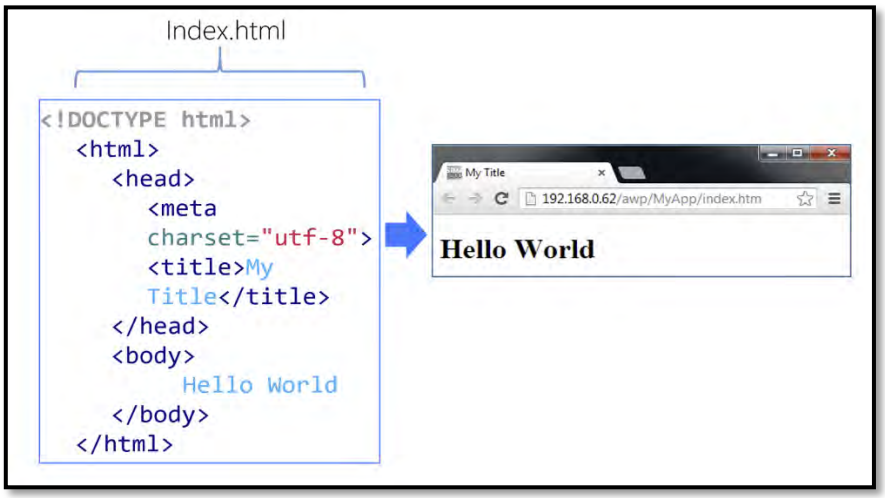


Figura A – 1 Formato de Datos: HTML
Recuperado de (Jager, 2013)

De acuerdo con (Cisco DevNet - Programming Fundamentals, 2020), los principales formatos de datos utilizados en automatización de una red son:

- JavaScript Object Notation – JSON
- eXtensible Markup Language – XML
- YAML Ain't Markup Language – YAML

La elección del formato adecuado dependerá de la aplicación, herramienta o infraestructura donde se ejecuten, pues cada uno tiene sus características en particular.

Estructura y Reglas de Formato de Datos para JSON, YAML y XML

Los diversos formatos de datos cuentan con una estructura similar al de los lenguajes de programación e incluso el lenguaje natural humano, por ejemplo, entre las características más importantes están:

- Poseen una sintaxis determinada, la cual puede incluir indentación (sangría o espaciado), uso de comillas, comas, así como [ ], { } o ( ).
- Formas de identificar objetos ya sean como listas, arreglos o cadenas.
- Uso del concepto clave:valor con el fin de describir un objeto, dato o variable.

Formato JSON

JSON es considerado un formato de datos de fácil lectura y comprensión, empleado principalmente en aplicaciones para almacenar, transferir o leer datos.

En la actualidad, JSON es uno de los formatos de mayor popularidad según (Safris, 2019). Quizá una de las razones para ello es su facilidad de integración con lenguajes como *Python*. Esa tendencia se puede ver en la siguiente imagen:

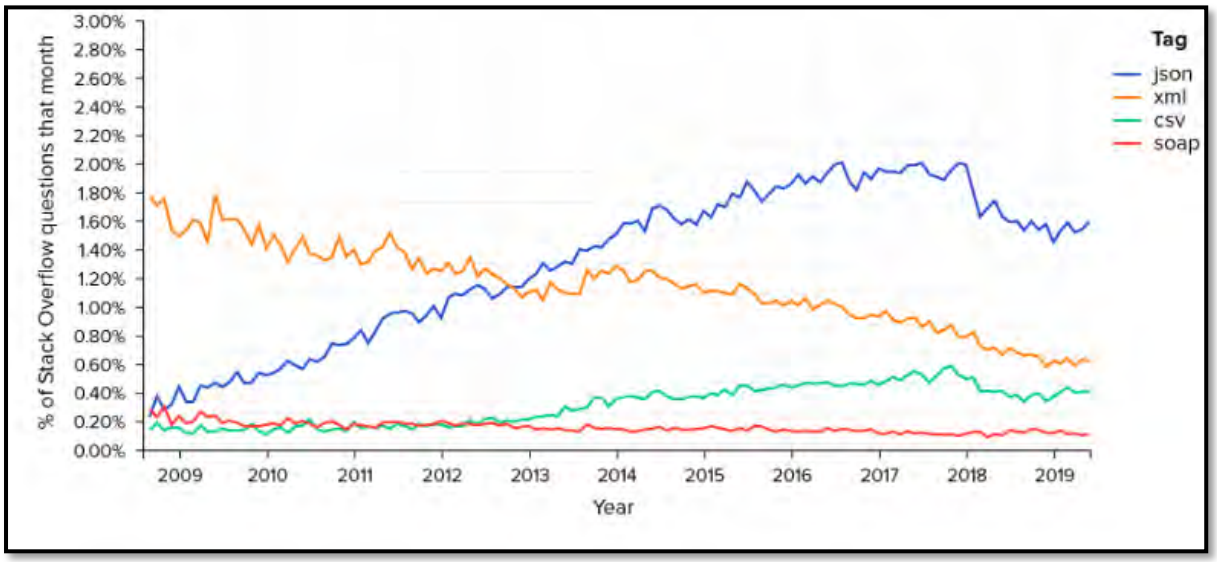


Figura A – 2 Popularización de JSON frente al resto de formato de datos
***Recuperado de* (Safris, 2019)**

Reglas de Sintaxis para JSON

Las características fundamentales de JSON son:

- Utiliza una estructura de anidación con sentido jerárquico
- Emplea llaves { } para limitar objetos y corchetes [ ] para vectores.
- La información sobre variables o datos está escrita en clave:valor

En JSON, es importante tener claro lo que se conoce como OBJETO, el cual es uno o más pares clave:valor dentro de llaves { }. La sintaxis de un objeto incluye:

- Las claves debe ser cadenas de caracteres o *strings* escritas entre ‘’ “ o comillas dobles.
- Los valores de dichas claves deben estar acorde a los tipos de datos aceptados en JSON (números, cadenas de caracteres, arreglos, booleanos, vacío o *null* y cualquier otro objeto anidado).
- La clave y los valores están separados por : (dos puntos)

- Múltiples pares clave:valor dentro de un mismo objeto están separados por comas.
- Los espacios en blanco no tienen importancia en JSON.

En algunas ocasiones, una clave puede tener más de un valor, esto se conoce como un arreglo o *array*, por lo que un arreglo en JSON es una lista ordenada de valores dentro de una clave.

Las características principales de un arreglo incluyen:

- Luego de escribir la clave con sus respectivos dos puntos que indican el inicio del arreglo, la lista de valores está limitado por corchetes [ ].
- El arreglo, al ser una lista ordenada de valores de varios tipos, desde booleanos, *strings*, números, etc., cada arreglo, estará separado por una coma.

El siguiente ejemplo muestra una lista de direcciones IPv4. En él, la clave es *"addresses"*. El arreglo es el conjunto de direcciones IPv4 limitado entre [ ], mientras cada objeto dentro del *array* es el par de clave:valor compuesto por la dirección IP y su máscara de subred (*subnet mask*) limitados entre llaves { } y separados por una coma al ser varios objetos dentro del mismo valor.

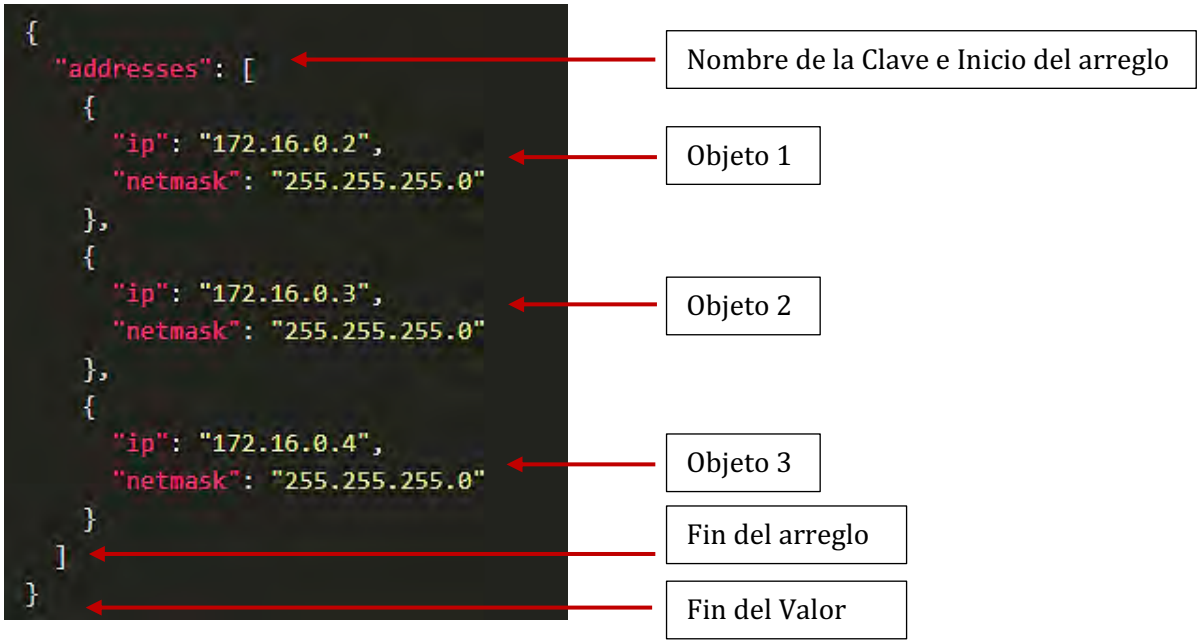


Figura A – 3 Ejemplo de Formato JSON
***Basado de* (Cisco Netacad, 2020)**

Formato YAML

Junto con JSON, YAML es otro tipo de formato de datos considerado de fácil lectura que permite interactuar con aplicaciones para almacenar, transferir y leer datos.

Reglas de Sintaxis para YAML

Entre las características principales de YAML están:

- Es un tipo de *superset* de JSON.
- Tiene un formato más simple y con menos limitadores de objetos que JSON, haciéndolo más fácil de leer y escribir en un *script*.
- Emplea indentación y espaciado para establecer la pertenencia a un objeto o a un *array* sin la necesidad de utilizar corchetes o comas.

Con el fin de entender de mejor manera a YAML, se comparará un resultado con información de direccionamiento IPv4 para una interfaz *GigabitEthernet2* de un equipo con Cisco IOS-XE, tanto en JSON y su respectivo *output* con YAML:

Formato JSON

```
{
   "ietf-interfaces:interface": {
      "name": "GigabitEthernet2",
      "description": "Wide Area Network",
      "enabled": true,
      "ietf-ip:ipv4": {
         "address": [
            {
               "ip": "172.16.0.2",
               "netmask": "255.255.255.0"
            },
            {
               "ip": "172.16.0.3",
               "netmask": "255.255.255.0"
            },
            {
               "ip": "172.16.0.4",
               "netmask": "255.255.255.0"
            }
         ]
      }
   }
}
```

Formato YAML

```
ietf-interfaces:interface:
  name: GigabitEthernet2
  description: Wide Area Network
  enabled: true
  ietf-ip:ipv4:
    address:
    - ip: 172.16.0.2
      netmask: 255.255.255.0
    - ip: 172.16.0.3
      netmask: 255.255.255.0
    - ip: 172.16.0.4
      netmask: 255.255.255.0
```

Figura A – 4 Comparación Formato JSON vs Formato YAML
***Recuperado de* (Cisco Netacad, 2020)**

La *Figura A – 4 Comparación Formato JSON vs Formato YAML* permite verificar que, en efecto, YAML es más fácil de leer, si bien mantiene una similitud con JSON, su sintaxis es más simple.

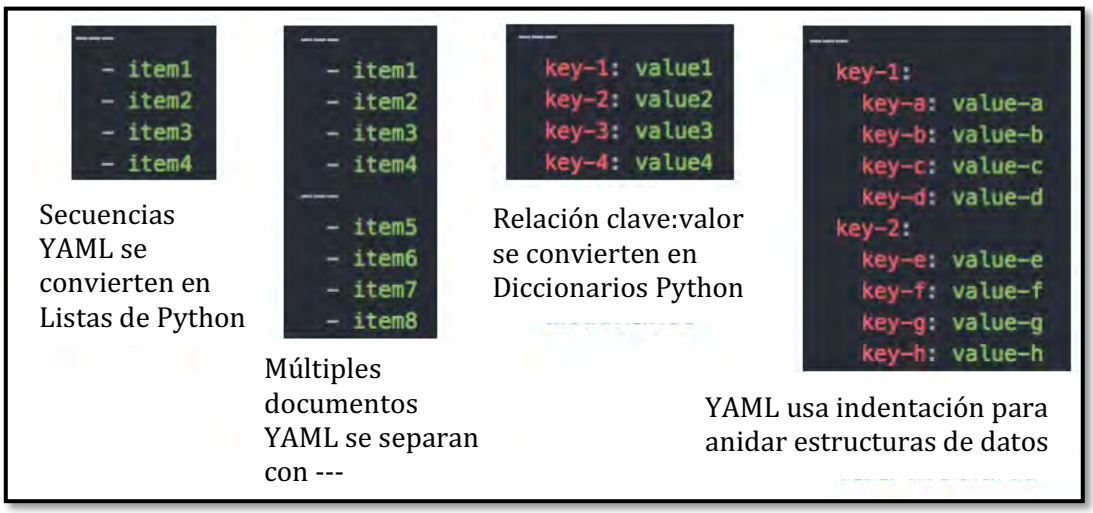


Figura A – 5 Explicación de Formato YAML
***Recuperado de* (Citakovic, 2019)**

Un objeto en YAML es un par de clave:valor. Un guion es usado para separar un elemento del *array* o lista de objetos en YAML.

Formato XML

XML es el tercer formato de datos más usado para *NetDevOps*, sin embargo, en los últimos años, su uso ha ido decreciendo, quizá debido a la simplicidad que muestran tanto JSON como YAML.

Reglas de Sintaxis para XML

Entre las características principales de XML están:

- Es un tipo de formato de dato muy similar a HTML, tanto en reglas como en sintaxis.
- Es considerado auto-descriptivo, ya que delimita los datos dentro de *tags:*

<tag>dato</tag>

- A diferencia de HTML, XML no usa tags predefinidos o estructura documental.

Un objeto en XML son el par clave/valor, siendo el nombre del tag, el nombre de la clave:

<clave>valor</clave>

Manteniendo el mismo ejemplo de la interfaz *GigabitEthernet2*, veremos un *script* con XML.

Cabe decir que la indentación o espaciado no es estrictamente necesario en XML, pero se puede usar para leer de mejor manera el *script*.

Formato YAML

```
ietf-interfaces:interface:
  name: GigabitEthernet2
  description: Wide Area Network
  enabled: true
  ietf-ip:ipv4:
    address:
    - ip: 172.16.0.2
      netmask: 255.255.255.0
    - ip: 172.16.0.3
      netmask: 255.255.255.0
    - ip: 172.16.0.4
      netmask: 255.255.255.0
```

Formato XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<ietf-interfaces:interface>
  <name>GigabitEthernet2</name>
  <description>Wide Area Network</description>
  <enabled>true</enabled>
  <ietf-ip:ipv4>
    <address>
      <ip>172.16.0.2</ip>
      <netmask>255.255.255.0</netmask>
    </address>
    <address>
      <ip>172.16.0.3</ip>
      <netmask>255.255.255.0</netmask>
    </address>
    <address>
      <ip>172.16.0.4</ip>
      <netmask>255.255.255.0</netmask>
    </address>
  </ietf-ip:ipv4>
</ietf-interfaces:interface>
```

Figura A – 6 Comparación Formato YAML vs Formato XML
***Recuperado de* (Cisco Netacad, 2020)**

Anexo B: Guía de Instalación GNS3-VM, EVE-ng y Mininet

En este anexo se mostrará un paso a paso de la instalación del *software* utilizado en los diversos PoCs de la tesis doctoral. Cada uno tiene sus características particulares que los hacen ideales para entornos de pruebas de concepto.

Instalación de GNS3-VM (WebUI)

GNS3 (*Graphical Network Simulator 3*), es un software de emulación de red, el cual nació en el 2008 gracias a Jeremy Grossman y a la evolución de *Dynamips*. Permite construir, diseñar y probar una red de datos, constituyéndose en un entorno ideal para pruebas de concepto.

GNS3-VM ha evolucionado hacia un entorno basado en navegador en lugar de una aplicación, lo que lo hace más rápido y versátil, funcionando muy bien con un hipervisor de VMWare, debido a que tiene habilitado Intel VT-x/AMD-v.

El primer paso comienza en la descarga de la VM de la página oficial de GNS3 y su importación en *VMWare Workstation Player/Pro/Fusion*

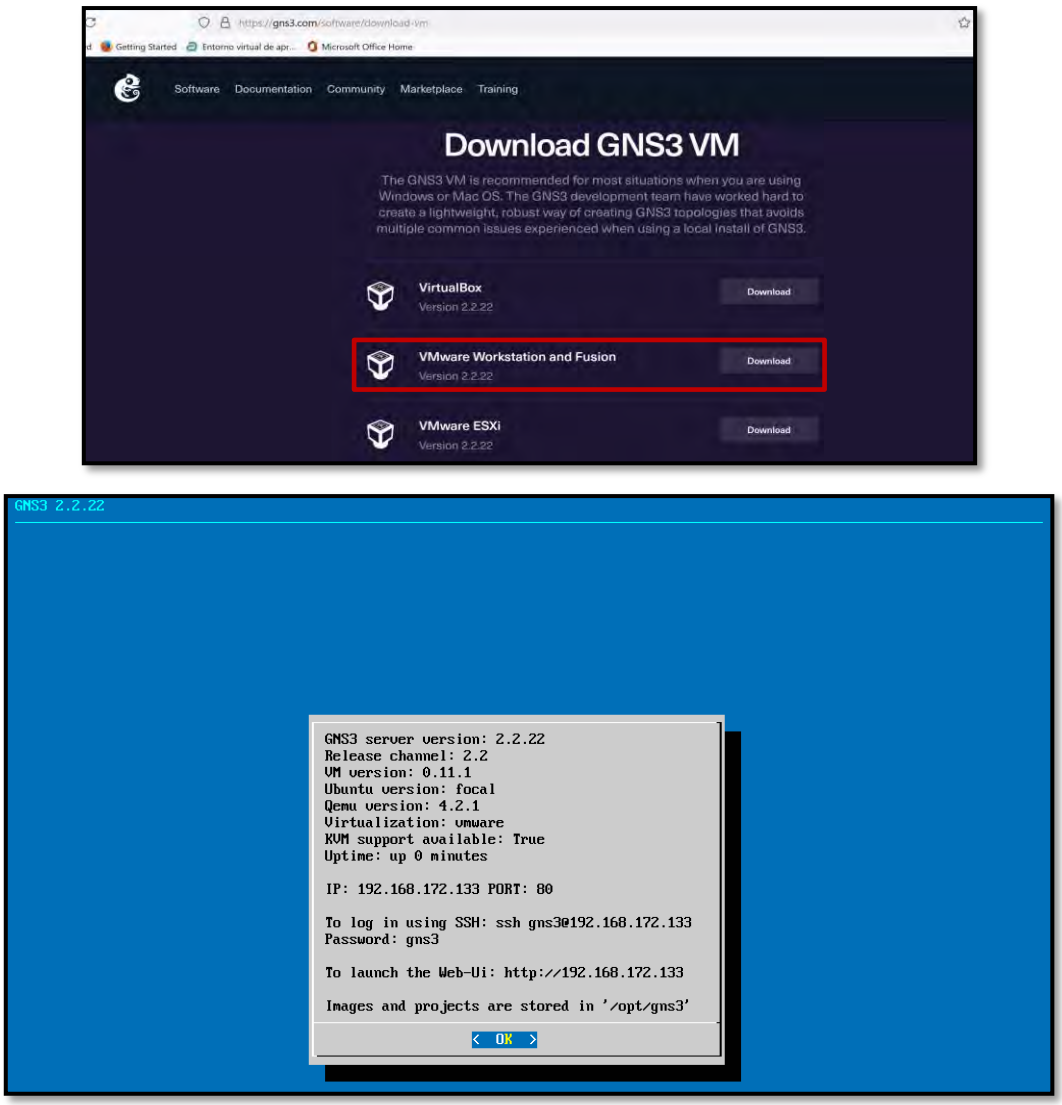


Figura B – 1 Descarga, Importación y Arranque de GNS3-VM
Fuente: *Autor*

El siguiente paso es abrir un navegador Web (de preferencia Mozilla Firefox) con la dir. IP de GNS3-VM como URL. Si la conexión es exitosa se muestra una GUI donde se carga las imágenes de los equipos a emular y se crean los proyectos de emulación (topologías).

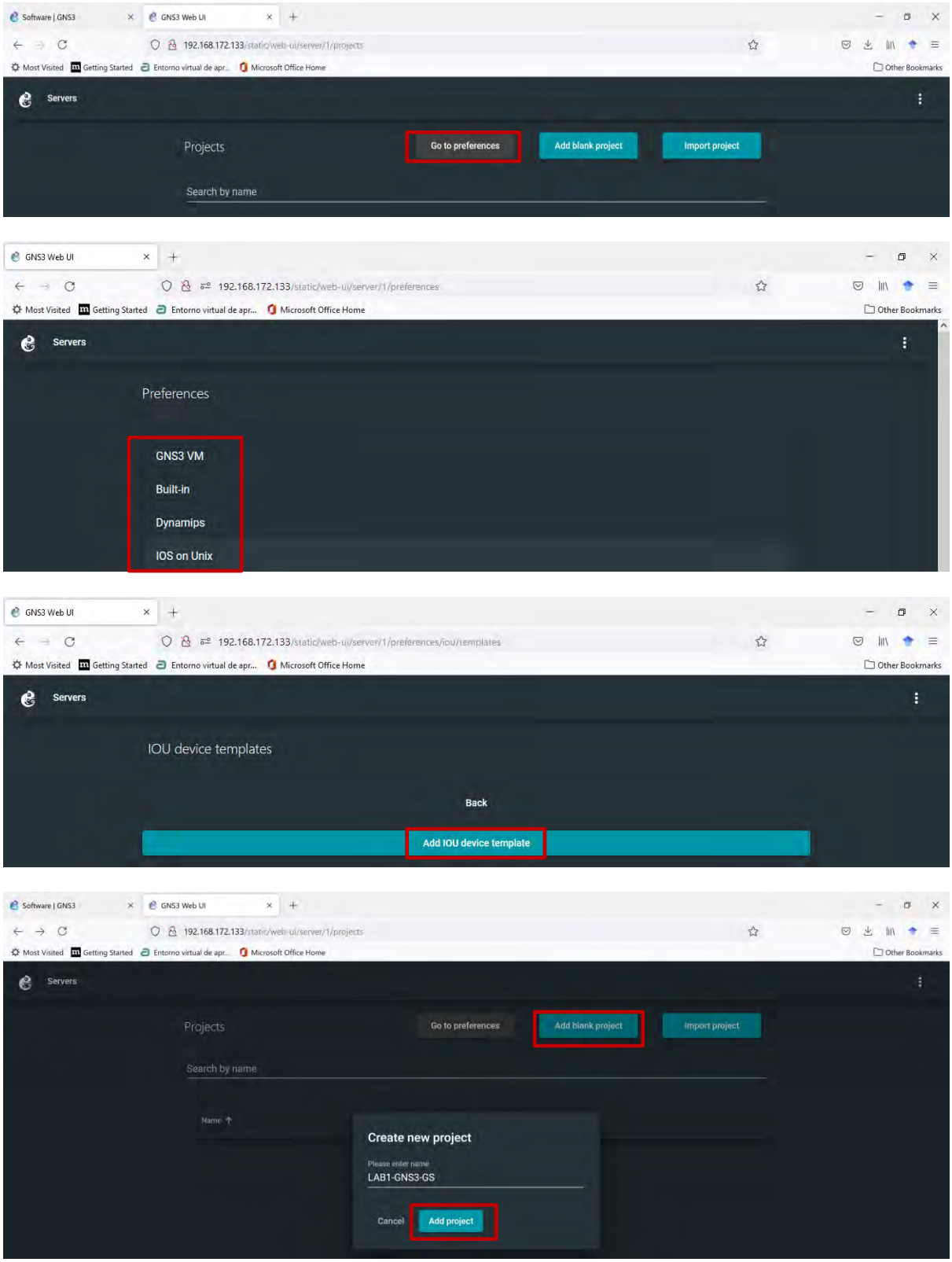


Figura B – 2 Carga de NOS a GNS3-VM y creación de nuevo proyecto
Fuente: *Autor*

Ya en la zona de trabajo, se podrá agregar los nodos para la emulación respectiva.

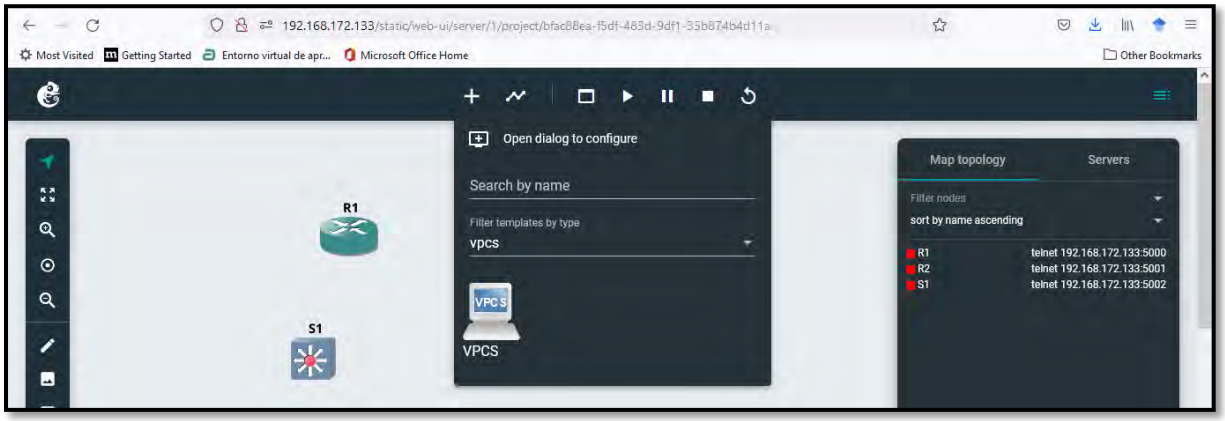


Figura B – 3 Zona de trabajo de GNS3-VM (WEBUI)
Fuente: *Autor*

Instalación de EVE-ng

EVE-ng (*Emulated Virtual Environment – Next Generation*), es un emulador similar a GNS3 en muchos aspectos, pero más flexible y con capacidad de emulación de más fabricantes de equipos de red y de nuevas tecnologías sin tanta complejidad o consumo de recursos.

EVE-ng cuenta con dos versiones: *Community Edition* (CE) y PRO/*Learning Center*.

Para instalarlas, debe ingresar a: https://www.eve-ng.net/index.php/download/

Para el caso de los PoCs, se usará la versión CE: https://www.eve-ng.net/index.php/community/

A la fecha de la realización de la presente tesis, EVE-ng v2.0.3-112 (11 de enero del 2021) es la más actual.

Descargue el ISO si se desea personalizar la configuración de la VM de EVE-ng o el archivo OVF si se desea utilizar las características que viene por defecto. La forma más sencilla es mediante OVF:

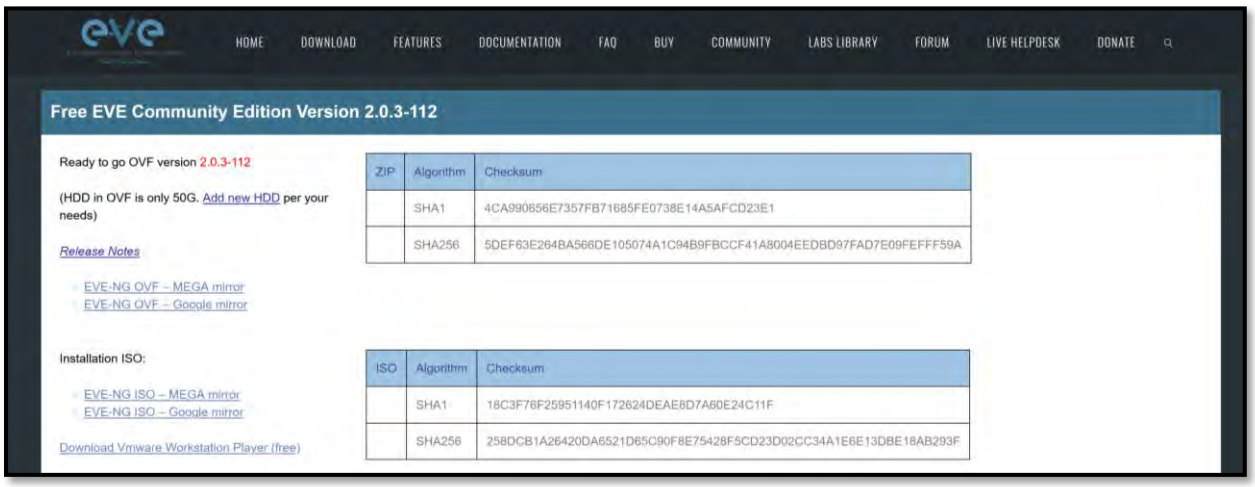
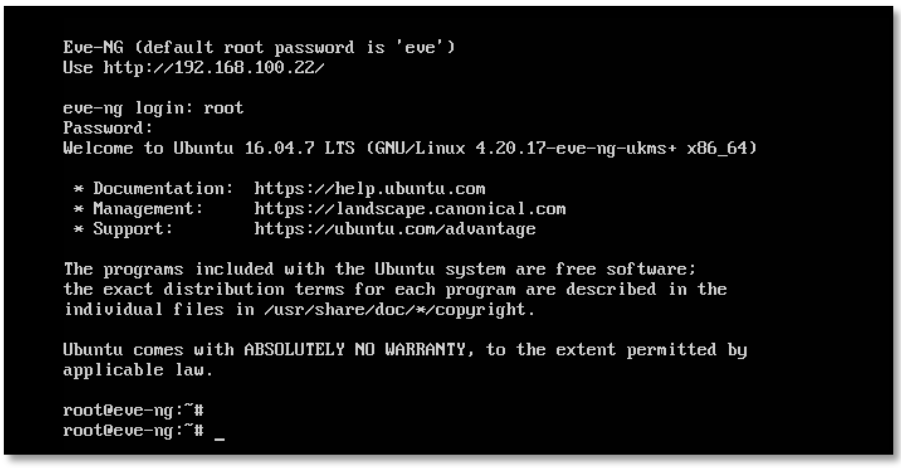


Figura B – 4 Descarga VM de EVE-ng.
Fuente: *Autor*

Al correr la VM, se podrá ingresar a su configuración, tanto por CLI (SSH), como por su GUI (mediante navegador WEB).



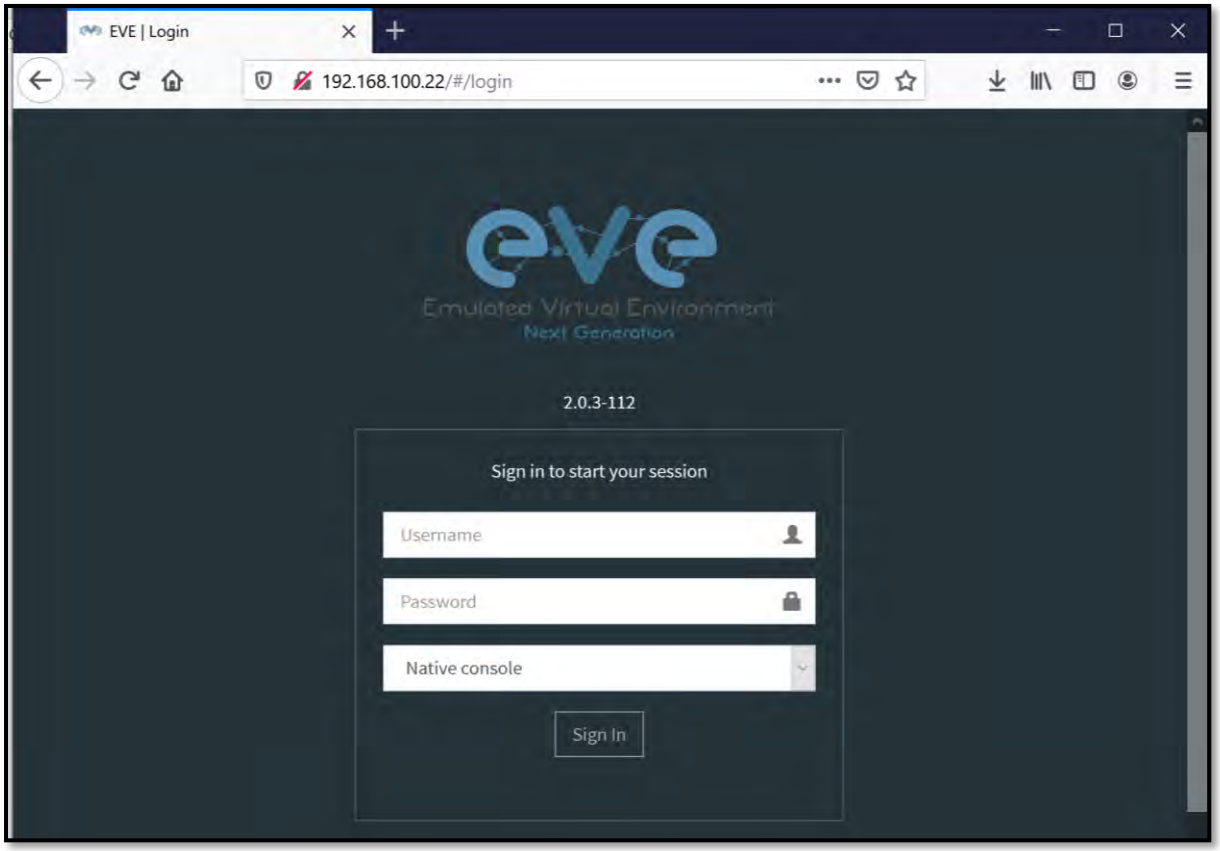


Figura B – 5 Acceso a EVE-ng.
Fuente: *Autor*

De la misma manera que en GNS3, al tener corriendo la VM, es necesario cargar los NOS a emular, pero en esta ocasión se realiza mediante un programa que establece una conexión SSH para luego compartir archivos con la VM de EVE-ng. En *Windows*, el *software* ideal es WinSCP.

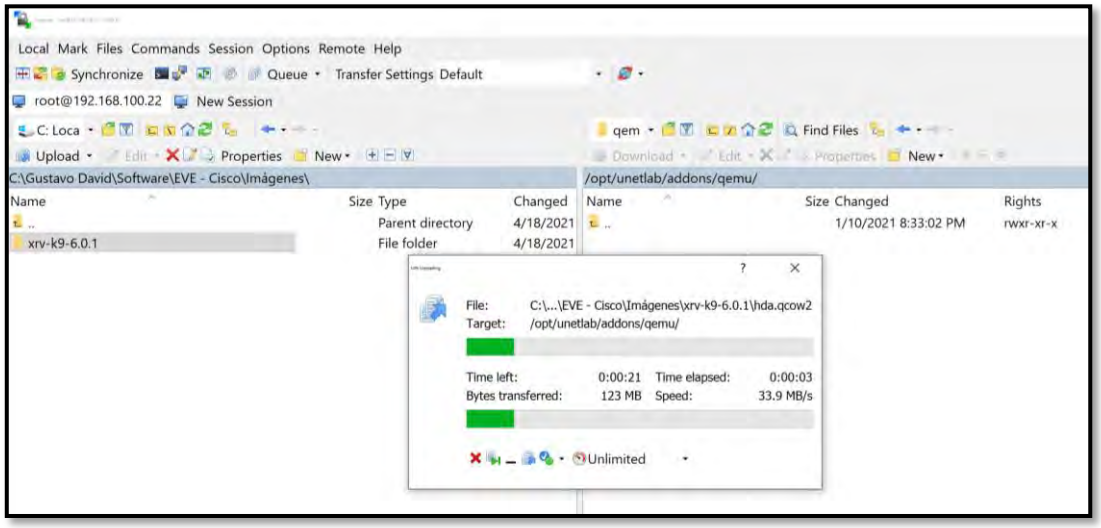


Figura B – 6 Envío de NOS a EVE-ng mediante WinSCP.
Fuente: *Autor*

Una vez están cargados los NOS a la VM, en EVE-ng, se debe otorgar permisos de ejecución con el siguiente comando:

/opt/unetlab/wrappers/unl_wrapper -a fixpermissions

A partir de ese momento, se puede crear un nuevo proyecto para empezar la emulación.

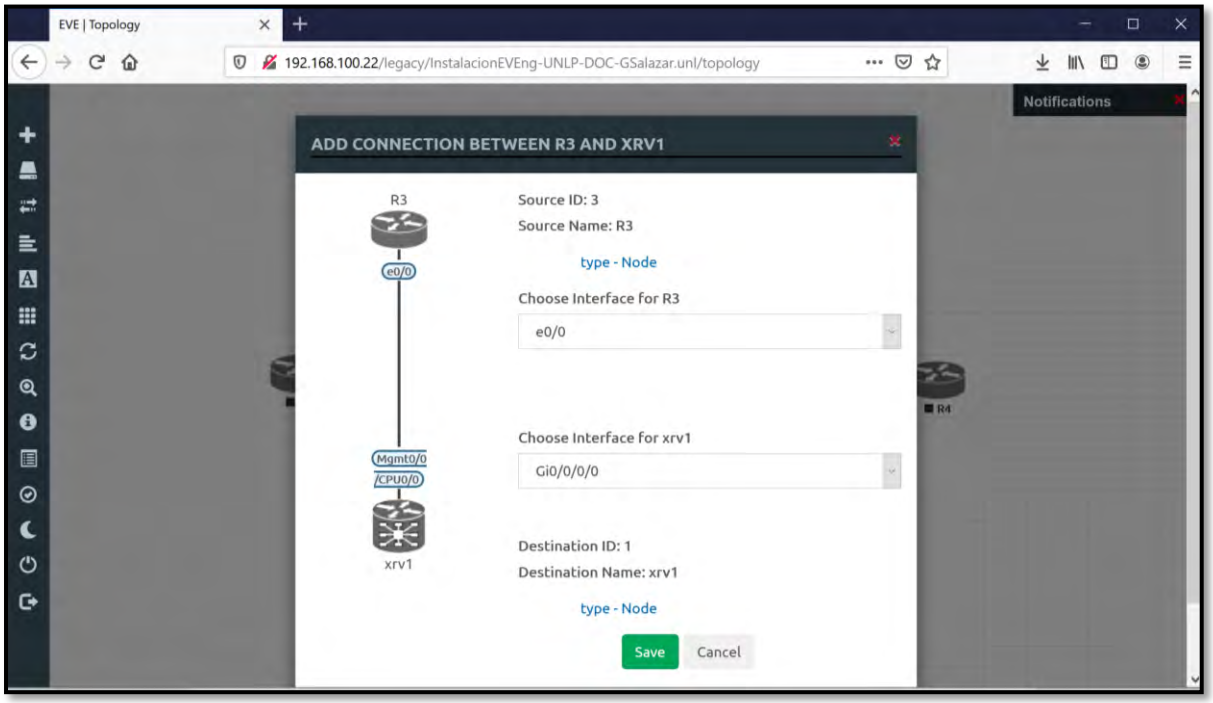


Figura B – 7 Ubicación de equipos y conexiones en Eve-ng
Fuente: *Autor*

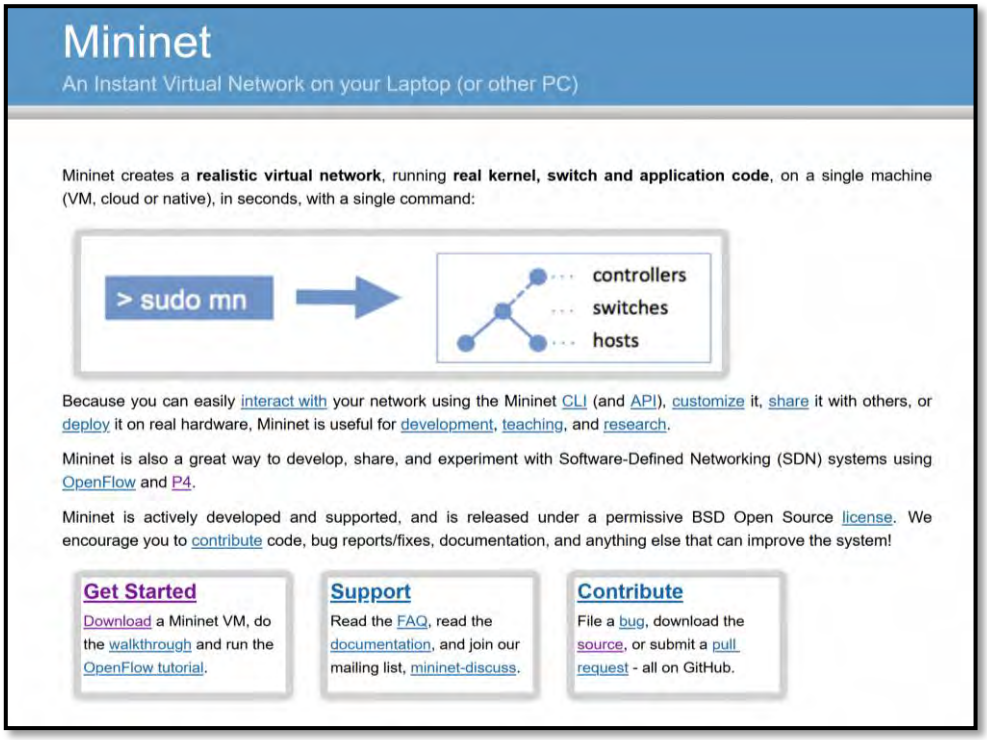
Instalación de Mininet

Mininet es un programa gratuito de emulación de redes de datos enfocado al análisis de paquetes y tráfico que circulan por una infraestructura. Permite emular desde usuarios finales, *routers, switches* y los enlaces que los conectan en un único Kernel Linux mediante una virtualización ligera para compartir recursos, lo que le otorga rapidez, optimización y adaptabilidad de creación de diversos escenarios.

Cabe recalcar que Mininet emula al 100% el comportamiento de un dispositivo de red, sólo que, en lugar de ser creados en *hardware*, son creados en *software*.

Mininet comprueba el comportamiento de redes basados en controlador como SDN.

Para la descarga de este emulador se ingresa a www.mininet.org. Una vez descargada, se importa a un hipervisor, para el caso de los PoCs se emplea *VMWare Workstation Pro*. Al arrancar, se podrá acceder a su CLI.



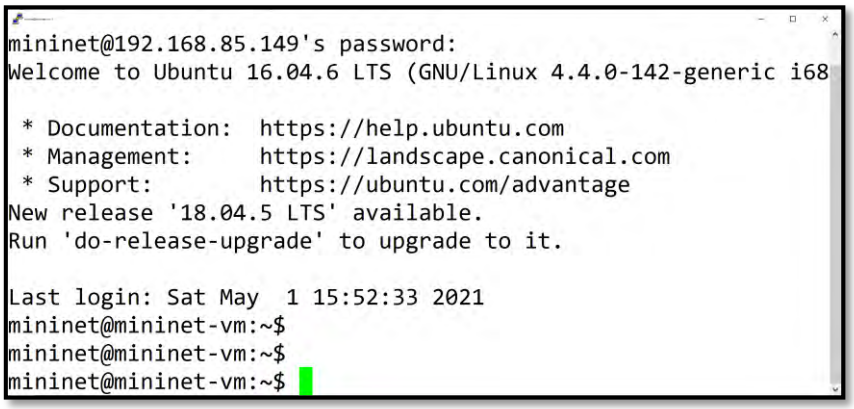


Figura B – 8 Descarga, Arranque e ingreso a CLI de Mininet
Fuente: *Autor*

Mininet cuenta con un controlador interno, pero se puede tener un controlador SDN externo como ODL, ONOS, RYU, tal como se comprobó en el PoC de integración de redes SDN con redes tradicionales y así contar con visibilidad, manejabilidad y las ventajas de un entorno controlado centralizadamente bajo un contexto *OpenSDN*.

Es factible también personalizar las infraestructuras a través de códigos escritos en *Python* tal como se puede apreciar en el blog de este enlace: https://zhuanlan.zhihu.com/p/141942411

Se debe mencionar que se puede instalar Mininet desde una CLI clonando el proyecto de GitHub:

git clone git://github.com/mininet/mininet

Anexo C: NETCONF en la Práctica

NETCONF es uno de los protocolos de administración y gestión que más relevancia tiene en la última ola de innovación en cuanto a *networking* se refiere, es por ello que en este Anexo se plantea una prueba de concepto práctico del uso de NETCONF para obtener datos de un equipo Cisco con Sistema Operativo IOS-XE (*CSR1000v-17.02.01-Amsterdam*), así como una configuración sencilla de este entorno.

Lo primero, es tener acceso a Internet tanto del Servidor Ubuntu (donde correrá NETCONF) como del *router,* para ello se utiliza un *Cloud* de GNS3-VM, ruta por defecto al *cloud* y PAT en *CSR1000v.*

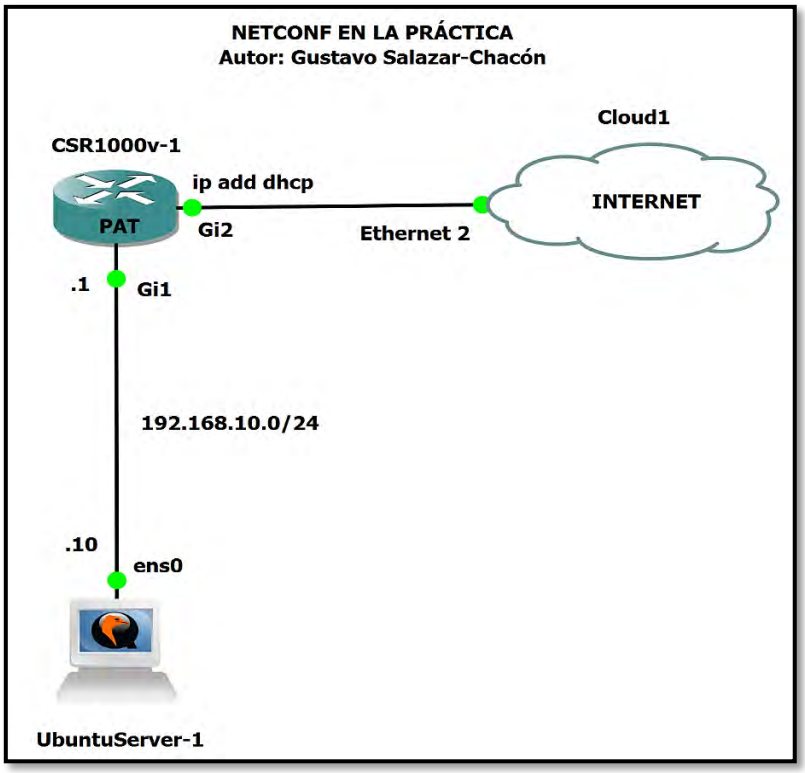


Figura C – 1 Infraestructura para Prueba de Concepto NETCONF (con salida a Internet)
Fuente: *Autor*

<u>Comprobación de Acceso a Internet por parte de los dispositivos</u>

Direccionamiento IP *Router CSR1000V*

```
CSR1KV_GSALAZAR1(config)#do show ip int brief | e ass
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.10.1    YES NVRAM  up                    up
GigabitEthernet2       192.168.1.113   YES DHCP   up                    up
```

```
CSR1KV_GSALAZAR1(config)#do ping www.google.com
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 142.250.78.36, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 17/18/20 ms
CSR1KV_GSALAZAR1(config)#
```

Figura C – 2 Direccionamiento Ipv4 y Conexión a Internet – CSR1000v
Fuente: *Autor*

Direccionamiento IP Servidor Ubuntu

```
gns3@gns3:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group defaul
t qlen 1000
    link/ether 0c:ca:16:2b:a0:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.10.10/24 brd 192.168.10.255 scope global noprefixroute ens3
       valid_lft forever preferred_lft forever
    inet6 fe80::2016:94c5:7365:bd5f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

```
gns3@gns3:~$ ping www.google.com
PING www.google.com (142.250.78.36) 56(84) bytes of data.
64 bytes from bog02s15-in-f4.1e100.net (142.250.78.36): icmp_seq=1 ttl=117 time=17.8 ms
64 bytes from bog02s15-in-f4.1e100.net (142.250.78.36): icmp_seq=2 ttl=117 time=18.1 ms
64 bytes from bog02s15-in-f4.1e100.net (142.250.78.36): icmp_seq=3 ttl=117 time=20.1 ms
64 bytes from bog02s15-in-f4.1e100.net (142.250.78.36): icmp_seq=4 ttl=117 time=17.2 ms
64 bytes from bog02s15-in-f4.1e100.net (142.250.78.36): icmp_seq=5 ttl=117 time=18.9 ms
^C
--- www.google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 17.265/18.465/20.141/0.991 ms
gns3@gns3:~$
```

Figura C – 3 Direccionamiento Ipv4 y Conexión a Internet – Servidor Ubuntu (Cliente NETCONF)
Fuente: *Autor*

Cabe decir, que tanto en el *router* como en el servidor se usó como *DNS-Server* a Cisco *Umbrella* (Cisco Systems - Cisco Umbrella) a través de las Dir. IPs: 208.67.222.222 / 208.67.220.220 por motivos de seguridad.

Establecimiento del Canal de comunicación NETCONF entre Servidor Ubuntu (Cliente NETCONF) y equipo de red (*Router CSR1000v)*

Con el fin de generar el canal de comunicación, es necesario establecer un programa en *Python* que permitirá obtener y configurar el equipo de red desde un cliente NETCONF.

- Habilitar credenciales de acceso (por motivos del PoC son simples) y NETCONF en *CSR1000v*

```
CSR1KV_GSALAZAR1(config)#
CSR1KV_GSALAZAR1(config)#username UNLP-GUS privilege 15 secret PHD-THESIS
CSR1KV_GSALAZAR1(config)#
CSR1KV_GSALAZAR1(config)#netconf-yang
CSR1KV_GSALAZAR1(config)#
CSR1KV_GSALAZAR1(config)#
*Dec 13 19:00:39.534: %PSD_MOD-5-DMI_NOTIFY_NETCONF_START: R0/0: psd: PSD/DMI: netconf-yang server has been n
otified to start
CSR1KV_GSALAZAR1(config)#
CSR1KV_GSALAZAR1(config)#
*Dec 13 19:01:55.069: %NDBMAN-5-ACTIVE: R0/0: ndbmand: All data providers active.
*Dec 13 19:02:07.003: %DMI-5-NACM_INIT: R0/0: dmiauthd: NACM configuration has been set to its initial config
uration.
*Dec 13 19:02:17.055: %DMI-5-SYNC_COMPLETE: R0/0: dmiauthd: The running configuration has been synchronized t
o the NETCONF running data store.
```

Script 61 Credenciales de acceso y habilitación de NETCONF – CSR1000v
Fuente: *Autor*

- Instalar *Python* y PIP (herramienta para administrar bibliotecas y módulos en los programas escritos en Python) en el Servidor Ubuntu en caso de no tenerlo instalado

```
gns3@gns3:~$ python3 --version
Python 3.6.8
gns3@gns3:~$
gns3@gns3:~$ python3.8 --version
Python 3.8.6
```

```
gns3@gns3:~$ pip3 --version
pip 9.0.1 from /usr/lib/python3/dist-packages (python 3.6)
```

Figura C – 4 Confirmación de instalación de Python3 y PIP3
Fuente: *Autor*

- Instalar el Cliente NETCONF (*ncclient package*[76])

```
gns3@gns3:~$ pip3 install ncclient
Collecting ncclient
  Downloading https://files.pythonhosted.org/packages/72/fd/ccae38393c22099229b68e8cdf06
1408b824e09ee207e2401c224398c5b0/ncclient-0.6.9.tar.gz (118kB)
    100% |████████████████████████████████| 122kB 882kB/s
Collecting lxml>=3.3.0 (from ncclient)
  Downloading https://files.pythonhosted.org/packages/bd/78/56a7c88a57d0d14945472535d0df
9fb4bbad7d34ede658ec7961635c790e/lxml-4.6.2-cp36-cp36m-manylinux1_x86_64.whl (5.5MB)
    100% |████████████████████████████████| 5.5MB 83kB/s
Collecting paramiko>=1.15.0 (from ncclient)
  Downloading https://files.pythonhosted.org/packages/95/19/124e9287b43e6ff3ebb9cdea3e5e
8e88475a873c05ccdf8b7e20d2c4201e/paramiko-2.7.2-py2.py3-none-any.whl (206kB)
    100% |████████████████████████████████| 215kB 127kB/s
```

Figura C – 5 Instalación de ncclient (Cliente NETCONF)
Fuente: *Autor*

- Crear un nuevo programa escrito en Python (**NETCONF-PoC-GSALAZAR.py**) teniendo como mánager a *ncclient* y usando un *template* Jinja2 junto con los parámetros de la sesión NETCONF

```
#Prueba de Concepto (PoC) de NETCONF
#Autor: Gustavo Salazar-Chacon

#Dependencias requeridas
from ncclient import manager
from jinja2 import Template

#Parametros Sesion NETCONF
m = manager.connect(host='192.168.10.1', port=830,
                    username='UNLP-GUS', password='PHD-THESIS',
                    device_params={'name': 'csr'})
```

Script 62 Definición de parámetros para la Sesión NETCONF
Fuente: Elaboración Propia

Selección del modelo YANG para NETCONF y *CSR1000v*

Para enviar una configuración desde un Cliente hacia el equipo de red o quizá realizar tareas de telemetría (recibir datos e información a distancia sobre un equipo), se debe establecer la estructura de los datos a emplear en el intercambio de mensajes durante la sesión de NETCONF.

[76] Ncclient: https://github.com/ncclient/ncclient

Los modelos YANG se emplean en las estructuras de datos de NETCONF. De igual manera, el equipo CSR1000v con su IOS-XE soporta diferentes modelos de estructuras de datos, entre ellos, modelos ***OpenConfig***[77], **YANG *IETF-Models (RFC 8343)***[78] y el formato ***IOS-XE native YANG model***[79]

Para el caso del PoC del Anexo C, se usará el formato **IOS-XE native YANG Model** de la siguiente manera para obtener información de una interfaz, de forma similar para ejecutar el comando **#show running-config interface gig1**

```
#Filtro de informacion de Configuracion con IOS-XE Native YANG Model
interface_filter = '''
  <filter>
      <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
          <interface>
            <GigabitEthernet>
              <name>1</name>
            </GigabitEthernet>
          </interface>
      </native>
  </filter>
'''

#Ejecucion del programa para obtener informacion tipo Show runn int gig1
result = m.get_config('running', interface_filter)
print(result)
```

Script 63 Estructura IOS-XE Native YANG Model para telemetría simple
Fuente: *Autor*

Uso del *output* de la consulta como *Jinja2 Template* para configurar una interfaz de *CSR1000v*

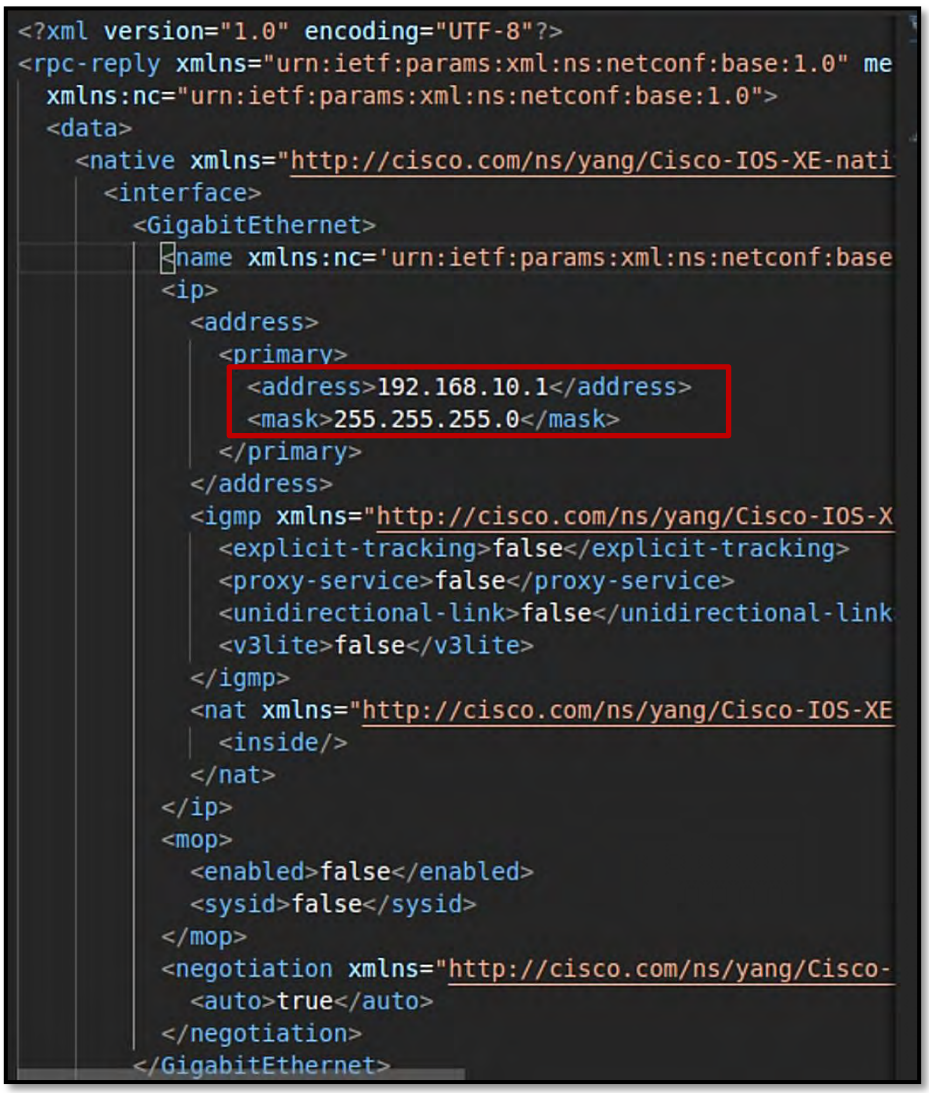
Al correr el programa con el comando **python3 NETCONF-PoC-GSALAZAR.py**, se obtiene este resultado:

```
gns3@gns3:~$ python3 NETCONF-PoC-GSALAZAR.py
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:bc6ed427-8267-419
1-9538-6232d6fa4130" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"><data><native xmlns="http
://cisco.com/ns/yang/Cisco-IOS-XE-native"><interface><GigabitEthernet><name xmlns:nc='urn:ietf:pa
rams:xml:ns:netconf:base:1.0'>1</name><ip><address><primary><address>192.168.10.1</address><mask>
255.255.255.0</mask></primary></address><igmp xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-igmp">
<explicit-tracking>false</explicit-tracking><proxy-service>false</proxy-service><unidirectional-l
ink>false</unidirectional-link><v3lite>false</v3lite></igmp><nat xmlns="http://cisco.com/ns/yang/
Cisco-IOS-XE-nat"><inside/></nat></ip><mop><enabled>false</enabled><sysid>false</sysid></mop><neg
otiation xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-ethernet"><auto>true</auto></negotiation></
GigabitEthernet></interface></native></data></rpc-reply>
gns3@gns3:~$
```

Script 64 Output del programa para telemetría simple – NETCONF
Fuente: *Autor*

[77] OpenConfig Models: https://github.com/openconfig
[78] IETF Models: https://tools.ietf.org/html/rfc8343
[79] IOS-XE YANG Models: https://github.com/YangModels/yang/tree/master/vendor/cisco/xe

Para entender de mejor manera el *output*, se lo abre con VS-CODE:



```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" me
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-nati
      <interface>
        <GigabitEthernet>
          <name xmlns:nc='urn:ietf:params:xml:ns:netconf:base
          <ip>
            <address>
              <primary>
                <address>192.168.10.1</address>
                <mask>255.255.255.0</mask>
              </primary>
            </address>
            <igmp xmlns="http://cisco.com/ns/yang/Cisco-IOS-X
              <explicit-tracking>false</explicit-tracking>
              <proxy-service>false</proxy-service>
              <unidirectional-link>false</unidirectional-link
              <v3lite>false</v3lite>
            </igmp>
            <nat xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE
              <inside/>
            </nat>
          </ip>
          <mop>
            <enabled>false</enabled>
            <sysid>false</sysid>
          </mop>
          <negotiation xmlns="http://cisco.com/ns/yang/Cisco-
            <auto>true</auto>
          </negotiation>
        </GigabitEthernet>
```

Script 65 Output del programa para telemetría simple en VS CODE – NETCONF
Fuente: *Autor*

El *Output* del *script* en VS-CODE servirá como un *Jinja2 Template* para configurar mediante NETCONF otra interfaz del *router CSR1000v*. Lo que se requiere, es copiar dicho output en otro archivo con extensión XML (**CONFIGNETCONF-PoC-GSALAZAR.xml**) bajo el *tag* ***<config>*** y reemplazando la dir. IP y máscara de subred con variables Jinja dentro de llaves {{}}, similar a la siguiente configuración:

```
<config>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
    <interface>
      <GigabitEthernet>
        <name
xmlns:nc='urn:ietf:params:xml:ns:332ocker332:base:1.0'>{{
INTERFACE_INDEX }}</name>
        <ip>
          <address>
            <primary>
              <address>{{ IP_ADDRESS }}</address>
              <mask>{{ SUBNET_MASK }}</mask>
            </primary>
```

```
            </address>
          </ip>
          <mop>
            <enabled>false</enabled>
            <sysid>false</sysid>
          </mop>
          <negotiation xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-
ethernet">
            <auto>true</auto>
          </negotiation>
        </GigabitEthernet>
      </interface>
    </native>
</config>
```

Script 66 Jinja2 Template de nombre CONFIGNETCONF-PoC-GSALAZAR.xml para Configuración de interfaz mediante NETCONF
Fuente: *Autor*

Finalmente, para comprobar la configuración mediante NETCONF, se definirán las variables antes de la corrida del *script* en el mismo archivo de Python, pero generando un nuevo *output*, ya no de telemetría, sino de cambio en el *running-config* del equipo *CSR1000v*:

```
 GNU nano 2.9.3                          NETCONF-PoC-GSALAZAR.py


#Ejecucion del programa para obtener informacion tipo Show runn int gig1
#result = m.get_config('running', interface_filter)
#print(result)

#Variables para el Jinja2 Template - Configuracion de Gig4
interface_template = Template(open('CONFIGNETCONF-PoC-GSALAZAR.xml').read())
interface_rendered = interface_template.render(
  INTERFACE_INDEX='4',
  IP_ADDRESS='172.16.100.100',
  SUBNET_MASK='255.255.255.240'
)

# Ejecutar la configuracion
result = m.edit_config(target='running', config=interface_rendered)
print(result)
```

Script 67 Modificación del Archivo en Python para correr junto con Jinja2 Template – NETCONF
Fuente: *Autor*

El resultado de la ejecución del programa del *script* anterior es la configuración mediante programabilidad NETCONF de la interfaz Gig4 del *router.*

```
gns3@gns3:~$
gns3@gns3:~$ python3 NETCONF-PoC-GSALAZAR.py
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:7d99fcee-6567-4d8
f-b938-c6dae28fa058" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"><ok/></rpc-reply>
```

Script 68 Ejecución exitosa (ok) del Archivo en Python – lectura de Jinja2 Template – NETCONF
Fuente: *Autor*

En el *router* se puede observar los logs de la autenticación del cliente NETCONF y la posterior configuración mediante programabilidad NETCONF:

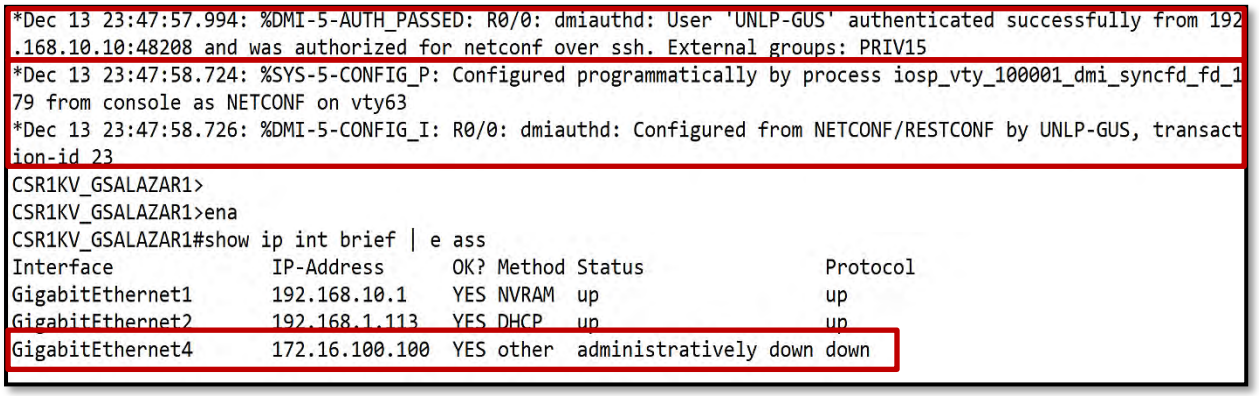


Figura C – 6 Comprobación de configuración programática mediante NETCONF en router CSR1000v
Fuente: *Autor*

Con este Anexo es posible comprobar la factibilidad de uso de NETCONF en la Era de la programabilidad, dando la posibilidad de realizar telemetría, monitoreo, adquisición de datos de equipos, así como la configuración de un equipo mediante estructura de datos tipo YANG.

Para más información sobre la configuración, filtraje, selección de modelos de datos YANG en equipos Cisco con IOS XE 16.X/17.X, puede ingresar al siguiente link:

https://www.cisco.com/c/en/us/support/docs/storage-networking/management/200933-YANG-NETCONF-Configuration-Validation.html

Anexo D: Ansible para entornos *NetDevOps* en infraestructuras de Red

Ansible es sin duda una de las mejores y más conocidas herramientas empleadas en la Era de la Programabilidad y en entornos *NetDevOps* debido a su facilidad de uso y entorno sin agente.

Esta prueba de concepto (PoC), la cual ha sido publicada en (Salazar-Chacón, Naranjo, & Marrone, 2020), permite comprobar la factibilidad de Ansible en entornos *OpenNetworking* con Cumulus Linux, implementando de forma programática y automática un entorno **VXLAN** entre dos sedes que comparten el mismo segmento L2, a pesar de estar separados un entorno L3, quitando así los límites tradicionales de una VLAN.

La topología tipo *Spine-Leaf* (Aruba - A Hewlett Packard Enterprise Company, n.d.) emulada es la siguiente:

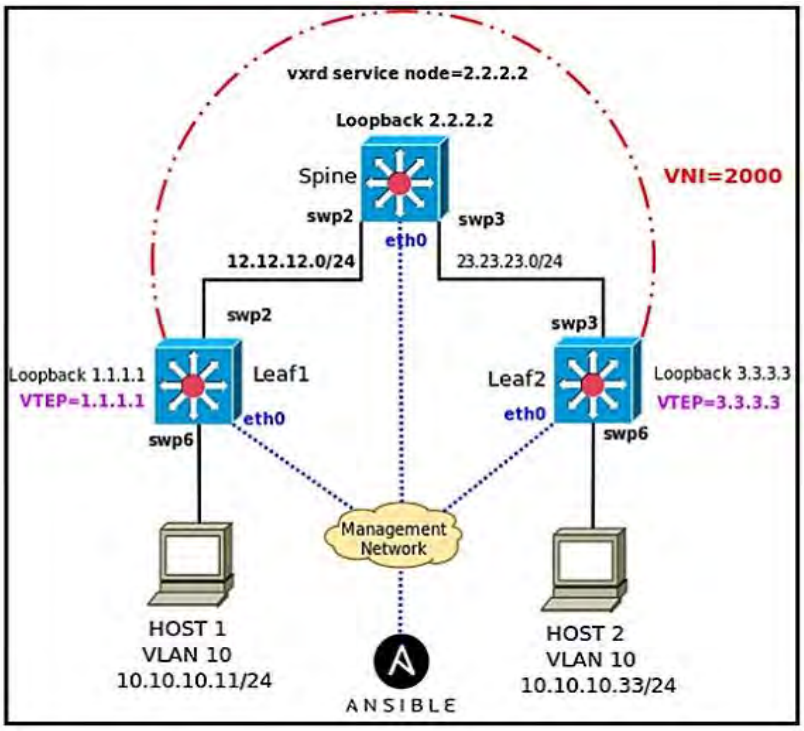


Figura D – 1 Topología para PoC de Ansible para configurar VXLAN en entornos Open Networking
***Recuperado** de (Salazar-Chacón, Naranjo, & Marrone, 2020)*

La infraestructura de la *Figura D – 1 Topología para PoC de Ansible para configurar VXLAN en entornos Open Networking*, se emula en la GNS3-VM junto con el *appliance* denominado *Network_Automation* de GNS3 *Marketplace* creado por Julien Duponchelle como dispositivo que cuenta con Ansible y otras herramientas de automatización de redes como Netmiko, NAPALM, entre otros. Este *appliance* es un Docker de Linux que contiene esas herramientas.

Figura D – 2 Network Automation Appliance – GNS3-VM Marketplace
Recuperado de (GNS3 Marketplace, 2017)

La topología emulada en GNS3-VM es la siguiente:

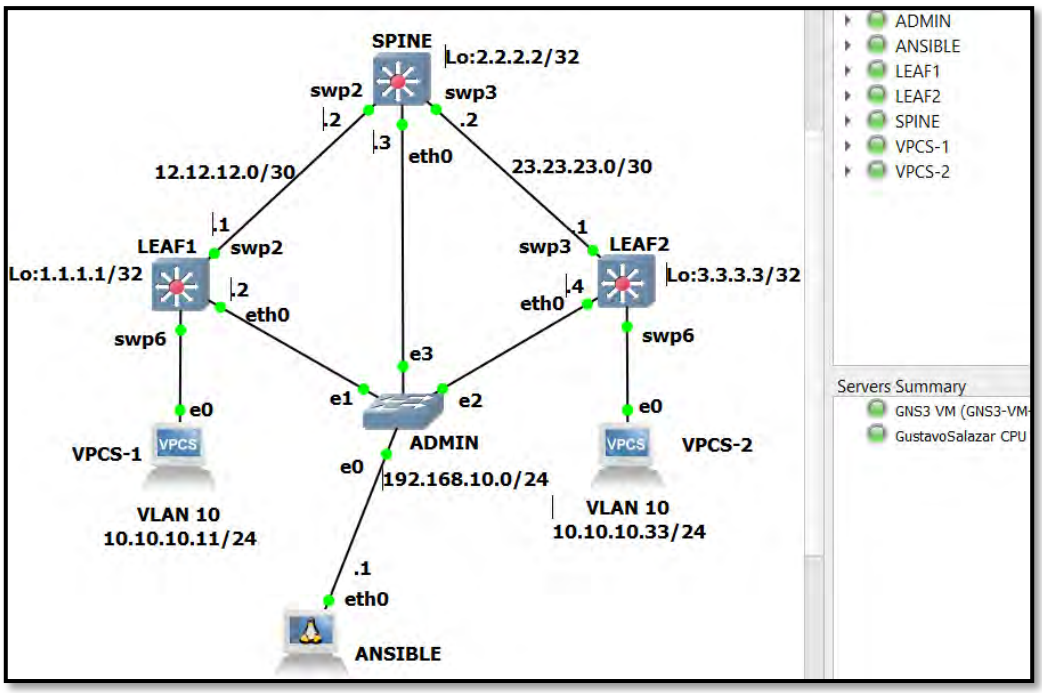


Figura D – 3 Topología emulada en GNS3-VM – Ansible para VXLAN en entornos OpenSource (Cumulus Linux) con Network Automation Docker
Fuente: *Autor*

Direccionamiento IP de Red de Administración

La red de Administración, donde se encuentra el equipo con Ansible, así como las interfaces *ethx* de los Cumulus Linux, está bajo la Dir. IP 192.168.10.0/24.

Se ingresa a los tres *switches Cumulus Linux* y se configura sus interfaces **eth0** con la dirección IP de administración correspondiente.

Para LEAF1

```
cumulus@cumulus:/etc/network$ net add interface eth0 ip address 192.168.10.2/24
cumulus@cumulus:/etc/network$
cumulus@cumulus:/etc/network$ net commit
--- /etc/network/interfaces     2019-05-03 05:39:38.000000000 +0000
+++ /run/nclu/ifupdown2/interfaces.tmp  2019-09-03 05:03:39.789000000 +0000
```

Para SPINE

```
cumulus@cumulus:~$ net add interface eth0 ip address 192.168.10.3/24
cumulus@cumulus:~$
cumulus@cumulus:~$ net commit
--- /etc/network/interfaces     2019-05-03 05:39:38.000000000 +0000
+++ /run/nclu/ifupdown2/interfaces.tmp  2019-09-03 05:05:55.175000000 +0000
```

Para LEAF2

```
cumulus@cumulus:~$ net add interface eth0 ip address 192.168.10.4/24
cumulus@cumulus:~$
cumulus@cumulus:~$ net commit
--- /etc/network/interfaces     2019-05-03 05:39:38.000000000 +0000
+++ /run/nclu/ifupdown2/interfaces.tmp  2019-09-03 05:07:18.053000000 +0000
```

Script 69 Configuración de Direccionamiento Ipv4 en equipos Cumulus Linux (OpenNetworking)
Fuente: *Autor*

Se prueba la conectividad entre *Ansible* con *Spine, Leaf1 y Leaf2*:

```
root@ANSIBLE:~# ping 192.168.10.2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=0.783 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=0.474 ms
^C
--- 192.168.10.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.474/0.628/0.783/0.156 ms
root@ANSIBLE:~# ping 192.168.10.3
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.
64 bytes from 192.168.10.3: icmp_seq=1 ttl=64 time=2.14 ms
64 bytes from 192.168.10.3: icmp_seq=2 ttl=64 time=0.376 ms
^C
--- 192.168.10.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.376/1.259/2.142/0.883 ms
root@ANSIBLE:~# ping 192.168.10.4
PING 192.168.10.4 (192.168.10.4) 56(84) bytes of data.
64 bytes from 192.168.10.4: icmp_seq=1 ttl=64 time=2.14 ms
64 bytes from 192.168.10.4: icmp_seq=2 ttl=64 time=0.335 ms
64 bytes from 192.168.10.4: icmp_seq=3 ttl=64 time=0.897 ms
^C
--- 192.168.10.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.335/1.126/2.147/0.757 ms
root@ANSIBLE:~#
```

Figura D – 4 Prueba de conectividad entre Ansible, Spine y Leaf1-Leaf2
Fuente: *Autor*

Llaves SSH en equipos Cumulus Linux

Una vez que se ha comprobado la conectividad en la red de administración, se debe generar **llaves SSH** en cada equipo de la red para la adecuada conexión entre Ansible y la red programable. Para este PoC, no se generará un passphrase.

```
cumulus@cumulus:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/cumulus/.ssh/id_rsa):
Created directory '/home/cumulus/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/cumulus/.ssh/id_rsa.
Your public key has been saved in /home/cumulus/.ssh/id_rsa.pub.
The key fingerprint is:
74:84:7e:f8:61:21:b9:c8:e0:48:04:f6:a8:27:7d:7e cumulus@cumulus
The key's randomart image is:
+---[RSA 2048]----+
|.+.      o.      |
|. + .   +..      |
| o + o o.+..     |
|... . o.+.+      |
|o o .   S+ .     |
| o o      .      |
|    . E          |
|     .           |
|                 |
+-----------------+
cumulus@cumulus:~$
```

Script 70 Generación de llaves SSH para conexión entre Ansible y equipos Cumulus Linux
Fuente: *Autor*

*Realice este paso en los equipos Spine, Leaf1 y Leaf2, no en Ansible.

Configuración de Ansible en *Network Automation Appliance*

Para configurar Ansible, se debe crear/editar tanto el archivo **hosts** como el *playbook*, que en este caso se llamará **vxlan.yml**

Hosts File

En el dispositivo con Ansible (*Network Automation Docker*) se debe ingresar al *path* ***/etc/ansible*** y editar el archivo **hosts** mediante *nano* para que quede de la siguiente manera:

```
root@ANSIBLE:~# cd /etc/ansible/
root@ANSIBLE:/etc/ansible#
root@ANSIBLE:/etc/ansible# ls
ansible.cfg  hosts  roles
root@ANSIBLE:/etc/ansible#
root@ANSIBLE:/etc/ansible# nano hosts
```

***Figura D – 5 Creación del archivo* host**
Fuente: *Autor*

Una vez creado el archivo *Hosts*, es necesario configurarlo así:

```
[Spine]
192.168.10.3

[Leaf1]
192.168.10.2

[Leaf2]
192.168.10.4

[cumulus:children]
Spine
Leaf1
Leaf2

[cumulus:vars]
ansible_connection=ssh
ansible_ssh_pass=CumulusLinux!
ansible_user=cumulus
ansible_ssh_key_private_file=home/cumulus/.ssh/id_rsa
```

Script 71 Configuración de Host File (Inventory File) – Ansible
Fuente: *Autor*

En el archivo Hosts se especifica el direccionamiento IPv4 de la red de Administración de los equipos a configurar mediante Ansible, así como el establecimiento de conexión SSH.

Llaves SSH en *NetAdmin Docker* (Ansible)

Ansible, para enviar las configuraciones a cada equipo usa SSH como protocolo de transporte, por lo que, para facilitar el proceso de comunicación, se generaron llaves en cada Cumulus Linux (en el *Spine* y en los dos *Leaf*), pero Ansible debe tener dicha relación de llaves SSH de forma activa.

Para lograr ello, se debe incluir estos comandos, una sola vez, en Ansible:

```
root@ANSIBLE:~# ssh-keyscan -H 192.168.10.2 >> ~/.ssh/known_host
# 192.168.10.2:22 SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u8
# 192.168.10.2:22 SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u8
# 192.168.10.2:22 SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u8
root@ANSIBLE:~#
root@ANSIBLE:~# ssh-keyscan -H 192.168.10.3 >> ~/.ssh/known_host
# 192.168.10.3:22 SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u8
# 192.168.10.3:22 SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u8
# 192.168.10.3:22 SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u8
root@ANSIBLE:~#
root@ANSIBLE:~# ssh-keyscan -H 192.168.10.4 >> ~/.ssh/known_host
# 192.168.10.4:22 SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u8
# 192.168.10.4:22 SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u8
# 192.168.10.4:22 SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u8
```

Script 72 Obtención de llaves SSH entre Ansible y dispositivos a configurar (Cumulus Linux)
Fuente: *Autor*

En caso que no se desee verificar la conexión SSH mediante las llaves, se puede editar el archivo **ansible.cfg** del *path* ***/etc/ansible*** mediante *nano* e incluir la siguiente línea de comando bajo [defaults]:

```
  GNU nano 2.5.3                File: ansible.cfg

# config file for ansible -- https://ansible.com/
# ===============================================

# nearly all parameters can be overridden in ansible-playbook
# or with command line flags. ansible will read ANSIBLE_CONFIG,
# ansible.cfg in the current working directory, .ansible.cfg in
# the home directory or /etc/ansible/ansible.cfg, whichever it
# finds first

[defaults]

host_key_checking=false
```

Script 73 Edición del archivo ansible.cfg para evitar el chequeo de las llaves SSH
Fuente: *Autor*

Una vez guardado el archivo, ejecute el comando **ansible –m ping all** y visualice que la conexión sea exitosa hacia el SPINE (192.168.10.3) y los dos LEAF (192.168.10.2 y 192.168.10.4):

```
root@ANSIBLE:/etc/ansible# ansible -m ping all
192.168.10.2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
192.168.10.3 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
192.168.10.4 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
root@ANSIBLE:/etc/ansible#
```

Figura D – 6 Prueba de conectividad desde Ansible a los equipos a configurar (Cumulus Linux)
Fuente: *Autor*

Playbook VXLAN.yml

El *playbook* de Ansible permite definir las tareas a realizar de forma automatizada en cada equipo a ser configurado.

Para el caso del PoC, se creará un **archivo de extensión yml con nombre VXLAN** en el *path* ***/etc/ansible***, ya que en él se definirán todas las configuraciones necesarias para tener VXLAN en la infraestructura.

Configuración de VXLAN.yml (*Playbook*)

Un *playbook* enfocado a automatizar equipos con Cumulus Linux empieza con tres guiones (---). Se emplearán los comandos de NCLU (*Network Command Line* Utility) para las configuraciones pero sin **net**.

Tarea 1: Direccionamiento IPv4

```
 GNU nano 2.5.3                File: VXLAN.yml

---
- hosts: Spine
  tasks:
    - name: Direccionamiento IP Spine
      nclu:
          commands:
          - add loopback lo ip address 2.2.2.2/32
          - add interface swp2 ip address 12.12.12.2/30
          - add interface swp3 ip address 23.23.23.2/30
          - commit

- hosts: Leaf1
  tasks:
    - name: Direccionamiento IP Leaf1
      nclu:
          commands:
          - add loopback lo ip address 1.1.1.1/32
          - add interface swp2 ip address 12.12.12.1/30
          - commit

- hosts: Leaf2
  tasks:
    - name: Direccionamiento IP Leaf2
      nclu:
          commands:
          - add loopback lo ip address 3.3.3.3/32
          - add interface swp3 ip address 23.23.23.1/30
          - commit
```

Script 74 Direccionamiento IP de equipos Cumulus Linux (NCLU) mediante Ansible Playbook
Fuente: *Autor*

Tarea 2: Enrutamiento *Underlay* (OSPF de una sola área)

La siguiente tarea es la configuración de OSPF como protocolo de enrutamiento *Underlay* de VXLAN.

```
 GNU nano 2.5.3                File: VXLAN.yml


- hosts: Spine
  tasks:
      - name: Configuracion OSPF Spine
        nclu:
          commands:
          - add ospf router-id 2.2.2.2
          - add loopback lo ospf area 0
          - add interface swp2 ospf area 0
          - add interface swp3 ospf area 0
          - add interface swp2 ospf network broadcast
          - add interface swp3 ospf network broadcast
          - commit
```

```
- hosts: Leaf1
  tasks:
     - name: Configuracion OSPF Leaf1
       nclu:
         commands:
         - add ospf router-id 1.1.1.1
         - add loopback lo ospf area 0
         - add interface swp2 ospf area 0
         - add interface swp2 ospf network broadcast
         - commit

- hosts: Leaf2
  tasks:
     - name: Configuracion OSPF Leaf2
       nclu:
         commands:
         - add loopback lo ospf area 0
         - add interface swp3 ospf area 0
         - add interface swp3 ospf network broadcast
         - add ospf router-id 3.3.3.3
         - commit
```

Script 75 Configuración de OSPF en equipos Cumulus Linux (NCLU) mediante Ansible Playbook – Underlay
Fuente: *Autor*

Configuración de VXLAN

Tarea 3: SNV (Service Node Functionality de VXLAN) y VTEPs (Virtual Tunnel Endpoints).

VXLAN es una técnica *underlay-overlay* empleada en entornos de DCs modernos, pues permite extender el dominio L2 a través de una infraestructura L3 mediante túneles, cuyos extremos se denominan VTEPs, equipos que realizan el proceso de encapsulación y desencapsulación de VXLAN (*tunnel pairs*).

De igual manera, VXLAN elimina las limitaciones que tienen las VLANs en cuanto a capacidad de segmentación de la red, pues es posible generar más de 16 millones de segmentos L2.

```
 GNU nano 2.5.3              File: VXLAN.yml                        Mo

- hosts: Spine
  tasks:
    - name: Configure Service Node functionality, junto con Anycast IP
      nclu:
         commands:
         - add lnv service-node source 2.2.2.2
         - add lnv service-node anycast-ip 2.2.2.2
         - commit

- hosts: Leaf1
  tasks:
     - name: Configure VTEP y Service Node IP en Leaf 1
       nclu:
         commands:
         - add loopback lo vxrd-src-ip 1.1.1.1
         - add loopback lo vxrd-svcnode-ip 2.2.2.2
         - commit
- hosts: Leaf2
  tasks:
     - name: Configure VTEP y Service Node IP en Leaf 2
       nclu:
         commands:
         - add loopback lo vxrd-src-ip 3.3.3.3
         - add loopback lo vxrd-svcnode-ip 2.2.2.2
         - commit
```

Script 76 Configuración de VXLAN (SNV y VTEP) en el Playbook VXLAN.yml
Fuente: *Autor*

Un *Leaf* dentro de la terminología de VXLAN es el equipo encargado de conectar los servidores en un DC, mientras los SPINES son los encargados de la conexión *underlay* entre cada *Leaf*.

Para que exista una mejor coordinación en la creación de los túneles, existe lo que se conoce como *Service Node*, el cual es el equipo que permite que el *fabric* L3 funcione adecuadamente. En el caso de la topología, sería el SPINE con dirección Ipv4 2.2.2.2 en su interfaz *Loopback*.

Cabe decir que en este PoC se plantea una configuración mediante VXRD en lugar de EVPN para VXLAN.

Tarea 4: Creación de VLAN10 en los LEAVES

Con el fin de extender el dominio L2 sobre un *fabric* L3, más allá del túnel VXLAN, se crea una misma VLAN, para el caso del PoC, la VLAN10 bajo la red 10.10.10.0/24. Las interfaces de los *Leaves* que apuntan a los equipos finales, serán puertos de acceso asignados a esa VLAN.

```
- hosts: Leaf1
  tasks:
     - name: host VLAN configs
       nclu:
         commands:
         - add vlan 10 ip address 10.10.10.1/24
         - add interface swp6 bridge access 10
         - commit

- hosts: Leaf2
  tasks:
     - name: host VLAN configs
       nclu:
         commands:
         - add vlan 10 ip address 10.10.10.3/24
         - add interface swp6 bridge access 10
         - commit
```

Script 77 Configuración de VXLAN (Asignación de VLANs en Leaf1 y Leaf2) en el Playbook VXLAN.yml
Fuente: *Autor*

Tarea 5: Mapeo de VXLAN con VLAN y VTEP (VNID 1010)

En los *Leaves* se deberá generar una correspondencia entre VLANs y VXLAN mediante su identificador denominado VNID o VXLAN-ID, que para el PoC es 1010.

De igual forma, se establece el origen y fin del túnel VXLAN (*local-tunnelip* y *remoteip*).

```
- hosts: Leaf1
  tasks:
     - name: VLAN-VxLAN Mapping
       nclu:
         commands:
         - add vxlan vni1010 vxlan id 1010
         - add vxlan vni1010 vxlan local-tunnelip 1.1.1.1
         - add vxlan vni1010 vxlan remoteip 3.3.3.3
         - add vxlan vni1010 bridge access 10
         - commit

- hosts: Leaf2
  tasks:
     - name: VLAN-VxLAN Mapping
       nclu:
         commands:
         - add vxlan vni1010 vxlan id 1010
         - add vxlan vni1010 bridge access 10
         - add vxlan vni1010 vxlan local-tunnelip 3.3.3.3
         - add vxlan vni1010 vxlan remoteip 1.1.1.1
         - commit
```

Script 78 Configuración de VXLAN (Mapeo VLAN-ID a VNID) en el Playbook VXLAN.yml
Fuente: *Autor*

Corrida (Play) de *Playbook* VXLAN.yml

Una de las ventajas de contar con Ansible, es su capacidad de monitoreo y observación de procesos.

Para correr todas las tareas configuradas en el *playbook* de Ansible, use el comando **ansible-playbook VXLAN.yml** (VXLAN.yml es el nombre dado en el PoC al playbook) dentro del path **/etc/ansible**.

Se verá el avance del proceso de configuración de cada tarea

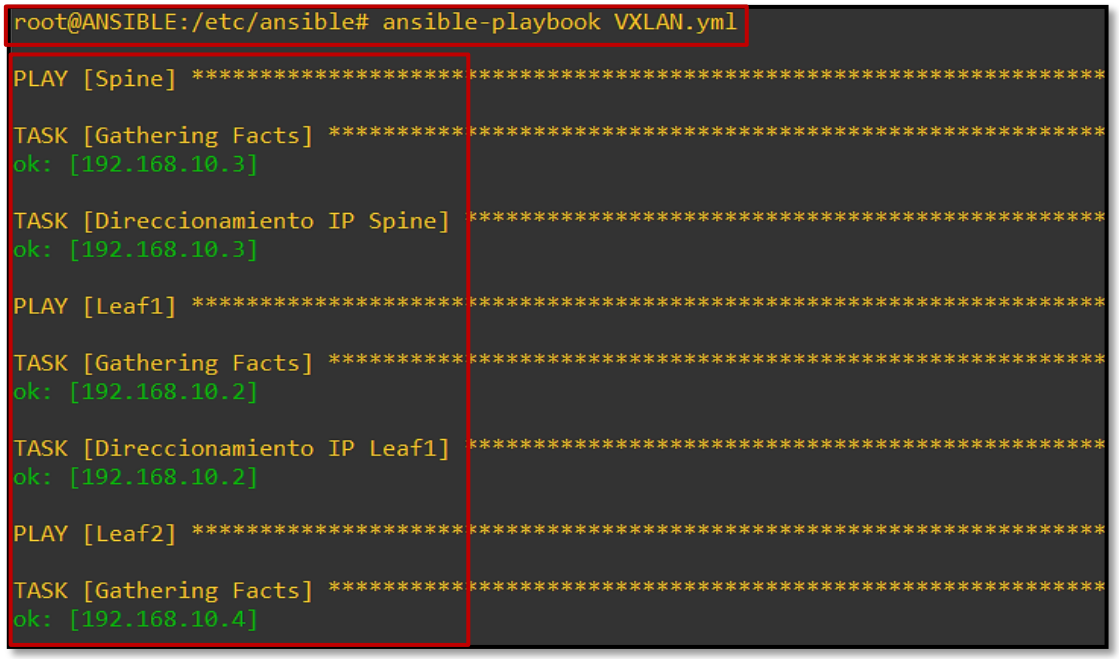


Figura D – 7 Corrida de Ansible Playbook para la configuración programática de VXLAN
Fuente: *Autor*

Al finalizar debe ver el reporte como OK y sin errores en cada equipo:

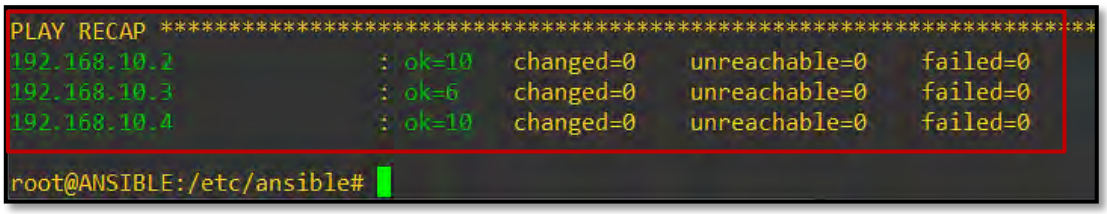


Figura D – 8 Resumen del resultado de la corrida de Ansible Playbook
Fuente: *Autor*

No se debe olvidar la activación de servicios para VXLAN y enrutamiento en SPINE, LEAF1 y LEAF2 (*Free Range Routing* o FRR y VXRD).

```
cumulus@cumulus:~$ sudo systemctl restart frr.service
[sudo] password for cumulus:
cumulus@cumulus:~$ sudo systemctl restart vxrd.service
```

Figura D – 9 Activación de servicios FRR y VXRD en Cumulus Linux
Fuente: *Autor*

Configuración de hosts (vPCs) y prueba de conectividad de extremo a extremo.

```
VPCS-1> ip 10.10.10.11/24
Checking for duplicate address...
PC1 : 10.10.10.11 255.255.255.0

VPCS-1> ping 10.10.10.33
84 bytes from 10.10.10.33 icmp_seq=1 ttl=64 time=2.779 ms
84 bytes from 10.10.10.33 icmp_seq=2 ttl=64 time=1.618 ms
84 bytes from 10.10.10.33 icmp_seq=3 ttl=64 time=2.561 ms
84 bytes from 10.10.10.33 icmp_seq=4 ttl=64 time=1.448 ms
84 bytes from 10.10.10.33 icmp_seq=5 ttl=64 time=1.725 ms
```

```
VPCS-2>
VPCS-2> ip 10.10.10.33/24
Checking for duplicate address...
PC1 : 10.10.10.33 255.255.255.0

VPCS-2> ping 10.10.10.11
84 bytes from 10.10.10.11 icmp_seq=1 ttl=64 time=2.620 ms
84 bytes from 10.10.10.11 icmp_seq=2 ttl=64 time=2.079 ms
84 bytes from 10.10.10.11 icmp_seq=3 ttl=64 time=2.946 ms
84 bytes from 10.10.10.11 icmp_seq=4 ttl=64 time=1.607 ms
84 bytes from 10.10.10.11 icmp_seq=5 ttl=64 time=1.570 ms
```

Figura D – 10 Prueba de conectividad de extremo a extremo
Fuente: *Autor*

Para verificar que los protocolos (OSPF y VXLAN) en SPINE y LEAF1 están funcionando bien respectivamente:

SPINE

```
cumulus@cumulus:~$
cumulus@cumulus:~$ net show route
show ip route
=============
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR,
       > - selected route, * - FIB route

O>* 1.1.1.1/32 [110/100] via 12.12.12.1, swp2, 00:06:33
O   2.2.2.2/32 [110/0] is directly connected, lo, 00:08:48
C>* 2.2.2.2/32 is directly connected, lo, 00:08:50
O>* 3.3.3.3/32 [110/100] via 23.23.23.1, swp3, 00:05:28
O   12.12.12.0/30 [110/100] is directly connected, swp2, 00:06:45
C>* 12.12.12.0/30 is directly connected, swp2, 00:08:50
O   23.23.23.0/30 [110/100] is directly connected, swp3, 00:05:46
C>* 23.23.23.0/30 is directly connected, swp3, 00:08:50
C>* 192.168.10.0/24 is directly connected, eth0, 00:08:50
```

Figura D – 11 Prueba de conectividad de extremo a extremo
Fuente: *Autor*

Leaf1:

```
cumulus@cumulus:~$ net show bridge macs

VLAN      Master  Interface  MAC               TunnelDest  State      Flags  LastSeen
--------  ------  ---------  ----------------  ----------  ---------  -----  --------
10        bridge  bridge     0c:1c:f6:b2:6c:06             permanent         00:15:39
10        bridge  swp6       00:50:79:66:68:00                               00:03:31
10        bridge  vni1010    00:50:79:66:68:01                               00:03:31
10        bridge  vni1010    0c:1c:f6:56:82:06                               00:04:32
untagged          vni1010    00:00:00:00:00:00  3.3.3.3     permanent  self   00:08:13
untagged          vni1010    00:50:79:66:68:01  3.3.3.3                 self   00:04:33
untagged          vni1010    0c:1c:f6:56:82:06  3.3.3.3                 self   00:04:49
untagged  bridge  swp6       0c:1c:f6:b2:6c:06             permanent         00:15:39
untagged  bridge  vni1010    16:20:2f:22:b5:65             permanent         00:15:31
cumulus@cumulus:~$
```

Figura D – 12 Aprendizaje de direccionamiento MAC mediante ARP de extremo a extremo (L2 sobre L3)
Fuente: *Autor*

Finalmente, se realiza una captura con Wireshark mientras hay un ping extendido entre vPC1 y vPC2.

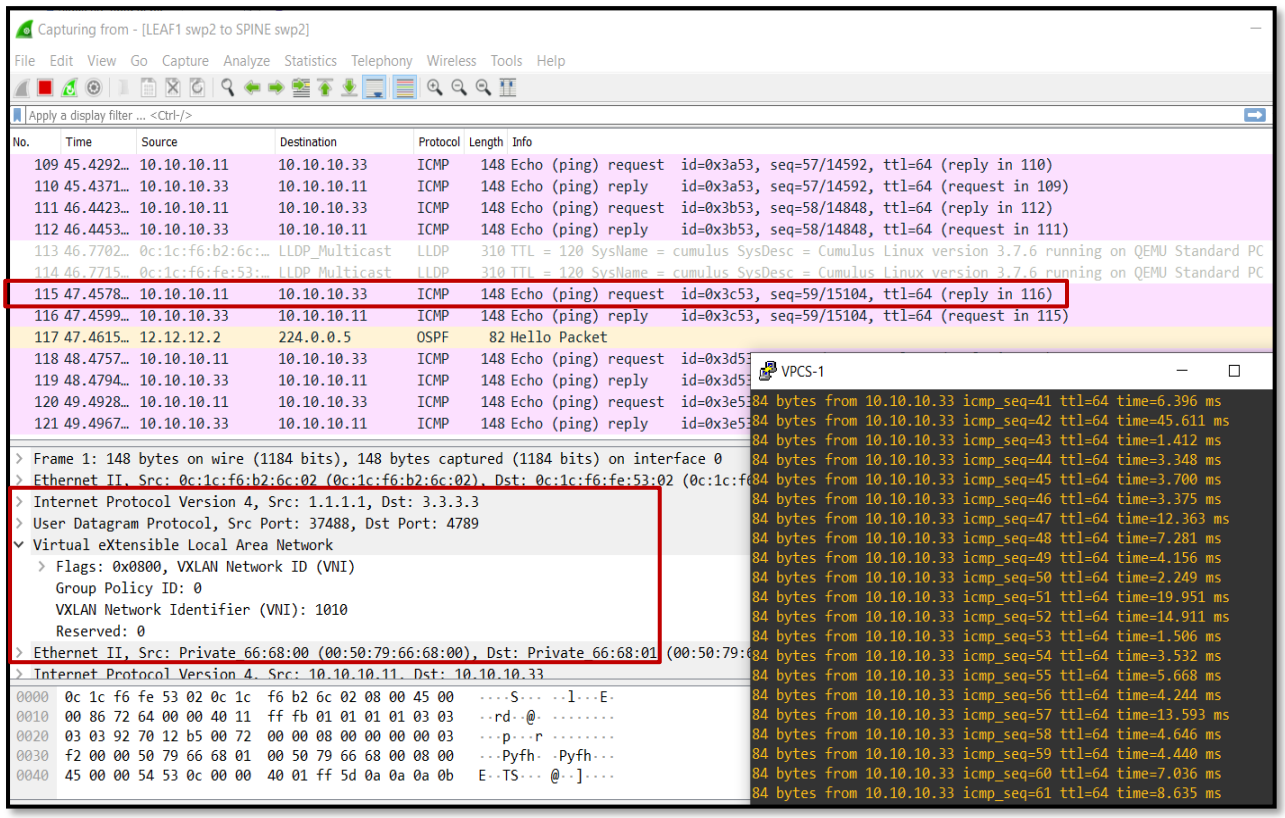


Figura D – 13 Captura Wireshark: Encapsulación VXLAN (VNID-VLAN mapping)
Fuente: *Autor*

*Nota: Al ser el *appliance Network Automation* de GNS3 un *Docker*, cada vez que se reinicie el emulador, pierde su contenido, por lo que es mejor guardar la configuración de los archivos de ANSIBLE en un bloc de notas adicional o hacerlo persistente.

Para complementar este estudio y análisis, el *paper* titulado "*VXLAN-IPSec Dual-Overlay as a Security Technique in Virtualized Datacenter Environments*" (Vaca & Salazar-Chacón, 2020), enfatiza en la parte tanto teórica como práctica de VXLAN, pero mediante un enfoque de dotar de mayor seguridad en los datos en tránsito de este tipo de tecnología *underlay-overlay* a través de IPSec.

Anexo E: Pruebas de Concepto de Netmiko y Napalm – Telemetría

Con el fin de poner a prueba a Netmiko y Napalm como herramientas para programabilidad en Redes, se realizó dos PoC con equipos tradicionales Cisco (IOS) reales y así evidenciar la interacción entre el mundo de la automatización con las redes tal como las hemos conocido hasta el momento de escribir la presente tesis.

Son PoCs que ejemplifican el uso de este tipo de herramientas como librerías de *Python*, comprobando así su eficacia en entornos híbridos (Infraestructuras tradicionales con Programabilidad) pero enfocados en el camino hacia SDN e *Infrastructure-as-Code*.

Envío de Comandos a través de Netmiko a una infraestructura tradicional

Tal como se conceptualizó en ***Netmiko, NAPALM y Nornir: Herramientas tipo Python Library***, Netmiko es una herramienta basada en Paramiko que permite la conexión sencilla de un equipo "Central de configuraciones" a una infraestructura subyacente mediante SSH y así enviarle configuraciones de manera programática y automatizada.

Para este PoC, se utilizará tres de equipos Cisco tradicionales:

- Router Cisco 2851
- L3 Switch Cisco 3560
- L2 Switch Cisco 2950 para conexión entre la Central de Configuraciones y la infraestructura

El objetivo de este PoC es el de crear un *script* en Python empleando la librería de Netmiko y enviar comandos de forma programática, desde el *Configuration-Server* a los dos equipos tradicionales, para así crear interfaces *loopbacks* y presentar la tabla de interfaces y direccionamiento IP de cada equipo confirmado la creación de dichas interfaces en la Central de configuraciones.

La topología lógica es la siguiente:

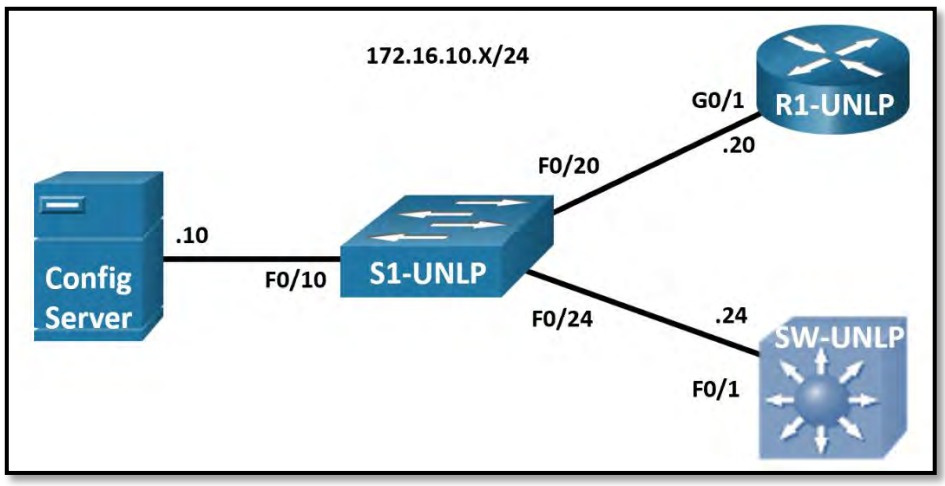


Figura E - 1 Topología Lógica PoC de Netmiko
Fuente: *Autor*

La topología física es la siguiente:

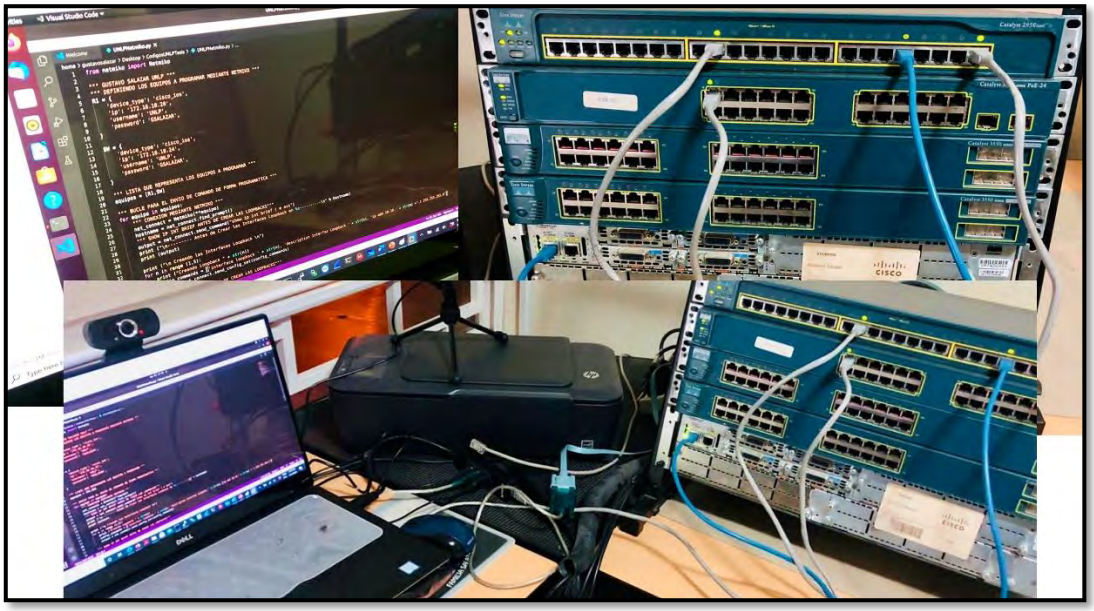


Figura E - 2 Topología Física PoC de Netmiko
Fuente: *Autor*

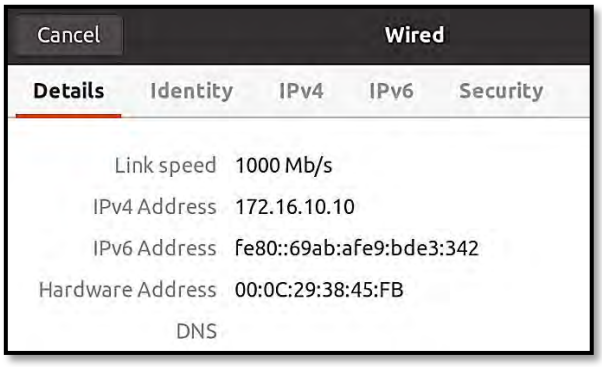
El primer paso es configurar las direcciones IP de los equipos con el fin de que tengan conexión entre ellos:

```
SW-UNLP(config)#ip routing
SW-UNLP(config)#int f0/1
SW-UNLP(config-if)#descrip SW->S1
SW-UNLP(config-if)#no switchport
SW-UNLP(config-if)#ip add 172.16.10.24 255.255.255.0
SW-UNLP(config-if)#no shut
```

```
R1-UNLP(config)#int g0/1
R1-UNLP(config-if)#descript R1->S1
R1-UNLP(config-if)#ip add 172.16.10.20 255.255.255.0
R1-UNLP(config-if)#no shut
```

Script 79 Configuración IP de la Infraestructura para conectividad SSH con Config-Server (Netmiko)
Fuente: *Autor*

El *Config-Server* está montado en un equipo con Ubuntu 20.04 bajo la dirección IP 172.16.10.10/24, el cual tiene conexión con toda la infraestructura de red:



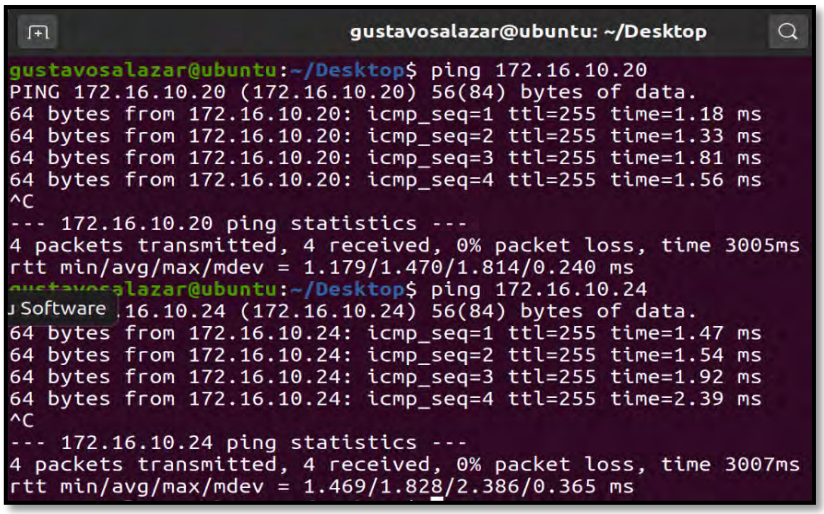


Figura E - 3 Configuración IP del Config-Server y prueba de conectividad
Fuente: *Autor*

El siguiente paso es la configuración de SSH en los equipos de red (R1-UNLP y SW-UNLP).

```
R1-UNLP(config)#ip domain-name unlp.edu.ec
R1-UNLP(config)#enable secret UNLP-GSALAZAR
R1-UNLP(config)#crypto key generate rsa
The name for the keys will be: R1-UNLP.unlp.edu.ec
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
% Generating 1024 bit RSA keys, keys will be non-exportable...[OK]
```

```
R1-UNLP(config)#username UNLP pass GSALAZAR
R1-UNLP(config)#username UNLP privilege 15
R1-UNLP(config)#
R1-UNLP(config)#line vty 0 4
R1-UNLP(config-line)#transport input ssh
R1-UNLP(config-line)#login local
R1-UNLP(config-line)#
R1-UNLP(config-line)#exit
```

Figura E - 4 Configuración SSH en RI-UNLP
Fuente: *Autor*

En el *Config-Server*, se debe instalar mediante pip3 Netmiko. Cabe decir que en Ubuntu 20.04 se tiene instalado Python 3.8.5 y se realizó previamente un *update* (**apt-get update**).

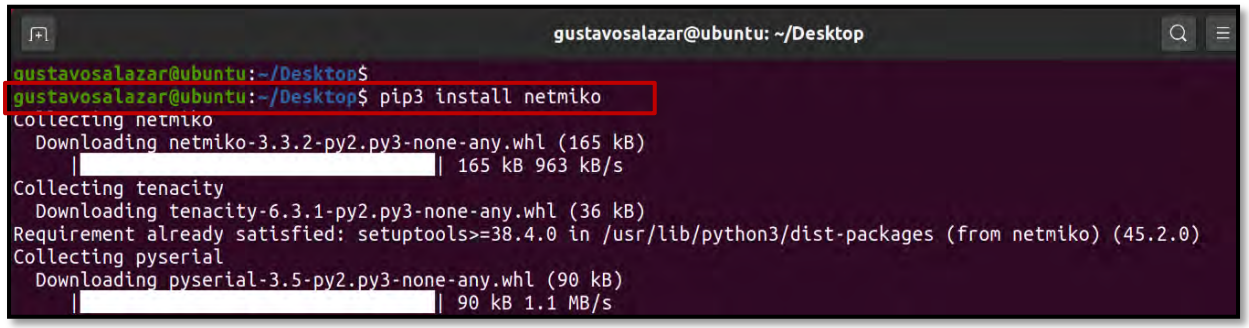


Figura E - 5 Instalación de Netmiko mediante pip3 en Config-Server
Fuente: *Autor*

En *Visual Studio Code* o cualquier IDE de su elección, se escribe el siguiente *script* para cumplir el objetivo de crear interfaces Loopbacks (bajo el formato 10.10.X.1/24, donde X representa el número de la interfaz) y presentar la Tabla de direccionamiento IP de las interfaces en los equipos antes y después de correr el código.

```
from netmiko import Netmiko

""" GUSTAVO SALAZAR UNLP """
""" DEFINIENDO LOS EQUIPOS A PROGRAMAR MEDIANTE NETMIKO """
R1 = {
    'device_type': 'cisco_ios',
    'ip': '172.16.10.20',
    'username': 'UNLP',
    'password': 'GSALAZAR',
}

SW = {
    'device_type': 'cisco_ios',
    'ip': '172.16.10.24',
    'username': 'UNLP',
    'password': 'GSALAZAR',
}

""" LISTA QUE REPRESENTA LOS EQUIPOS A PROGRAMAR """
equipos = [R1,SW]
```

```
""" BUCLE PARA EL ENVIO DE COMANDO DE FORMA PROGRAMATICA """
for equipo in equipos:
    """ CONEXION MEDIANTE NETMIKO """
    net_connect = Netmiko(**equipo)
    hostname = net_connect.find_prompt()
    """ SHOW IP INT BRIEF ANTES DE CREAR LAS LOOPBACKS"""
    output = net_connect.send_command("show ip int brief | e ass")
    print ("\n--------- Antes de Crear las Interfaces Loopback en %s----------\n" % hostname)
    print (output)

    print ("\n Creando las Interfaces Loopback \n")
    for n in range (1,5):
        print ("Creando Loopback " + str(n))
        config_commands = ['interface loopback ' + str(n), 'description Interfaz Loopback ' + str(n),
        'ip add 10.10.' + str(n) +'.1 255.255.255.0']
        output = net_connect.send_config_set(config_commands)

    """ SHOW IP INT BRIEF ANTES DE CREAR LAS LOOPBACKS"""
    print ("\n--------- Resultado Despues de Crear las Interfaces Loopback en %s----------\n" % hostname)
    output = net_connect.send_command("show ip int brief | e ass")
    print (output)

    """ DESCONEXION DE NETMIKO """

    net_connect.disconnect()
```

Script 80 Configuración de script (UNLPNetmiko.py) para envío de comandos a equipos Cisco IOS
Fuente: *Autor*

El resultado de correr el *script* mediante el comando **python3 UNLPNetmiko.py** es:

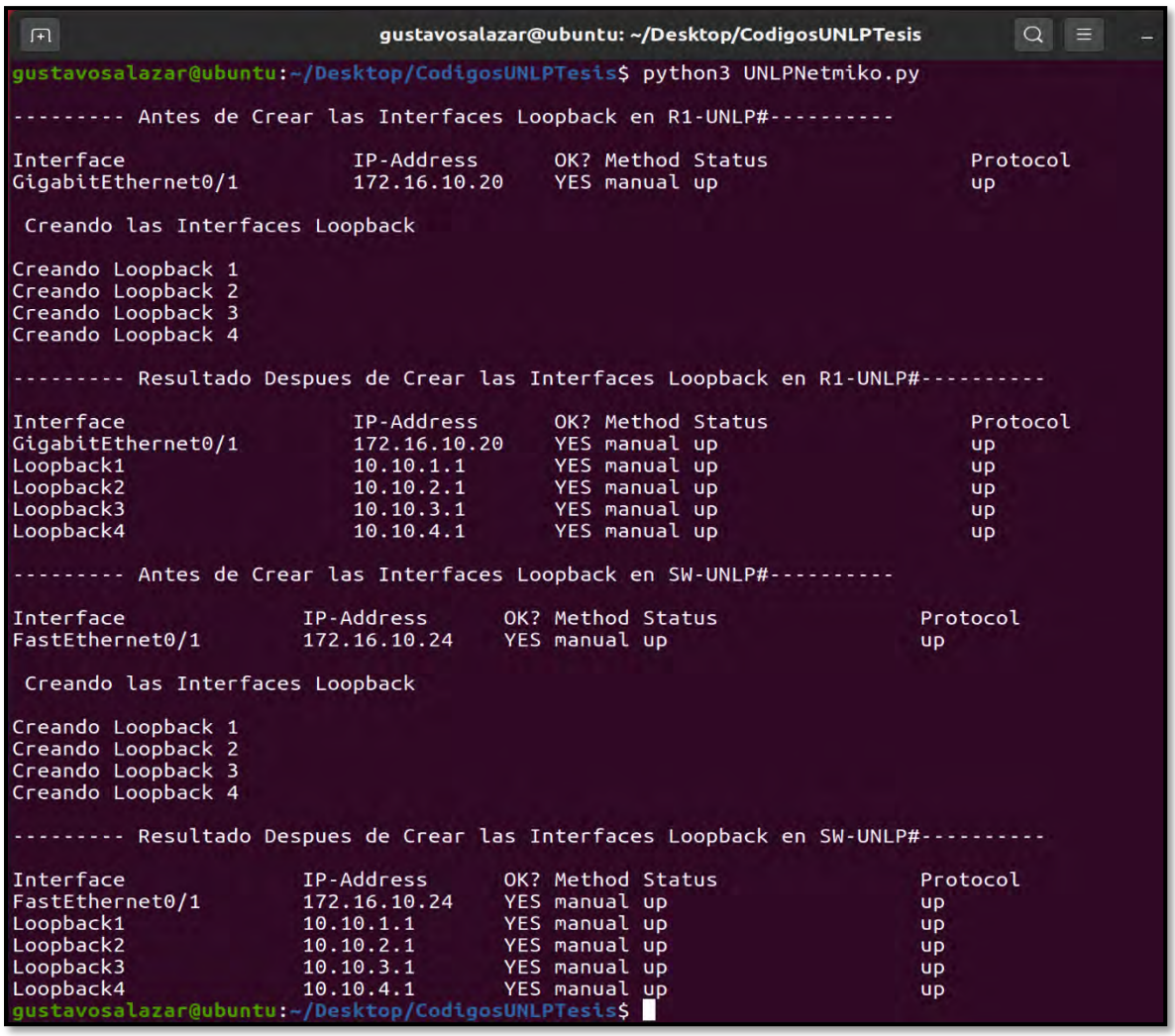


Figura E - 6 Resultado de PoC Netmiko
Fuente: *Autor*

Resultado que se corrobora con el *output* de **show ip int brief** en SW-UNLP:

```
SW-UNLP#show ip int brief | e ass
Interface              IP-Address      OK? Method Status
FastEthernet0/1        172.16.10.24    YES manual up
Loopback1              10.10.1.1       YES manual up
Loopback2              10.10.2.1       YES manual up
Loopback3              10.10.3.1       YES manual up
Loopback4              10.10.4.1       YES manual up
```

Figura E - 7 Corroboración de Resultado de PoC Netmiko
Fuente: *Autor*

Uso de NAPALM para configurar un equipo desde un Archivo y generar procesos de Telemetría y comparación de cambios de configuración

NAPALM (*Network Automation and Programmability Abstraction Layer with Multivendor support*) es una librería de Python que implementa un conjunto de funciones para interactuar con equipos de red de diversas marcas a través de una API unificada a través de Netmiko

Para este PoC, se comprobará la factibilidad de uso de NAPALM para configurar el *router* Cisco 2851 usado en el PoC de Netmiko, pero a través de un archivo con los comandos de dicha configuración sencilla (activación de OSPF de área única y la creación de una interfaz Loopback adicional a las ya creadas en el PoC de Netmiko).

La topología y los equipos son los mismos del PoC de Netmiko (*Figura E - 1 Topología Lógica PoC de Netmiko*), pero solamente R1-UNLP será configurado mediante NAPALM.

Para tener claro el escenario, se debe tomar en cuenta estos datos de la infraestructura:

- Usuario SSH: UNLP / Contraseña SSH: GSALAZAR
- Dir. IP R1-UNLP: 172.16.10.20/24
- Dir. IP *Config-Server*: 172.16.10.10/24

El primer paso es la instalación de NAPALM en el *Config-Server* (equipo Ubuntu 20.04) luego de haber realizado un *update* a Ubuntu:

```
gustavosalazar@ubuntu:~/Desktop/CodigosUNLPTesis$ pip3 install napalm
Collecting napalm
  Downloading napalm-3.2.0-py2.py3-none-any.whl (230 kB)
     |                                | 230 kB 807 kB/s
Requirement already satisfied: netmiko>=3.1.0 in /home/gustavosalazar/.local/lib/python3.8
/site-packages (from napalm) (3.3.2)
```

Figura E - 8 Instalación de NAPALM mediante pip3 en Config-Server
Fuente: *Autor*

Para probar que la conexión entre el equipo donde correrá el *script* en Python con NAPALM y el *router* R1-UNLP es correcto, se usó un código de (Álvarez, 2020) pero con las credenciales SSH del equipo, así como su dirección IP.

Welcome | UNLPNetmiko.py | CheckNAPALM.py

home > gustavosalazar > Desktop > CodigosUNLPTesis > CheckNAPALM.py > main

```python
import napalm

def main():

    driver_ios = napalm.get_network_driver("ios")

    ios_router = driver_ios(
    hostname = "172.16.10.20",
    username = "UNLP",
    password = "GSALAZAR"
    )

    print("Connecting to IOS Router...")
    ios_router.open()
    print("Checking IOS Router Connection Status:")
    print(ios_router.is_alive())

    ios_router.close()
    print("Test Completed")

if __name__ == "__main__":
    main()
```

Script 81 Script para comprobar conexión NAPALM-Equipo
Basado de (Álvarez, 2020)

El resultado de la ejecución del *script* fue exitoso:

```
gustavosalazar@ubuntu:~/Desktop/CodigosUNLPTesis$ ls
CheckNAPALM.py  UNLPNetmiko.py
gustavosalazar@ubuntu:~/Desktop/CodigosUNLPTesis$ python3 CheckNAPALM.py
Connecting to IOS Router...
Checking IOS Router Connection Status:
{'is_alive': True}
Test Completed
gustavosalazar@ubuntu:~/Desktop/CodigosUNLPTesis$
```

Figura E - 9 Ejecución exitosa de CheckNAPALM.py
Fuente: *Autor*

Con la comprobación exitosa del funcionamiento de NAPALM, se creará el archivo de extensión .cfg con los comandos de R1-UNLP que se incluirán (OSPF y creación de nueva Interfaz Loopback):

Welcome | UNLPNetmiko.py | CheckNAPALM.py | R1-UNLP-Config.cfg

home > gustavosalazar > Desktop > CodigosUNLPTesis > R1-UNLP-Config.cfg

```
1  interface Lo10
2   description Loopback creada usando NAPALM
3   ip add 192.168.100.1 255.255.255.0
4   ip ospf net point-to-point
5
6  router ospf 1
7   router-id 1.1.1.1
8   network 172.16.10.0 0.0.0.255 area 0
9   network 192.168.100.0 0.0.0.255 area 0
10  passive-interface Lo10
11
12  do wr
13 end
14
```

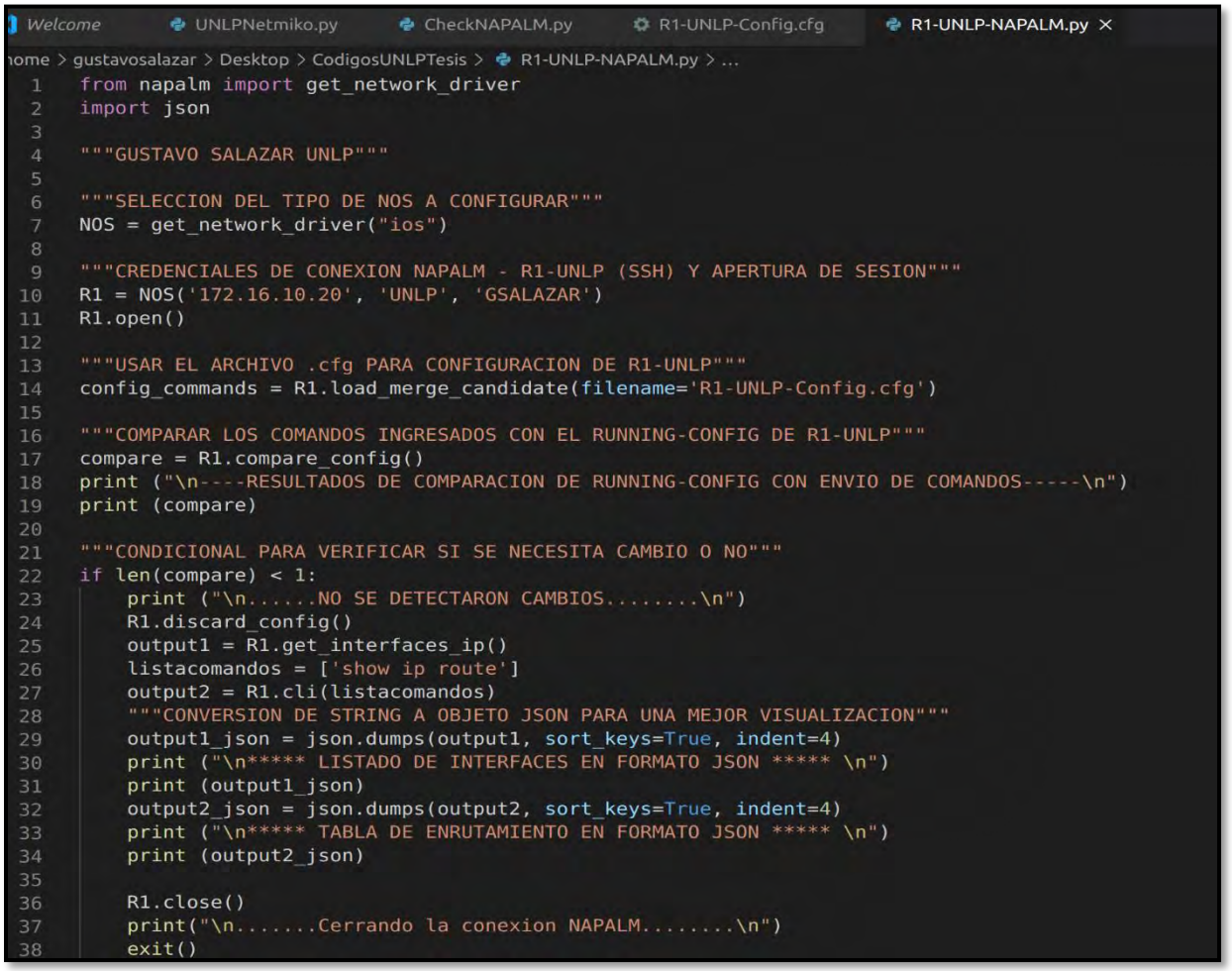
Script 82 Configuración mediante comandos Cisco IOS para enviar mediante NAPALM (archivo .cfg)
Fuente: Elaboración Propia

Antes de empezar a configurar el *script*, para que NAPALM interactúe correctamente con R1-UNLP, se debe activar el servicio de SCP en dicho equipo (R1-UNLP tiene la configuración del PoC de Netmiko) y así enviar de forma segura las configuraciones.

```
R1-UNLP(config)#ip scp server enable
R1-UNLP(config)#
```

Figura E - 10 Activación de SCP en R1-UNLP para la buena comunicación con NAPALM
Fuente: Elaboración Propia

El *script* en Python usando NAPALM es el siguiente:



```python
from napalm import get_network_driver
import json

"""GUSTAVO SALAZAR UNLP"""

"""SELECCION DEL TIPO DE NOS A CONFIGURAR"""
NOS = get_network_driver("ios")

"""CREDENCIALES DE CONEXION NAPALM - R1-UNLP (SSH) Y APERTURA DE SESION"""
R1 = NOS('172.16.10.20', 'UNLP', 'GSALAZAR')
R1.open()

"""USAR EL ARCHIVO .cfg PARA CONFIGURACION DE R1-UNLP"""
config_commands = R1.load_merge_candidate(filename='R1-UNLP-Config.cfg')

"""COMPARAR LOS COMANDOS INGRESADOS CON EL RUNNING-CONFIG DE R1-UNLP"""
compare = R1.compare_config()
print ("\n----RESULTADOS DE COMPARACION DE RUNNING-CONFIG CON ENVIO DE COMANDOS-----\n")
print (compare)

"""CONDICIONAL PARA VERIFICAR SI SE NECESITA CAMBIO O NO"""
if len(compare) < 1:
    print ("\n......NO SE DETECTARON CAMBIOS........\n")
    R1.discard_config()
    output1 = R1.get_interfaces_ip()
    listacomandos = ['show ip route']
    output2 = R1.cli(listacomandos)
    """CONVERSION DE STRING A OBJETO JSON PARA UNA MEJOR VISUALIZACION"""
    output1_json = json.dumps(output1, sort_keys=True, indent=4)
    print ("\n***** LISTADO DE INTERFACES EN FORMATO JSON ***** \n")
    print (output1_json)
    output2_json = json.dumps(output2, sort_keys=True, indent=4)
    print ("\n***** TABLA DE ENRUTAMIENTO EN FORMATO JSON ***** \n")
    print (output2_json)

    R1.close()
    print("\n.......Cerrando la conexion NAPALM........\n")
    exit()
```

```python

""" CONFIRMAR LOS CAMBIOS """
try:
    eleccion = input("\n Desea aplicar los cambios? s/n \n")
except NameError:
    eleccion = input("\n Desea aplicar los cambios? s/n \n")

if eleccion == "s":
    print ("\n..........CONFIGURANDO R1-UNLP............")
    R1.commit_config()
    output1 = R1.get_interfaces_ip()
    output1_json = json.dumps(output1, sort_keys=True, indent=4)

    output2 = R1.get_route_to('172.16.10.0/24')
    #listacomandos = ['show ip route']
    #output2 = R1.cli(listacomandos)
    output2_json = json.dumps(output2, sort_keys=True, indent=4)

    print (output1_json)
    #print (output2_json)
    print (output2_json)
else:
    print ("\n..........DESCARTANDO CONFIGURACION............")
    R1.discard_config()

"""CERRANDO LA CONEXION NAPALM"""
R1.close()
```

Script 83 Script en Python usando NAPALM para configurar OSPF, crear una interfaz Lo10 y comparar cambios en las configuraciones
Fuente: *Autor*

El resultado exitoso de la ejecución de ese *script* mediante el comando **R1-UNLP-NAPALM.py**, el cual permite confirmar si se desea enviar la configuración es:

```
gustavosalazar@ubuntu:~/Desktop/CodigosUNLPTesis$
gustavosalazar@ubuntu:~/Desktop/CodigosUNLPTesis$ python3 R1-UNLP-NAPALM.py

----RESULTADOS DE COMPARACION DE RUNNING-CONFIG CON ENVIO DE COMANDOS-----

+interface Lo10
+ description Loopback creada usando NAPALM
+ ip add 192.168.100.1 255.255.255.0
+ ip ospf net point-to-point
+router ospf 1
+ router-id 1.1.1.1
+ network 172.16.10.0 0.0.0.255 area 0
+ network 192.168.100.0 0.0.0.255 area 0
+ passive-interface Lo10

 Desea aplicar los cambios? s/n
s

.........CONFIGURANDO R1-UNLP............
{
    "GigabitEthernet0/1": {
        "ipv4": {
            "172.16.10.20": {
                "prefix_length": 24
            }
        }
    },
    "Loopback10": {
        "ipv4": {
            "192.168.100.1": {
                "prefix_length": 24
            }
        }
    }
}
{
    "172.16.10.0/24": [
        {
            "age": "",
            "current_active": true,
            "inactive_reason": "",
            "last_active": true,
            "next_hop": "",
            "outgoing_interface": "GigabitEthernet0/1",
            "preference": 0,
            "protocol": "connected",
            "protocol_attributes": {},
            "routing_table": "default",
            "selected_next_hop": true
        }
    ]
}
gustavosalazar@ubuntu:~/Desktop/CodigosUNLPTesis$
```

Figura E - 11 Ejecución exitosa de script con NAPALM
Fuente: *Autor*

Para tener más información sobre los métodos y funciones a usar con NAPALM, se puede ver este link:

https://napalm.readthedocs.io/en/latest/base.html

Las interfaces Loopback en R1-UNLP fueron eliminadas previo a la ejecución del *script*, así como se configuró en SW-UNLP OSPF para comprobar el funcionamiento del protocolo de enrutamiento, logrando resultados exitosos.

```
SW-UNLP#show ip route ospf
O    192.168.100.0/24 [110/2] via 172.16.10.20, 00:21:03,
SW-UNLP#
```

Figura E - 12 Funcionamiento correcto de OSPF en SW-UNLP
Fuente: *Autor*

Anexo F: Scripts para la Implementación de PoC NG-SDN

Durante el desarrollo del PoC de NG-SDN (Ver ***4.4.2 Topología del PoC de NG-SDN***), se requirió de ciertos *scripts* para que corra adecuadamente, *scripts* que definen parámetros, dependencias y la topología en sí.

De igual manera, la VM del PoC NG-SDN utiliza comandos ***make*** para su adecuado funcionamiento. Un resumen de los más importantes se aprecia en la siguiente tabla:

Tabla F-1 Comandos *make* usado en Poc NG-SDN

Comando make	Descripción
make pull-deps	Descarga y actualiza todas las dependencias de la VM
make p4-build	Compila y construye un programa P4
make start	Inicializa Mininet y contenedores ONOS
make stop	Para todos los contenedores
make reset	Para los contenedores y remueve cualquier estado asociado
make onos-cli	Permite acceder a ONOS CLI (contraseña: rocks, Ctrl+D para salir)
make onos-log	Muestra los logs de ONOS
make mn-cli	Permite acceder a Mininet CLI (Ctrl+D para salir)
make mn-log	Muestra los logs de Mininet
make app-build	Construye las app de ONOS
make app-reload	Instala y activa app de ONOS
make netcfg	Envía *netcfg.json* (configuración de red) a ONOS

Fuente: (Cascone, 2019)

Los parámetros de Mininet se establecieron en el siguiente *script:*

```
version: "3"

services:
  mininet:
    image: opennetworking/mn-stratum
    hostname: mininet
    container_name: mininet
    privileged: true
    tty: true
    stdin_open: true
    volumes:
      - ./tmp:/tmp
      - ./mininet:/mininet
```

```
    ports:
      - "50001:50001"
      - "50002:50002"
      - "50003:50003"
      - "50004:50004"
    entrypoint: "/mininet/topo.py"
  onos:
    image: onosproject/onos:2.2.2
    hostname: onos
    container_name: onos
    ports:
      - "8181:8181" # HTTP
      - "8101:8101" # SSH (CLI)
    volumes:
      - ./tmp/onos:/root/onos/apache-karaf-4.2.8/data/tmp
    environment:
      - ONOS_APPS=gui2,drivers.bmv2,lldpprovider,hostprovider
    links:
      - mininet
```

Script 84 Parámetros Mininet – Docker-compose.yml
Basado en (Cascone & Condon, 2020)

Mientras el *script* en Python de la topología del PoC de SDN es el siguiente:

```python
import argparse

from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.net import Mininet
from mininet.node import Host
from mininet.topo import Topo
from stratum import StratumBmv2Switch

CPU_PORT = 255


class IPv6Host(Host):

    def config(self, ipv6, ipv6_gw=None, **params):
        super(IPv6Host, self).config(**params)
        self.cmd('ip -4 addr flush dev %s' % self.defaultIntf())
        self.cmd('ip -6 addr flush dev %s' % self.defaultIntf())
        self.cmd('ip -6 addr add %s dev %s' % (ipv6, self.defaultIntf()))
        if ipv6_gw:
            self.cmd('ip -6 route add default via %s' % ipv6_gw)
        # Disable offload
        for attr in ["rx", "tx", "sg"]:
            cmd = "/sbin/ethtool --offload %s %s off" % (self.defaultIntf(), attr)
            self.cmd(cmd)

        def updateIP():
            return ipv6.split('/')[0]

        self.defaultIntf().updateIP = updateIP

    def terminate(self):
        super(IPv6Host, self).terminate()
```

```
class TutorialTopo(Topo):
    """2x2 fabric topology"""

    def __init__(self, *args, **kwargs):
        Topo.__init__(self, *args, **kwargs)

        # Leaves
        # gRPC port 50001
        leaf1 = self.addSwitch('leaf1', cls=StratumBmv2Switch, cpuport=CPU_PORT)
        # gRPC port 50002
        leaf2 = self.addSwitch('leaf2', cls=StratumBmv2Switch, cpuport=CPU_PORT)

        # Spines
        # gRPC port 50003
        spine1 = self.addSwitch('spine1', cls=StratumBmv2Switch, cpuport=CPU_PORT)
        # gRPC port 50004
        spine2 = self.addSwitch('spine2', cls=StratumBmv2Switch, cpuport=CPU_PORT)

        # Switch Links
        self.addLink(spine1, leaf1)
        self.addLink(spine1, leaf2)
        self.addLink(spine2, leaf1)
        self.addLink(spine2, leaf2)

        # IPv6 hosts attached to leaf 1
        h1a = self.addHost('h1a', cls=IPv6Host, mac="00:00:00:00:00:1A",
                           ipv6='2001:1:1::a/64', ipv6_gw='2001:1:1::ff')
        h1b = self.addHost('h1b', cls=IPv6Host, mac="00:00:00:00:00:1B",
                           ipv6='2001:1:1::b/64', ipv6_gw='2001:1:1::ff')
        h1c = self.addHost('h1c', cls=IPv6Host, mac="00:00:00:00:00:1C",
                           ipv6='2001:1:1::c/64', ipv6_gw='2001:1:1::ff')
        h2 = self.addHost('h2', cls=IPv6Host, mac="00:00:00:00:00:20",
                          ipv6='2001:1:2::1/64', ipv6_gw='2001:1:2::ff')
        self.addLink(h1a, leaf1)  # port 3
        self.addLink(h1b, leaf1)  # port 4
        self.addLink(h1c, leaf1)  # port 5
        self.addLink(h2, leaf1)  # port 6

        # IPv6 hosts attached to leaf 2
        h3 = self.addHost('h3', cls=IPv6Host, mac="00:00:00:00:00:30",
                          ipv6='2001:2:3::1/64', ipv6_gw='2001:2:3::ff')
        h4 = self.addHost('h4', cls=IPv6Host, mac="00:00:00:00:00:40",
                          ipv6='2001:2:4::1/64', ipv6_gw='2001:2:4::ff')
        self.addLink(h3, leaf2)  # port 3
        self.addLink(h4, leaf2)  # port 4


def main():
    net = Mininet(topo=TutorialTopo(), controller=None)
    net.start()
    CLI(net)
    net.stop()


if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description='Mininet topology script for 2x2 fabric with stratum_bmv2 and IPv6 hosts')
    args = parser.parse_args()
    setLogLevel('info')

    main()
```

Script 85 Topología para Mininet – PoC NG-SDN
Basado en (Cascone, 2019)

NG-SDN tiene la ventaja de usar modelos YANG para representar la infraestructura como código. En el caso del PoC de NG-SDN (***Topología del PoC de NG-SDN: Programación en P4, YANG, OpenConfig-gNMI (Telemetría) y ONOS***), el modelo YANG fue planteado en el Tutorial de NG-SDN para la ONF y se lo denominó ***yang/demo-port.yang,*** el cual se observa a continuación:

```
// A module is a self-contained tree of nodes
module demo-port {

// YANG Boilerplate
yang-version "1";
namespace "https://opennetworking.org/yang/demo";
prefix "demo-port";
description "Demo model for managing ports";
revision "2019-09-10" {
description "Initial version";
reference "1.0.0";
}

// Identities and Typedefs
identity SPEED {
description "base type for port speeds";
}

identity SPEED_10GB {
base SPEED;
description "10 Gbps port speed";
}

typedef port-number {
type uint16 {
range 1..32;
}
description "New type for port number that ensure the number is between 1 and 32, inclusive";
}

// Reusable groupings for port config and state
grouping port-config {
description "Set of configurable attributes / leaves";
leaf speed {
type identityref {
base demo-port:SPEED;
}
```

```
description "Configurable speed of a switch port";
}
}

grouping port-state {
description "Set of read-only state";
leaf status {
type boolean;
description "Number";
}
}

// Top-level model definition
container ports {
description "The root container for port configuration and state";
list port {
key "port-number";
description "List of ports on a switch";

leaf port-number {
type port-number;
description "Port number (maps to the front panel port of a switch); also the key for the port list";
}

// each individual will have the elements defined in the grouping
container config {
description "Configuration data for a port";
uses port-config; // reference to grouping above
}
container state {
config false; // makes child nodes read-only
description "Read-only state for a port";
uses port-state; // reference to grouping above
}
}
}
}
```

Script 86 Módulo YANG para PoC NG-SDN
Basado en (Cascone, 2019)

```
{
"devices": {
"device:leaf1": {
"basic": {
"managementAddress": "grpc://mininet:50001?device_id=1",
"driver": "stratum-bmv2",
"pipeconf": "org.onosproject.ngsdn-tutorial",
"locType": "grid",
"gridX": 200,
"gridY": 600
},
"fabricDeviceConfig": {
"myStationMac": "00:aa:00:00:00:01",
"isSpine": false
}
},
"device:leaf2": {
"basic": {
"managementAddress": "grpc://mininet:50002?device_id=1",
"driver": "stratum-bmv2",
"pipeconf": "org.onosproject.ngsdn-tutorial",
"locType": "grid",
"gridX": 800,
"gridY": 600
},
"fabricDeviceConfig": {
"myStationMac": "00:aa:00:00:00:02",
"isSpine": false
}
},
"device:spine1": {
"basic": {
"managementAddress": "grpc://mininet:50003?device_id=1",
"driver": "stratum-bmv2",
"pipeconf": "org.onosproject.ngsdn-tutorial",
"locType": "grid",
"gridX": 400,
"gridY": 400
},
```

```
"fabricDeviceConfig": {
"myStationMac": "00:bb:00:00:00:01",
"isSpine": true
}
},
"device:spine2": {
"basic": {
"managementAddress": "grpc://mininet:50004?device_id=1",
"driver": "stratum-bmv2",
"pipeconf": "org.onosproject.ngsdn-tutorial",
"locType": "grid",
"gridX": 600,
"gridY": 400
},
"fabricDeviceConfig": {
"myStationMac": "00:bb:00:00:00:02",
"isSpine": true
}
}
},
"ports": {
"device:leaf1/3": {
"interfaces": [
{
"name": "leaf1-3",
"ips": ["2001:1:1::ff/64"]
}
]
},
"device:leaf1/4": {
"interfaces": [
{
"name": "leaf1-4",
"ips": ["2001:1:1::ff/64"]
}
]
},
"device:leaf1/5": {
"interfaces": [
```

```
{
"name": "leaf1-5",
"ips": ["2001:1:1::ff/64"]
}
]
},
"device:leaf1/6": {
"interfaces": [
{
"name": "leaf1-6",
"ips": ["2001:1:2::ff/64"]
}
]
},
"device:leaf2/3": {
"interfaces": [
{
"name": "leaf2-3",
"ips": ["2001:2:3::ff/64"]
}
]
},
"device:leaf2/4": {
"interfaces": [
{
"name": "leaf2-4",
"ips": ["2001:2:4::ff/64"]
}
]
}
},
"hosts": {
"00:00:00:00:00:1A/None": {
"basic": {
"name": "h1a",
"locType": "grid",
"gridX": 100,
"gridY": 700
}
```

```
},
"00:00:00:00:00:1B/None": {
"basic": {
"name": "h1b",
"locType": "grid",
"gridX": 100,
"gridY": 800
}
},
"00:00:00:00:00:1C/None": {
"basic": {
"name": "h1c",
"locType": "grid",
"gridX": 250,
"gridY": 800
}
},
"00:00:00:00:00:20/None": {
"basic": {
"name": "h2",
"locType": "grid",
"gridX": 400,
"gridY": 700
}
},
"00:00:00:00:00:30/None": {
"basic": {
"name": "h3",
"locType": "grid",
"gridX": 750,
"gridY": 700
}
},
"00:00:00:00:00:40/None": {
"basic": {
"name": "h4",
"locType": "grid",
"gridX": 850,
"gridY": 700
```

```
    }
    }
    }
    }
```

Script 87 Netcfg.json[80] (ONOS) para PoC NG-SDN
Basado en (Cascone, 2019)

```
/*
* Copyright 2019-present Open Networking Foundation
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package org.onosproject.ngsdn.tutorial.pipeconf;

import com.google.common.collect.ImmutableList;
import com.google.common.collect.ImmutableMap;
import org.onlab.packet.DeserializationException;
import org.onlab.packet.Ethernet;
import org.onlab.util.ImmutableByteSequence;
import org.onosproject.net.ConnectPoint;
import org.onosproject.net.DeviceId;
import org.onosproject.net.Port;
import org.onosproject.net.PortNumber;
import org.onosproject.net.device.DeviceService;
import org.onosproject.net.driver.AbstractHandlerBehaviour;
import org.onosproject.net.flow.TrafficTreatment;
import org.onosproject.net.flow.criteria.Criterion;
import org.onosproject.net.packet.DefaultInboundPacket;
```

[80] Netcfg: https://github.com/opennetworkinglab/ngsdn-tutorial/blob/master/mininet/netcfg.json

```
import org.onosproject.net.packet.InboundPacket;
import org.onosproject.net.packet.OutboundPacket;
import org.onosproject.net.pi.model.PiMatchFieldId;
import org.onosproject.net.pi.model.PiPacketMetadataId;
import org.onosproject.net.pi.model.PiPipelineInterpreter;
import org.onosproject.net.pi.model.PiTableId;
import org.onosproject.net.pi.runtime.PiAction;
import org.onosproject.net.pi.runtime.PiPacketMetadata;
import org.onosproject.net.pi.runtime.PiPacketOperation;

import java.nio.ByteBuffer;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.Optional;

import static java.lang.String.format;
import static java.util.stream.Collectors.toList;
import static org.onlab.util.ImmutableByteSequence.copyFrom;
import static org.onosproject.net.PortNumber.CONTROLLER;
import static org.onosproject.net.PortNumber.FLOOD;
import static org.onosproject.net.flow.instructions.Instruction.Type.OUTPUT;
import static org.onosproject.net.flow.instructions.Instructions.OutputInstruction;
import static org.onosproject.net.pi.model.PiPacketOperationType.PACKET_OUT;
import static org.onosproject.ngsdn.tutorial.AppConstants.CPU_PORT_ID;

/**
* Interpreter implementation.
*/
public class InterpreterImpl extends AbstractHandlerBehaviour
implements PiPipelineInterpreter {

// From v1model.p4
private static final int V1MODEL_PORT_BITWIDTH = 9;

// From P4Info.
private static final Map<Criterion.Type, String> CRITERION_MAP =
new ImmutableMap.Builder<Criterion.Type, String>()
.put(Criterion.Type.IN_PORT, "standard_metadata.ingress_port")
.put(Criterion.Type.ETH_DST, "hdr.ethernet.dst_addr")
.put(Criterion.Type.ETH_SRC, "hdr.ethernet.src_addr")
```

```
.put(Criterion.Type.ETH_TYPE, "hdr.ethernet.ether_type")
.put(Criterion.Type.IPV6_DST, "hdr.ipv6.dst_addr")
.put(Criterion.Type.IP_PROTO, "hdr.ipv6.next_hdr")
.put(Criterion.Type.ICMPV6_TYPE, "hdr.icmpv6.type")
.build();

/**
* Returns a collection of PI packet operations populated with metadata
* specific for this pipeconf and equivalent to the given ONOS
* OutboundPacket instance.
*
* @param packet ONOS OutboundPacket
* @return collection of PI packet operations
* @throws PiInterpreterException if the packet treatments cannot be
* executed by this pipeline
*/
@Override
public Collection<PiPacketOperation> mapOutboundPacket(OutboundPacket packet)
throws PiInterpreterException {
TrafficTreatment treatment = packet.treatment();

// Packet-out in main.p4 supports only setting the output port,
// i.e. we only understand OUTPUT instructions.
List<OutputInstruction> outInstructions = treatment
.allInstructions()
.stream()
.filter(i -> i.type().equals(OUTPUT))
.map(i -> (OutputInstruction) i)
.collect(toList());

if (treatment.allInstructions().size() != outInstructions.size()) {
// There are other instructions that are not of type OUTPUT.
throw new PiInterpreterException("Treatment not supported: " + treatment);
}

ImmutableList.Builder<PiPacketOperation> builder = ImmutableList.builder();
for (OutputInstruction outInst : outInstructions) {
if (outInst.port().isLogical() && !outInst.port().equals(FLOOD)) {
throw new PiInterpreterException(format(
"Packet-out on logical port '%s' not supported",
outInst.port()));
```

```
} else if (outInst.port().equals(FLOOD)) {
// To emulate flooding, we create a packet-out operation for
// each switch port.
final DeviceService deviceService = handler().get(DeviceService.class);
for (Port port : deviceService.getPorts(packet.sendThrough())) {
builder.add(buildPacketOut(packet.data(), port.number().toLong()));
}
} else {
// Create only one packet-out for the given OUTPUT instruction.
builder.add(buildPacketOut(packet.data(), outInst.port().toLong()));
}
}
return builder.build();
}

/**
* Builds a pipeconf-specific packet-out instance with the given payload and
* egress port.
*
* @param pktData packet payload
* @param portNumber egress port
* @return packet-out
* @throws PiInterpreterException if packet-out cannot be built
*/
private PiPacketOperation buildPacketOut(ByteBuffer pktData, long portNumber)
throws PiInterpreterException {

// Make sure port number can fit in v1model port metadata bitwidth.
final ImmutableByteSequence portBytes;
try {
portBytes = copyFrom(portNumber).fit(V1MODEL_PORT_BITWIDTH);
} catch (ImmutableByteSequence.ByteSequenceTrimException e) {
throw new PiInterpreterException(format(
"Port number %d too big, %s", portNumber, e.getMessage()));
}

// Create metadata instance for egress port.
// TODO EXERCISE 3: modify metadata names to match P4 program
// ---- START SOLUTION ----
final String outPortMetadataName = "egress_port";
```

```
// ---- END SOLUTION ----
final PiPacketMetadata outPortMetadata = PiPacketMetadata.builder()
.withId(PiPacketMetadataId.of(outPortMetadataName))
.withValue(portBytes)
.build();

// Build packet out.
return PiPacketOperation.builder()
.withType(PACKET_OUT)
.withData(copyFrom(pktData))
.withMetadata(outPortMetadata)
.build();
}

/**
* Returns an ONS InboundPacket equivalent to the given pipeconf-specific
* packet-in operation.
*
* @param packetIn packet operation
* @param deviceId ID of the device that originated the packet-in
* @return inbound packet
* @throws PiInterpreterException if the packet operation cannot be mapped
* to an inbound packet
*/
@Override
public InboundPacket mapInboundPacket(PiPacketOperation packetIn, DeviceId deviceId)
throws PiInterpreterException {

// Find the ingress_port metadata.
// TODO EXERCISE 3: modify metadata names to match P4 program
// ---- START SOLUTION ----
final String inportMetadataName = "ingress_port";
// ---- END SOLUTION ----
Optional<PiPacketMetadata> inportMetadata = packetIn.metadatas()
.stream()
.filter(meta -> meta.id().id().equals(inportMetadataName))
.findFirst();

if (!inportMetadata.isPresent()) {
throw new PiInterpreterException(format(
"Missing metadata '%s' in packet-in received from '%s': %s",
```

```
inportMetadataName, deviceId, packetIn));
}

// Build ONOS InboundPacket instance with the given ingress port.

// 1. Parse packet-in object into Ethernet packet instance.
final byte[] payloadBytes = packetIn.data().asArray();
final ByteBuffer rawData = ByteBuffer.wrap(payloadBytes);
final Ethernet ethPkt;
try {
ethPkt = Ethernet.deserializer().deserialize(
payloadBytes, 0, packetIn.data().size());
} catch (DeserializationException dex) {
throw new PiInterpreterException(dex.getMessage());
}

// 2. Get ingress port
final ImmutableByteSequence portBytes = inportMetadata.get().value();
final short portNum = portBytes.asReadOnlyBuffer().getShort();
final ConnectPoint receivedFrom = new ConnectPoint(
deviceId, PortNumber.portNumber(portNum));

return new DefaultInboundPacket(receivedFrom, ethPkt, rawData);
}

@Override
public Optional<Integer> mapLogicalPortNumber(PortNumber port) {
if (CONTROLLER.equals(port)) {
return Optional.of(CPU_PORT_ID);
} else {
return Optional.empty();
}
}

@Override
public Optional<PiMatchFieldId> mapCriterionType(Criterion.Type type) {
if (CRITERION_MAP.containsKey(type)) {
return Optional.of(PiMatchFieldId.of(CRITERION_MAP.get(type)));
} else {
return Optional.empty();
}
}
```

```
    @Override
    public PiAction mapTreatment(TrafficTreatment treatment, PiTableId piTableId)
    throws PiInterpreterException {
    throw new PiInterpreterException("Treatment mapping not supported");
    }

    @Override
    public Optional<PiTableId> mapFlowRuleTableId(int flowRuleTableId) {
    return Optional.empty();
    }
                                                    }
```

Script 88 InterpreterImpl.java[81] (ONOS) para PoC NG-SDN
Basado en (Cascone, 2019)

[81] InterpreterImpl.java: https://github.com/opennetworkinglab/ngsdn-tutorial/blob/master/solution/exercise3/InterpreterImpl.java

Anexo G: Proceso de Instalación de SDWAN Viptela en EVE-ng

El PoC de SD-WAN Viptela se desarrolla en EVE-ng, además de una breve explicación de la implementación de un laboratorio en un *Sandbox* de DevNet y *dCloud* para comprender de mejor manera este entorno de redes de nueva generación.

En primer lugar, hay que descargarse las imágenes necesarias de los equipos que serán parte de la red SD-WAN de la página oficial de Cisco. Para ello, se requiere ser *partner*, cliente o empleado de Cisco para una descarga legal de dichas imágenes.

La URL para las descargas es:

https://software.cisco.com/download/home/286320995/type

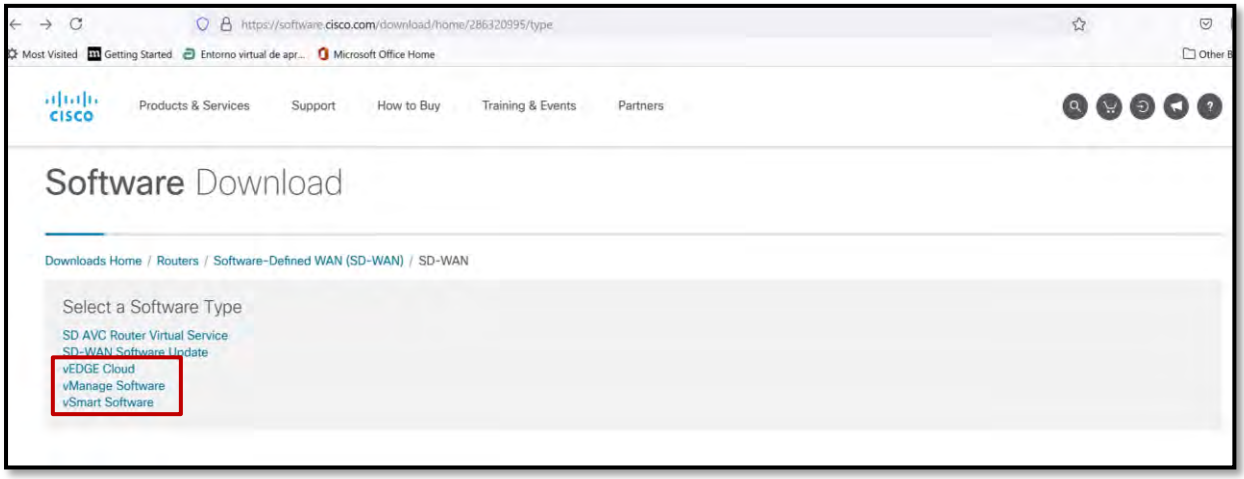


Figura G - 1 Página de descarga de las Imágenes de equipos – PoC SDWAN
Fuente: *Autor*

La versión actual en el momento de la escritura de esta tesis es la **19.2.4(ED)**. Todas las versiones de los equipos a descargar (*vEdge/vBond, vManage* y *vSmart*) deben tener la misma versión para su funcionamiento correcto. La extensión de los archivos debe ser ***.qcow2***

Una vez descargados, abriremos EVE-ng y creamos un nuevo laboratorio.

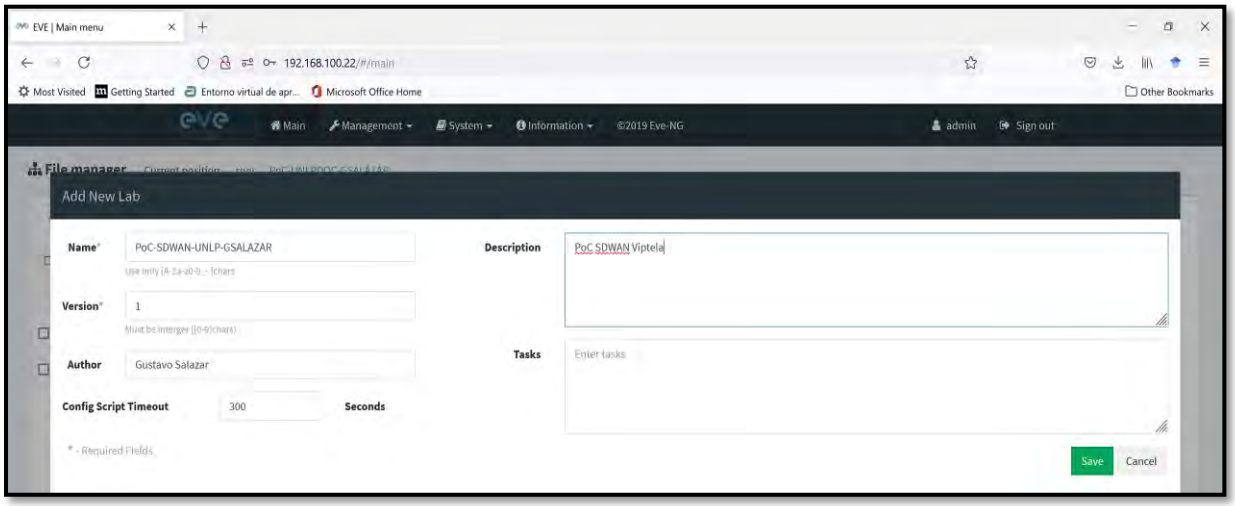


Figura G - 2 Creación de nuevo Lab en EVE-ng – PoC SDWAN
Fuente: *Autor*

Al tratar de incluir los elementos básicos del entorno SDWAN Viptela, se puede verificar que aún no están activos, por lo que debemos incluirlos.

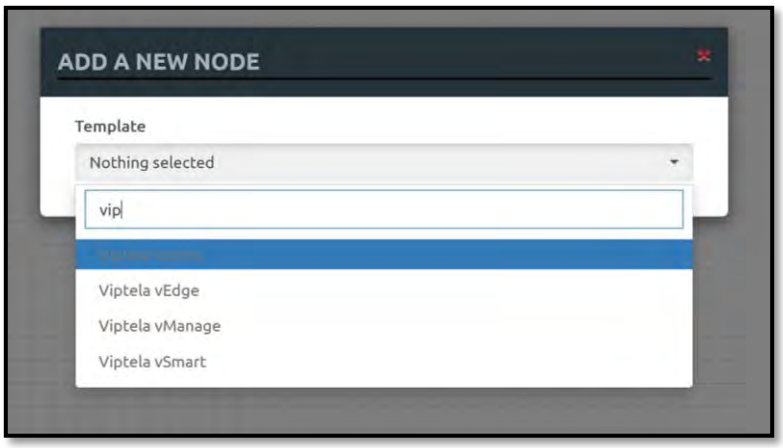


Figura G - 3 Equipos aún no habilitados – PoC SDWAN
Fuente: *Autor*

No hay que olvidar que para cargar las imágenes en EVE-ng, se deben ubicar en una carpeta con un nombre en específico. Como referencia es posible visualizar el *QEMU Image Naming* de EVE-ng:

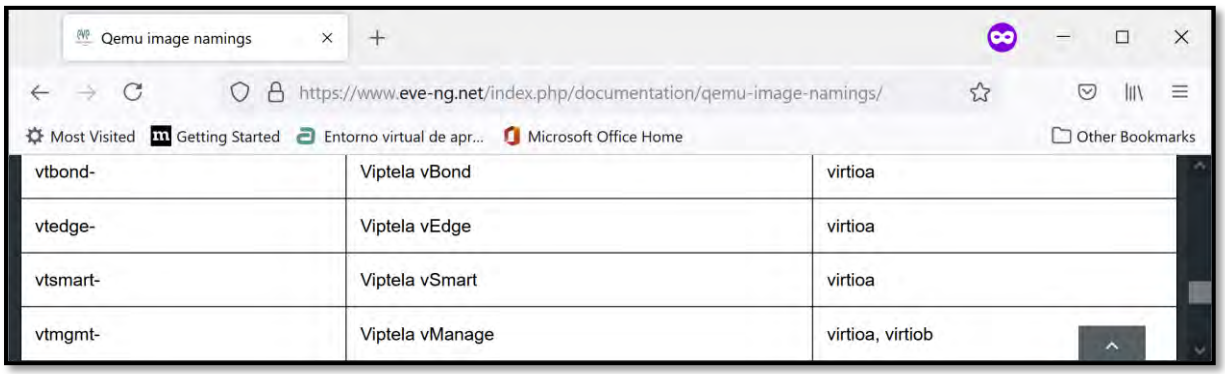


Figura G - 4 Nombres específicos para emulación EVE-ng – PoC SDWAN
Fuente: *Autor*

La primera columna de la *Fig. G -4* indica el nombre de la carpeta y la tercera columna el nombre del archivo en sí.

Como se ha realizado en anteriores PoCs, se debe utilizar un *software* que permita la transferencia de archivos desde la PC *host* hacia la VM de EVE-ng, por ejemplo, WinSCP.

Realizamos la conexión mediante WinSCP a la VM y creamos las carpetas respectivas con su nombre indicado en la *Fig. G-4*. El *path* para la creación de los *folders* es **/opt/unetlab/addons/qemu/.**

Cabe recordar que la misma imagen de *vEdge* servirá para *vBond*. Una vez las imágenes están en las carpetas respectivas, todas deben llamarse **virtioa.qcow2**

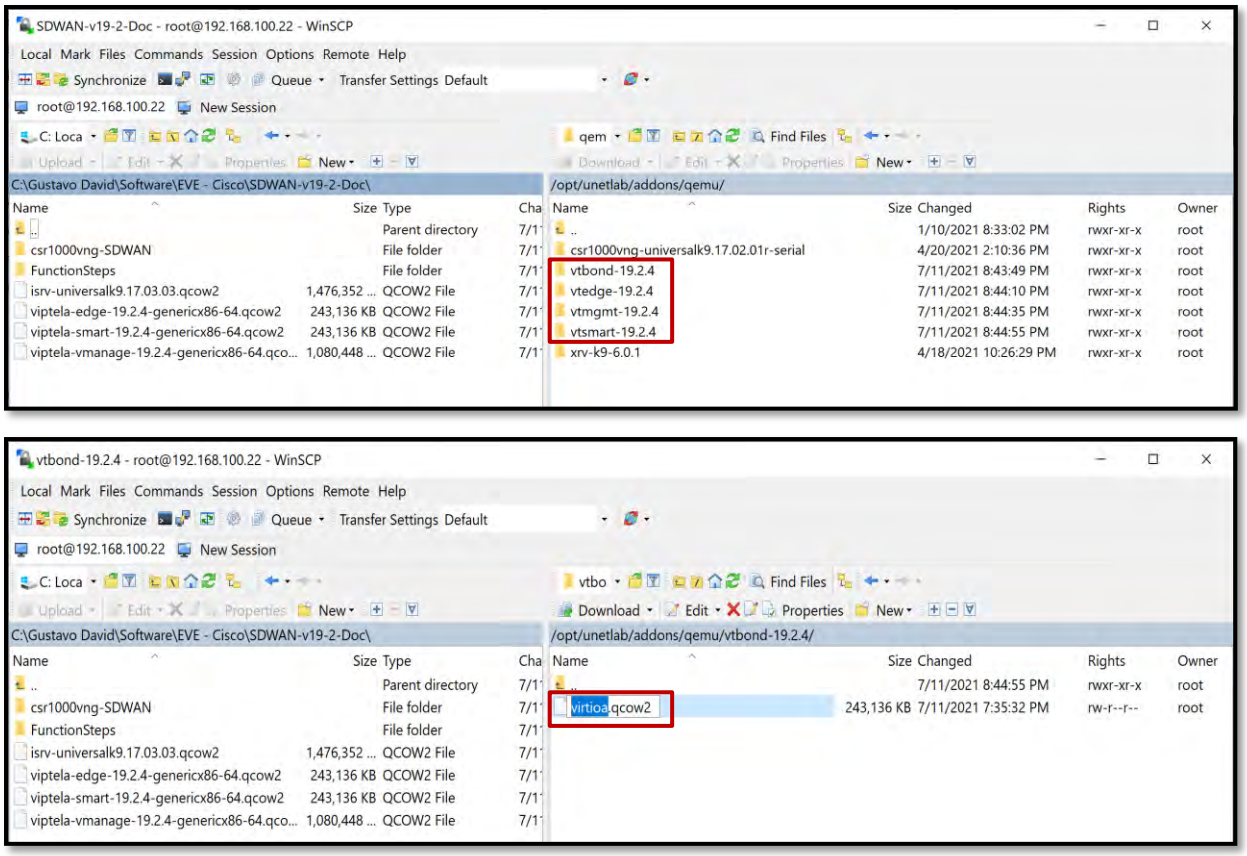


Figura G - 5 Carga de Imágenes a EVE-ng – PoC SDWAN
Fuente: *Autor*

Para el funcionamiento del *vManage* se requiere de un disco duro de 100GB creado en EVE-ng que servirá como Base de Datos para los equipos parte de a infraestructura SD-WAN, para lo cual ingresamos al emulador y creamos dicho disco de nombre **virtiob.qcow2** en la carpeta **vtmgmt-19.2.4** de la siguiente manera:

```
root@eve-ng: /opt/unetlab/addons/qemu/vtmgmt-19.2.4
login as: root
root@192.168.100.22's password:
Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.20.17-eve-ng-ukms+ x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
Last login: Fri Apr 23 10:36:25 2021
root@eve-ng:~#
root@eve-ng:~#
root@eve-ng:~# cd /opt/unetlab/addons/qemu/vtmgmt-19.2.4/
root@eve-ng:/opt/unetlab/addons/qemu/vtmgmt-19.2.4#
root@eve-ng:/opt/unetlab/addons/qemu/vtmgmt-19.2.4# /opt/qemu/bin/qemu-img create -f qcow2 virtiob.qco
w2 100G
Formatting 'virtiob.qcow2', fmt=qcow2 size=107374182400 encryption=off cluster_size=65536 lazy_refcoun
ts=off refcount_bits=16
root@eve-ng:/opt/unetlab/addons/qemu/vtmgmt-19.2.4# ls
virtioa.qcow2  virtiob.qcow2
root@eve-ng:/opt/unetlab/addons/qemu/vtmgmt-19.2.4#
```

Figura G - 6 Creación de disco para vManage en EVE-ng – PoC SDWAN
Fuente: *Autor*

El paso final para a carga de las imágenes es arreglar los permisos de ejecución de dichas imágenes en EVE-ng:

```
root@eve-ng:~#
root@eve-ng:~# /opt/unetlab/wrappers/unl_wrapper -a fixpermissions

root@eve-ng:~#
```

Figura G - 7 Arreglo de permisos en EVE-ng – PoC SDWAN
Fuente: *Autor*

Realizado esos pasos, es factible incluir los nodos a la topología de EVE-ng, dejando la cantidad de RAM por defecto a cada uno (Se requiere al menos de 16 GB de RAM, donde el *vManage* al menos requiere 12GB de RAM). Se requiere también de una nube de transporte (puede usarse una red tipo ISP con *Segment-Routing/*MPLS que permita conectividad entre Plano de Control y Datos) y un acceso para la interfaz vía Web desde la PC *host* (*cloud*).

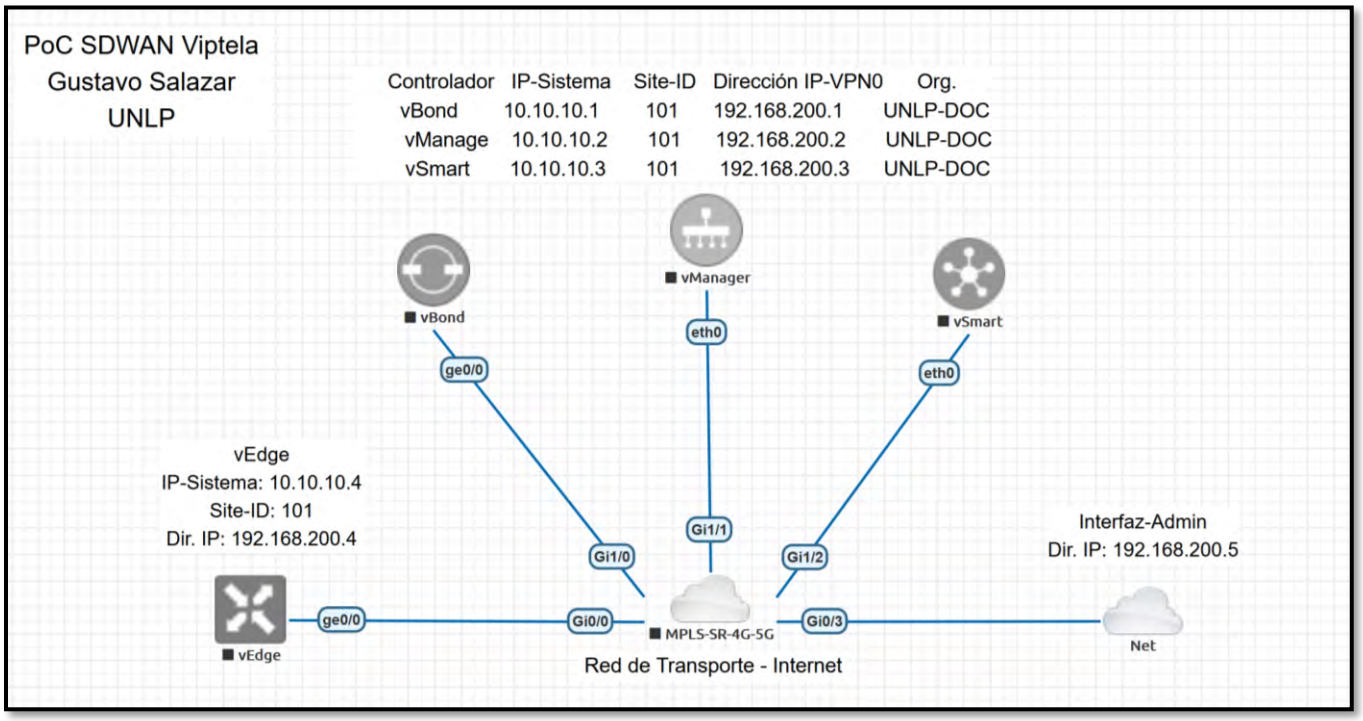


Figura G - 8 Equipos SDWAN en EVE-ng – PoC SDWAN
Fuente: *Autor*

Una vez se tiene la topología base, se encenderán los equipos uno a uno, comenzando por el *vManage*. Los demás pasos se indican en ***4.5 PoC de SD-WAN en EVE-ng***

Anexo H: Entrada en Funcionamiento de *Zodiac FX* y *Zodiac WX* – SDN en infraestructura física con *Aruba VAN SDN Controller y RYU-FlowManager*

El primer paso para el funcionamiento adecuado de los equipos de la red física SDN es la conocer sus elementos principales y cómo conectarlo para su administración, para lo cual se debe seguir los pasos mostrados a continuación:

Configurando a *Zodiac FX*

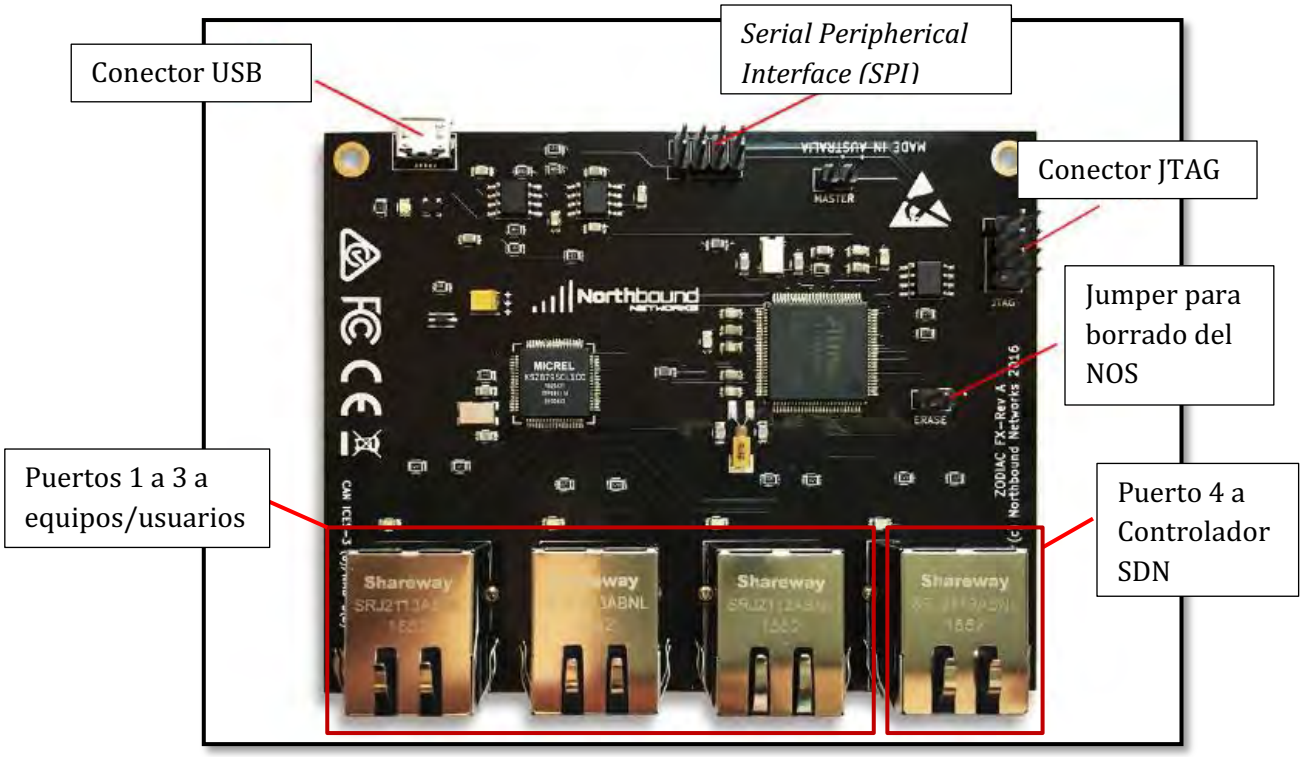


Figura H - 1 Zodiac FX
Basado en (Northbound Networks, 2017)

- Encienda el dispositivo y conéctelo al Puerto 4 por defecto (puerto *Controller-Non OpenFlow Enabled*) hacia la PC donde estará ubicado el controlador SDN.
  Conecte además mediante el puerto USB a la PC/Laptop, recibiendo así la energización.
  En caso de que el dispositivo no sea reconocido por la computadora, se debe descargar los *drivers* los cuales se encuentran en:

  https://forums.northboundnetworks.com/index.php?topic=52.0

  Al ser reconocido, se le asignará un puerto COM para la conexión mediante un emulador de terminal, para el caso de este PoC es el puerto COM4.

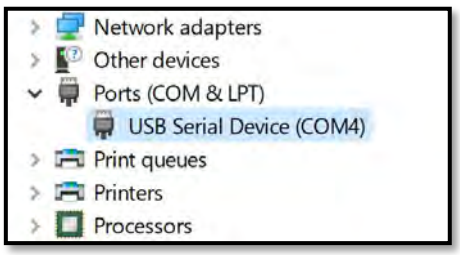


Figura H - 2 Puerto COM4 asignado a Zodiac FX – Administrador de Dispositivos
Fuente: *Autor*

Figura H - 3 Conexión y encendido inicial - Zodiac FX
Fuente: *Autor*

A través de *Putty* (o cualquier emulador de terminal de su elección), ingrese mediante comunicación serial a *Zodiac FX* a su CLI:

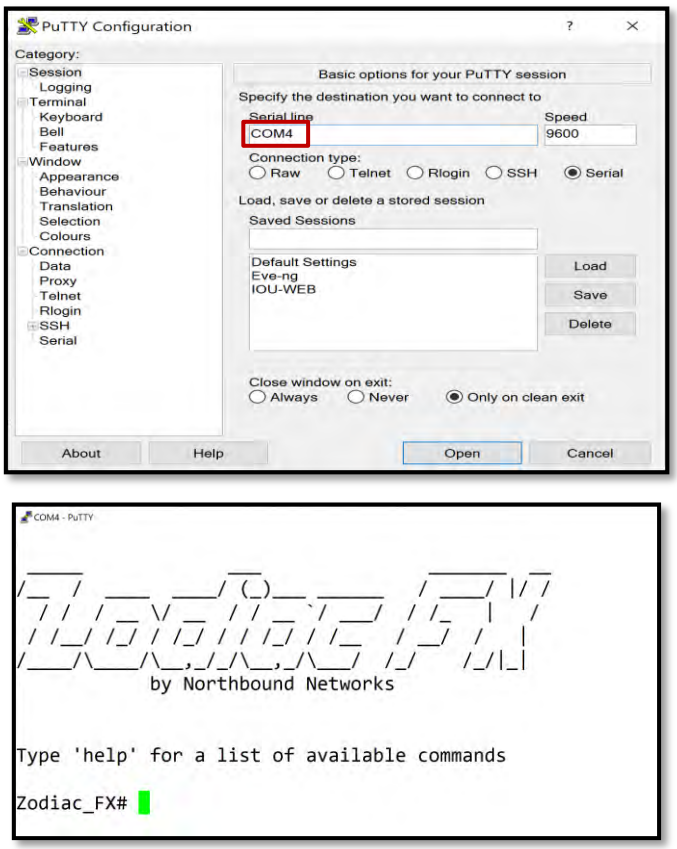


Figura H - 4 Acceso a CLI de Zodiac FX
Fuente: *Autor*

No tiene usuario/contraseña para acceso por defecto. El primer comando a usar es ***help***, el cual no muestra los comandos disponibles agrupados en sus cuatro categorías o modos de configuración: *Base, Config, OpenFlow* y *Debug*.

Para más información sobre dichos modos acceda a (Northbound Networks, 2017).

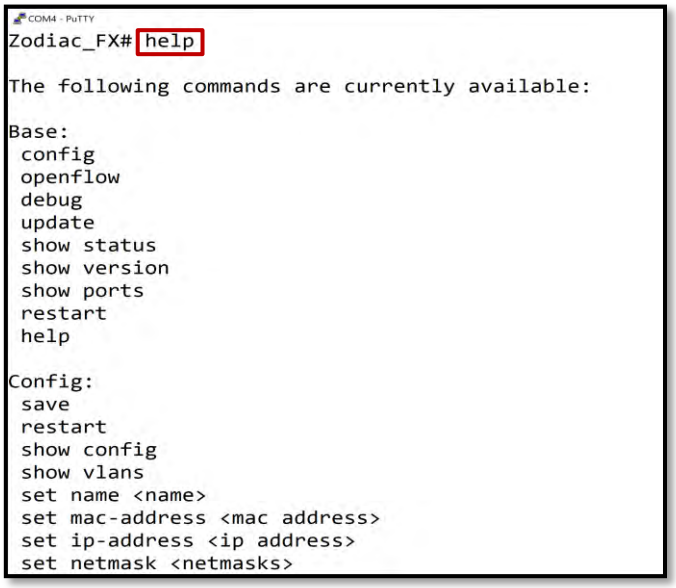
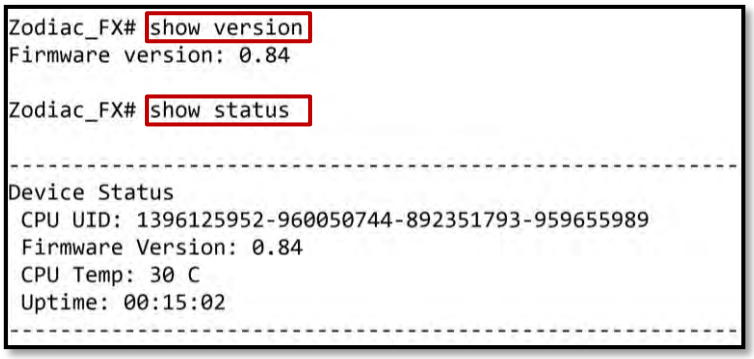


Figura H - 5 Comando help y modos de configuración - Zodiac FX
Fuente: *Autor*

- Unos comandos básicos ***show*** en *Zodiac FX* son:



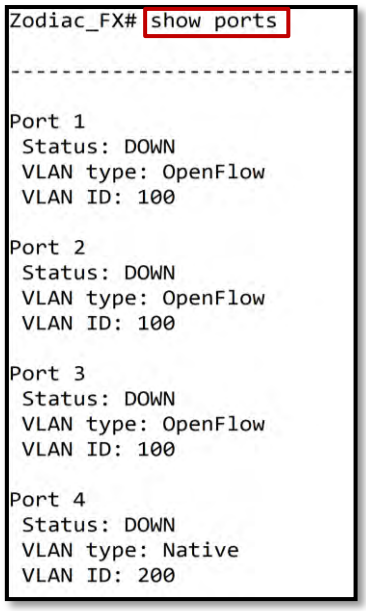


Figura H - 6 Comandos show básicos - Zodiac FX
Fuente: *Autor*

La configuración por defecto de *Zodiac FX* se observa con los siguientes comandos ***show***:

```
Zodiac_FX# config
Zodiac_FX(config)# show config

-----------------------------------------------
Configuration
 Name: Zodiac_FX
 MAC Address: 70:B3:D5:6C:DF:0D
 IP Address: 10.0.1.99
 Netmask: 255.255.255.0
 Gateway: 10.0.1.1
 OpenFlow Controller: 10.0.1.8
 OpenFlow Port: 6633
 Openflow Status: Enabled
 Failstate: Secure
 Force OpenFlow version: Disabled
 EtherType Filtering: Disabled
```

```
Zodiac_FX(config)# show vlans

        VLAN ID         Name                    Type            Tag
-------------------------------------------------------------------------------
        100             'OpenFlow'              OpenFlow        Untagged
        200             'Controller'            Native          Untagged

-------------------------------------------------------------------------------
```

Figura H - 7 Configuración por defecto - Zodiac FX
Fuente: *Autor*

La *Fig. H-7* indica que el equipo viene con la Dir. IP **10.0.1.99/24,** *Gateway* por defecto **10.0.1.1**, Dir. IP del controlador *OpenFlow* **10.0.1.8** y puerto *OpenFlow* **6633**.
Además, se aprecia que tiene dos VLANs por defecto, utilizadas para segmentar tráfico: hacia el Controlador SDN (VLAN 200 – asignada al puerto 4) y hacia los equipos/usuarios finales (VLAN 100 – asignada a los puertos 1-3).

- Con el fin de adecuar la configuración de los equipos *Zodiac FX* al PoC del Capítulo 5, se requiere de cambiar su direccionamiento IP, *Gateway* y dirección de Controlador *OpenSDN*, por ejemplo, para uno de los *switches* sería:

```
Zodiac_FX(config)# set ip-address 192.168.85.10
IP Address set to 192.168.85.10
Zodiac_FX(config)# set gateway 192.168.85.1
Gateway set to 192.168.85.1
Zodiac_FX(config)# set of-controller 192.168.85.100
OpenFlow Server address set to 192.168.85.100
Zodiac_FX(config)# save
Writing Configuration to EEPROM (197 bytes)
Zodiac_FX(config)#
```

```
Zodiac_FX(config)# show config

-----------------------------------------------
Configuration
 Name: Zodiac_FX
 MAC Address: 70:B3:D5:6C:DF:0D
 IP Address: 192.168.85.10
 Netmask: 255.255.255.0
 Gateway: 192.168.85.1
 OpenFlow Controller: 192.168.85.100
 OpenFlow Port: 6633
 Openflow Status: Enabled
 Failstate: Secure
 Force OpenFlow version: Disabled
 EtherType Filtering: Disabled
```

Figura H - 8 Cambio de Dir. IP, Gateway y Dir. De Controlador OpenSDN - Zodiac FX
Fuente: *Autor*

Conectando *Zodiac FX* a un controlador *OpenSDN*

Aruba VAN SDN Controller

- En el PoC con equipos físicos se usará a *Aruba VAN (Virtual Application Networks) Controller* como Controlador *OpenSDN* virtualizado en una VM.

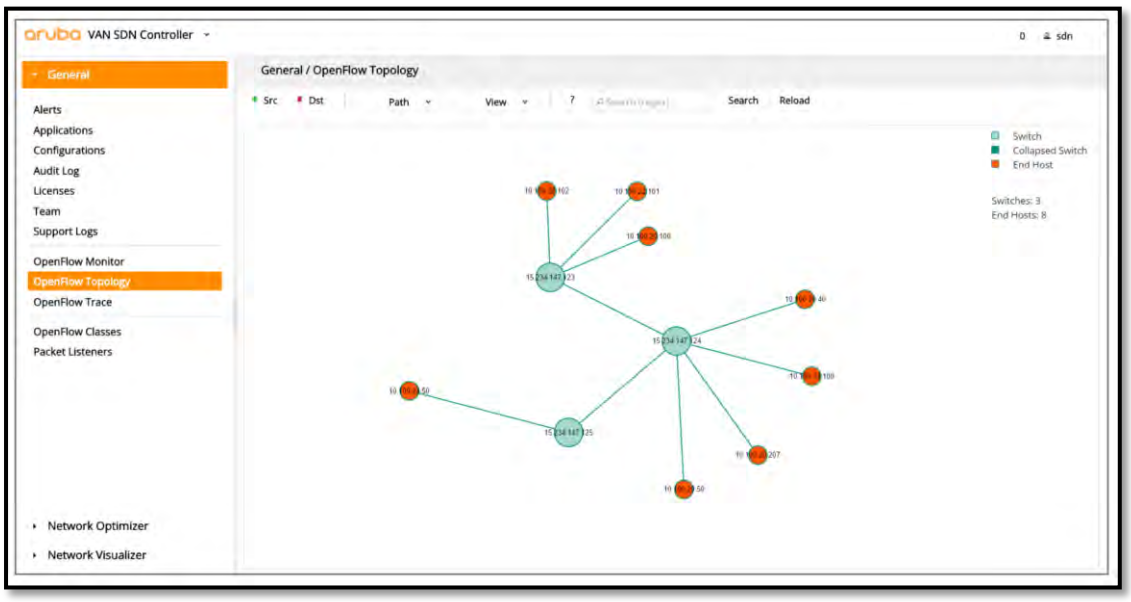


Figura H - 9 GUI – Araba VAN SDN Controller
Fuente: *Autor*

Las características principales de este controlador SDN son:

Tabla H-2 Características de *Aruba VAN SDN Controller*

Características	Descripción
Plataforma	Empresarial enfocada a la innovación
Openflow	Soporta OF 1.0 y OF 1.3
Nodos	<50 dispositivos para *OpenSDN.* Con licencia <4000
APIs	Abiertas (RESTful) y parte del ecosistema *Aruba Open SDN*
Implementación	En VM o *Bare-Metal*

Bases de Datos	Abiertas como *Hazelcast, Cassandra* y *PostgreSQL*
Escalabilidad	Provee un entorno 2n+1 (tres controladores)
Seguridad	HTTPS (REST APIs), TLS (hacia el *Southbound*) y llaves de autenticación
Servicios de Monitoreo	*Topology Service Module* (visibilidad de la infraestructura)
NOS	Basado en Ubuntu 14.04 LTS-64bits
Flujos OF	Hasta 2.3 millones por segundo y 6.5 millones con redundancia (licencia E-LTU – J9863AAE)

Fuente: (Aruba Networks, 2017)

Para más información sobre el controlador:

- **https://support.hpe.com/hpesc/public/docDisplay?docId=a00003658en_us&docLocale=en_US**
- **https://community.arubanetworks.com/blogs/gregory-weaver1/2019/08/14/installing-hpe-van-sdn-controller**
- **https://h20195.www2.hpe.com/v2/GetDocument.aspx?docname=c04111355&doctype=quickspecs&doclang=EN_US&searchquery=&cc=ar&lc=es#**

HPE mantiene el soporte de dicho controlador SDN, sin embargo, desde el año 2017 no ha lanzado una nueva actualización, de todas maneras, es ideal para realizar PoCs que se orienten a comprobar la factibilidad de uso con equipos totalmente reales y comercialmente viables.

Una vez se tiene la imagen del controlador, la cual requiere de 8-16 GB de RAM y un procesador de 4-8 núcleos para su virtualización, cárguelo a un hipervisor (como VMWare *Workstation*) y arránquelo:

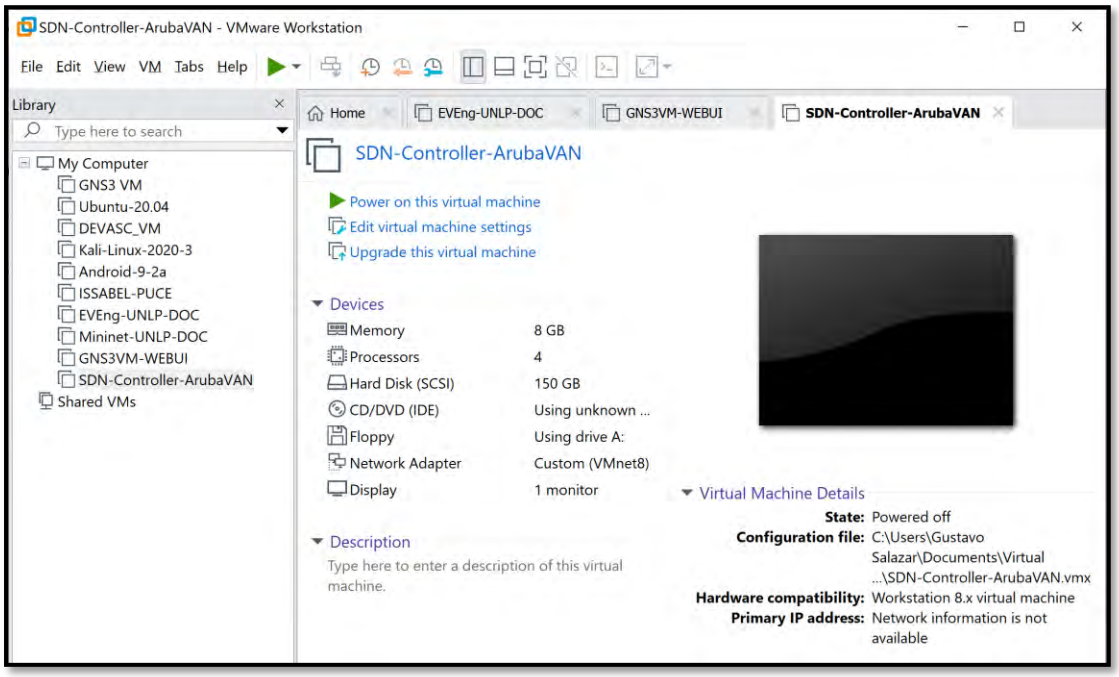


Figura H - 10 Arranque en VM – Araba VAN SDN Controller
Fuente: *Autor*

Configure su dirección IP de forma estática a **192.168.85.1/24**:

```
  GNU nano 2.2.6               File: interfaces                      Modified

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback


# The sw-mgmt network interface
# auto eth1
# iface eth1 inet dhcp
#Configured from installation
#The mgmt network interfaces
auto eth0
#iface eth0 inet dhcp
iface eth0 inet static
        address 192.168.85.100
        netmask 255.255.255.0
        gateway 192.168.85.1_

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

```
PING 192.168.85.1 (192.168.85.1) 56(84) bytes of data.
^C
--- 192.168.85.1 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5000ms

sdn@medium-hLinux:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:06:93:03
          inet addr:192.168.85.100  Bcast:192.168.85.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe06:9303/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:183 errors:0 dropped:0 overruns:0 frame:0
          TX packets:44 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17291 (16.8 KiB)  TX bytes:4500 (4.3 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:1039 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1039 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:176787 (172.6 KiB)  TX bytes:176787 (172.6 KiB)

sdn@medium-hLinux:~$
```

Figura H - 11 Configuración IP estática – Araba VAN SDN Controller
Fuente: *Autor*

Abra un navegador Web e ingrese a la dir. IP del Controlador SDN usando https y el puerto 8443.
El usuario y contraseña de ingreso para *Aruba VAN SDN Controller* es **sdn/skyline**

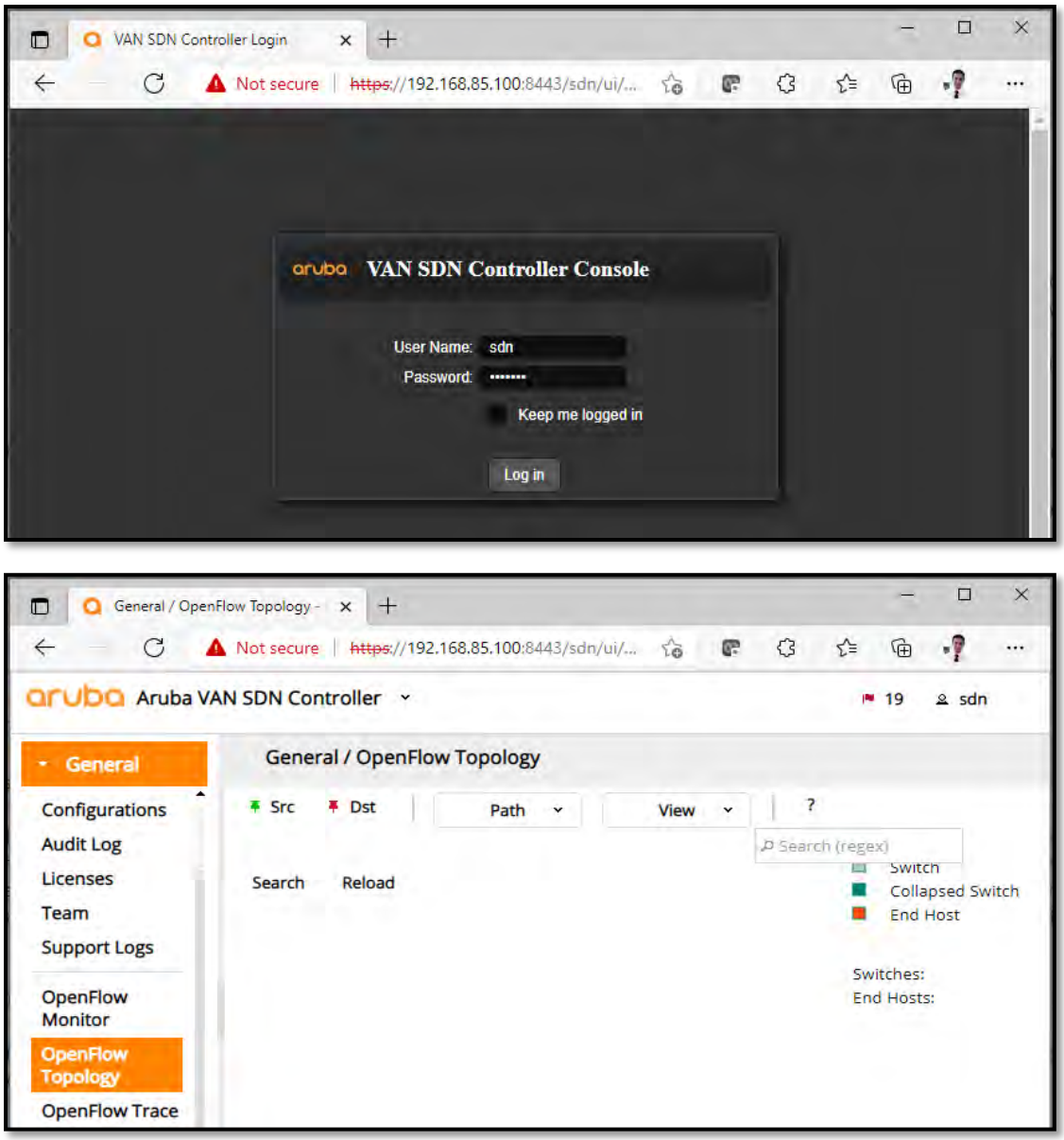


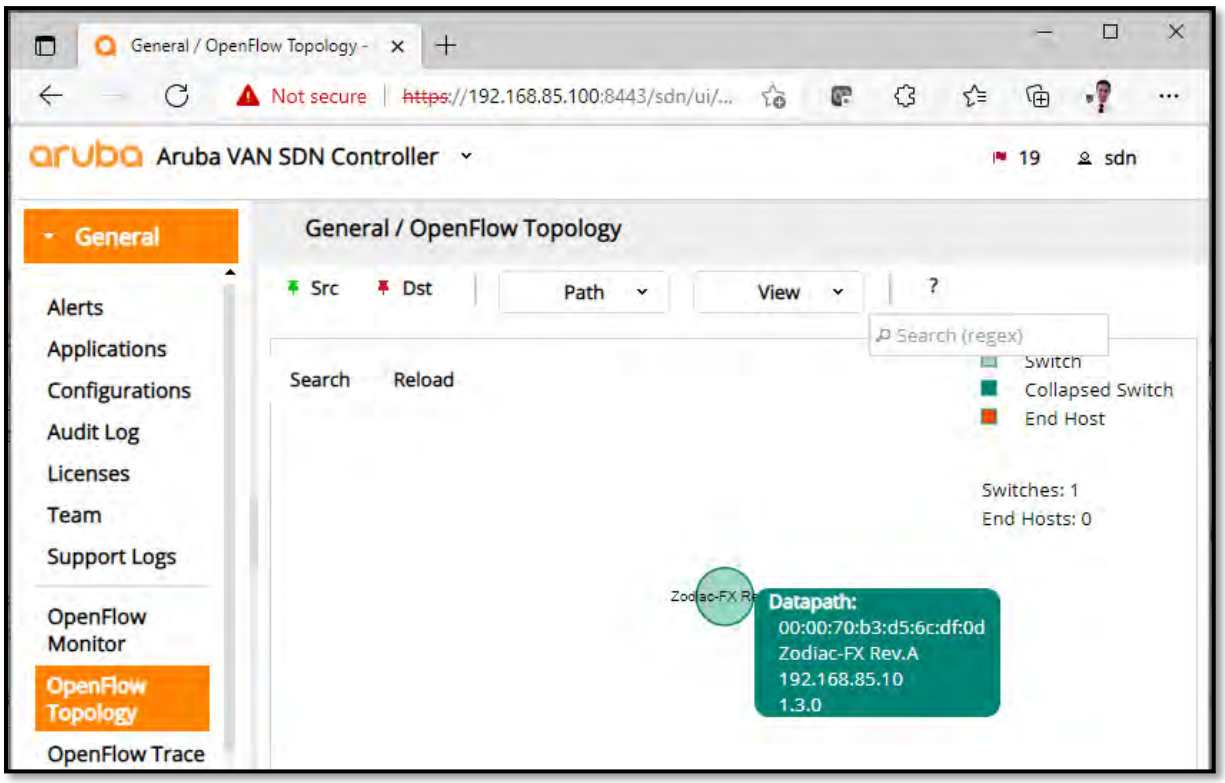
Figura H - 12 Ingreso a interfaz GUI – Araba VAN SDN Controller
Fuente: *Autor*

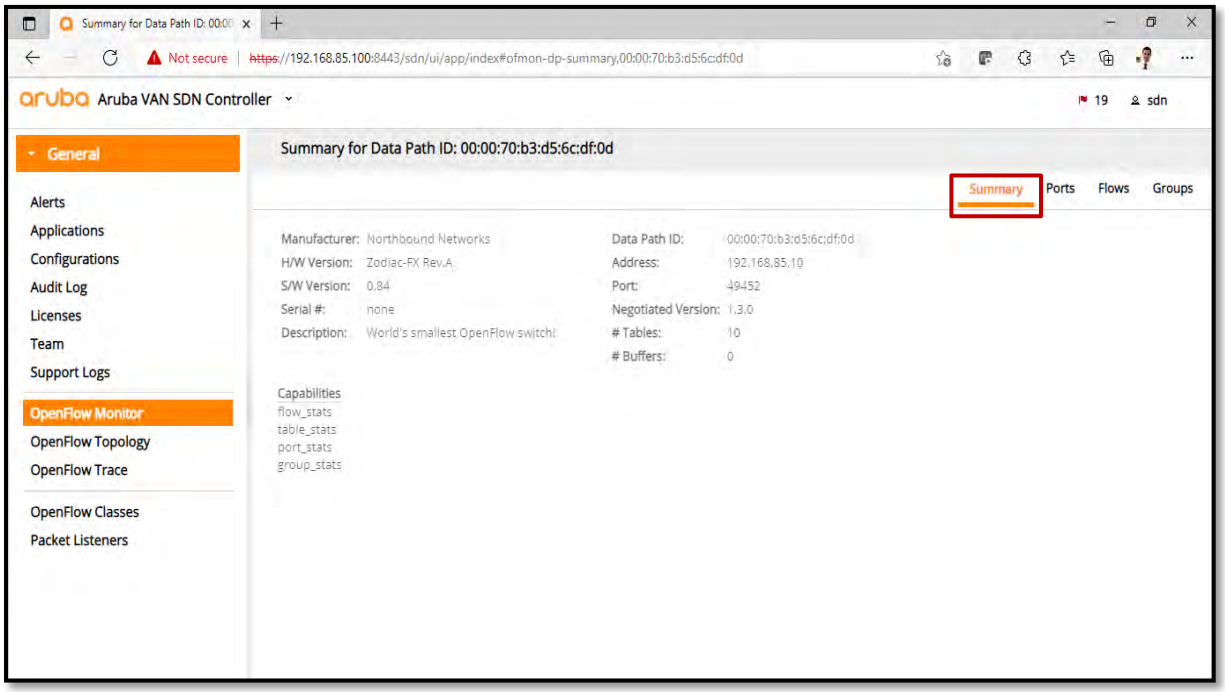
Es momento de conectar *Zodiac FX* al Controlador SDN mediante su puerto 4.
Una vez conectado, se prueba la conectividad de la máquina huésped de la VM del controlador hacia el *Switch OpenFlow*.

```
C:\Users\Gustavo Salazar>ping 192.168.85.10

Pinging 192.168.85.10 with 32 bytes of data:
Reply from 192.168.85.10: bytes=32 time<1ms TTL=255
Reply from 192.168.85.10: bytes=32 time=1ms TTL=255
Reply from 192.168.85.10: bytes=32 time=1ms TTL=255
Reply from 192.168.85.10: bytes=32 time=16ms TTL=255

Ping statistics for 192.168.85.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 16ms, Average = 4ms
```

Figura H - 13 Prueba conectividad máquina huésped a Switch OpenFlow
Fuente: *Autor*

Al ingresar nuevamente a *Aruba VAN* y recargar la topología es factible observar al *Switch Zodiac FX*.





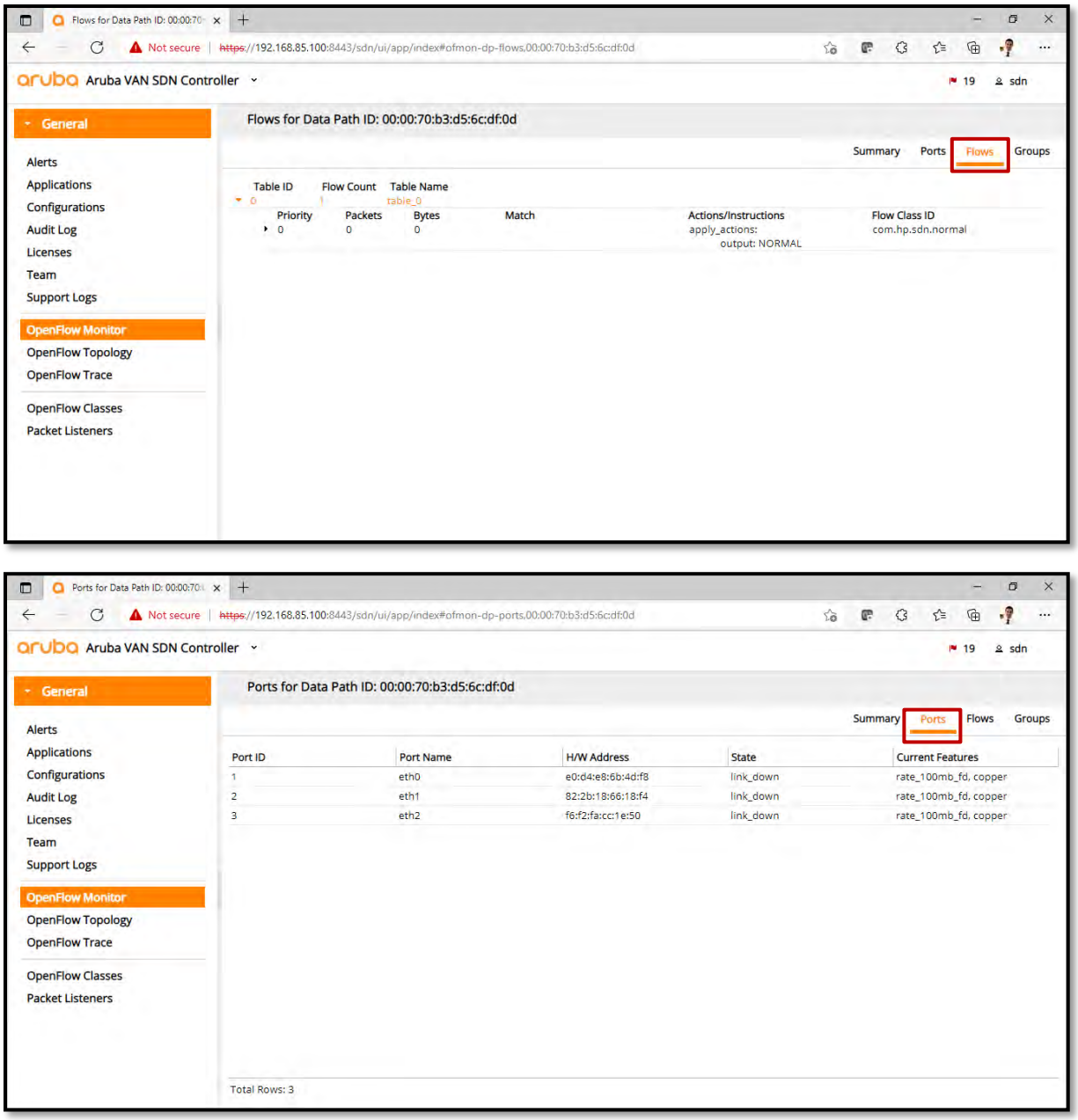


Figura H - 14 Reconocimiento de Zodiac FX – Araba VAN SDN Controller
Fuente: *Autor*

La última imagen de *Fig. H-14* muestra la existencia de los tres puertos de usuario aún no conectados.

En la siguiente figura se puede corroborar la conexión por *OpenFlow* exitosa:

```
Zodiac_FX(openflow)# show status

---------------------------------------------
 Status: Connected
 Version: 1.3 (0x04)
 No tables: 1
 No flows: 1
 Total Lookups: 0
 Total Matches: 0
```

```
Zodiac_FX(openflow)# show flows

---------------------------------------------------------------------------

Flow 1
 Match:
 Attributes:
  Table ID: 0                          Cookie:0xffff000000000000
  Priority: 0                          Duration: 844 secs
  Hard Timeout: 0 secs                 Idle Timeout: 0 secs
  Byte Count: 0                  Packet Count: 0
  Last Match: 00:14:04
 Instructions:
  Apply Actions:
```

Figura H - 15 Conexión exitosa entre Zodiac FX y Araba VAN SDN Controller por OpenFlow
Fuente: *Autor*

Configurando a *Zodiac WX*

Zodiac WX por defecto tiene deshabilitado *OpenFlow*, por lo que para su funcionamiento como AP no necesita de un Controlador SDN.
Zodiac WX tiene configurado por defecto dos redes inalámbricas, una para redes a 2.4GHz (SSID: **ZodiacWX_24GHz**) y otra a 5GHz (SSID: **ZodiacWX_5GHz**), ambas con contraseña **66666666**.

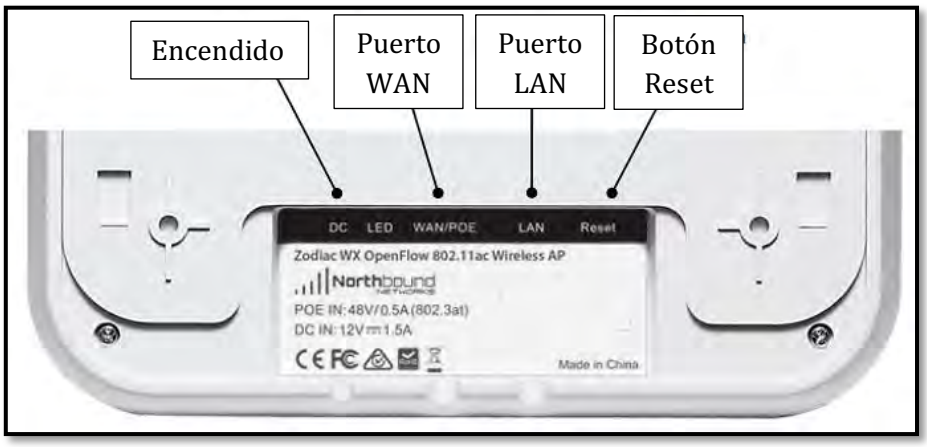
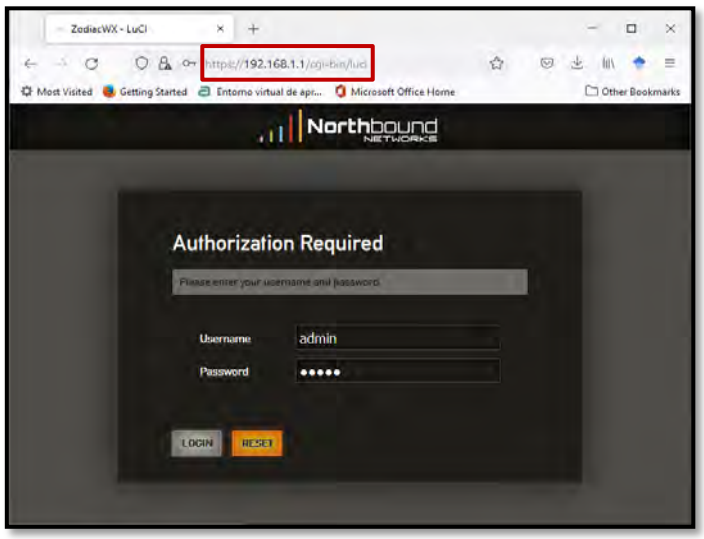


Figura H - 16 Puerto en Zodiac WX
Recuperado de (Northbound Networks, 2017)

Encienda el equipo y conecte al Controlador vía Puerto LAN. La dir. IP por defecto de *Zodiac WX* es **192.168.1.1/24**. Ingrese a su GUI a través de un navegador Web mediante https (*Zodiac WX* rechaza conexiones no seguras). El usuario y contraseña por defecto son **admin/admin**.

Una de las primeras tareas será cambiar la contraseña por defecto. Para el PoC será **UNLPDOC**.

Figura H - 17 Encendido y conexión inicial de Zodiac WX
Fuente: *Autor*



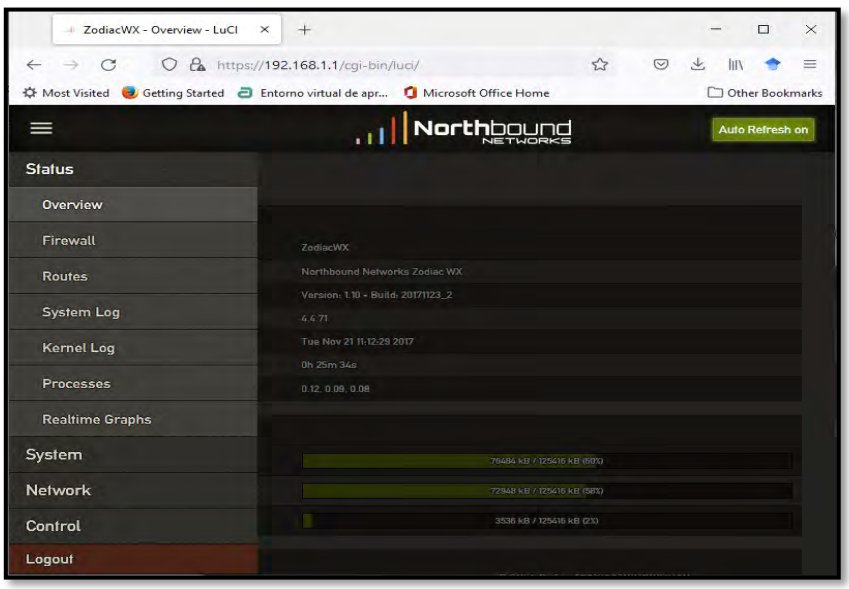


Figura H - 18 Ingreso a GUI de Zodiac WX – LuCI (LUA Configuration Interface)
Fuente: *Autor*

Para cambiar las credenciales de ingreso, es posible hacerlo en *System-Administration*.

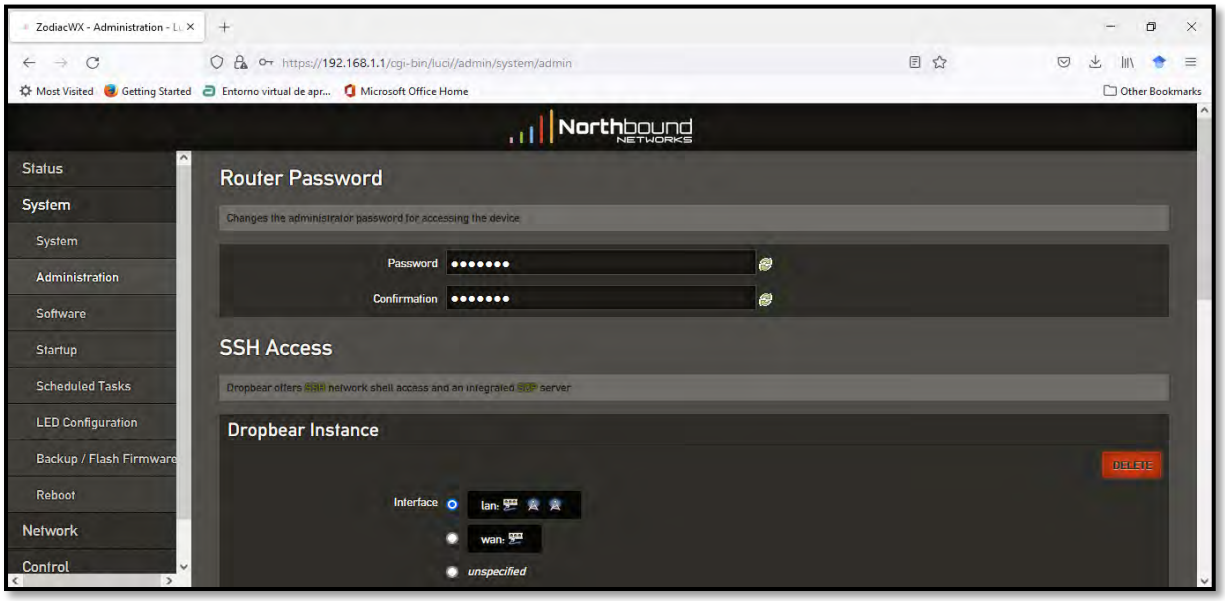


Figura H - 19 Cambio de Contraseñas – Zodiac WX
Fuente: *Autor*

Para configurar parámetros de la red inalámbrica (SSID y Contraseña de acceso WPA2-PSK), se debe dirigir a *Network-Wireless*. En esa primera pantalla se observará los dos SSIDs:

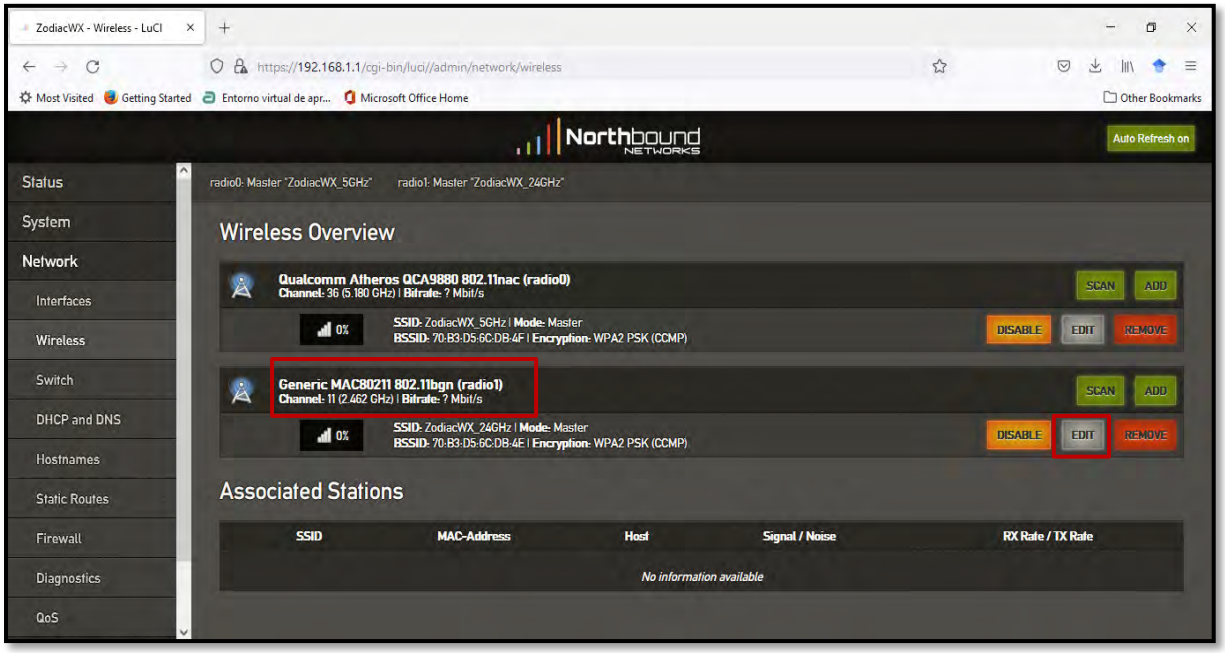


Figura H - 20 Redes Inalámbricas por defecto – Zodiac WX
Fuente: *Autor*

Para el PoC del Capítulo 5 se usará la SSID de 2.4 GHz. Para hacer cambios a dicha red, se ingresará a *Edit* tal como se observa en la *Fig. H-20.*
El SSID es: **PoC-UNLP-GS-24GHz** y su contraseña: **UNLPDOC-GS**

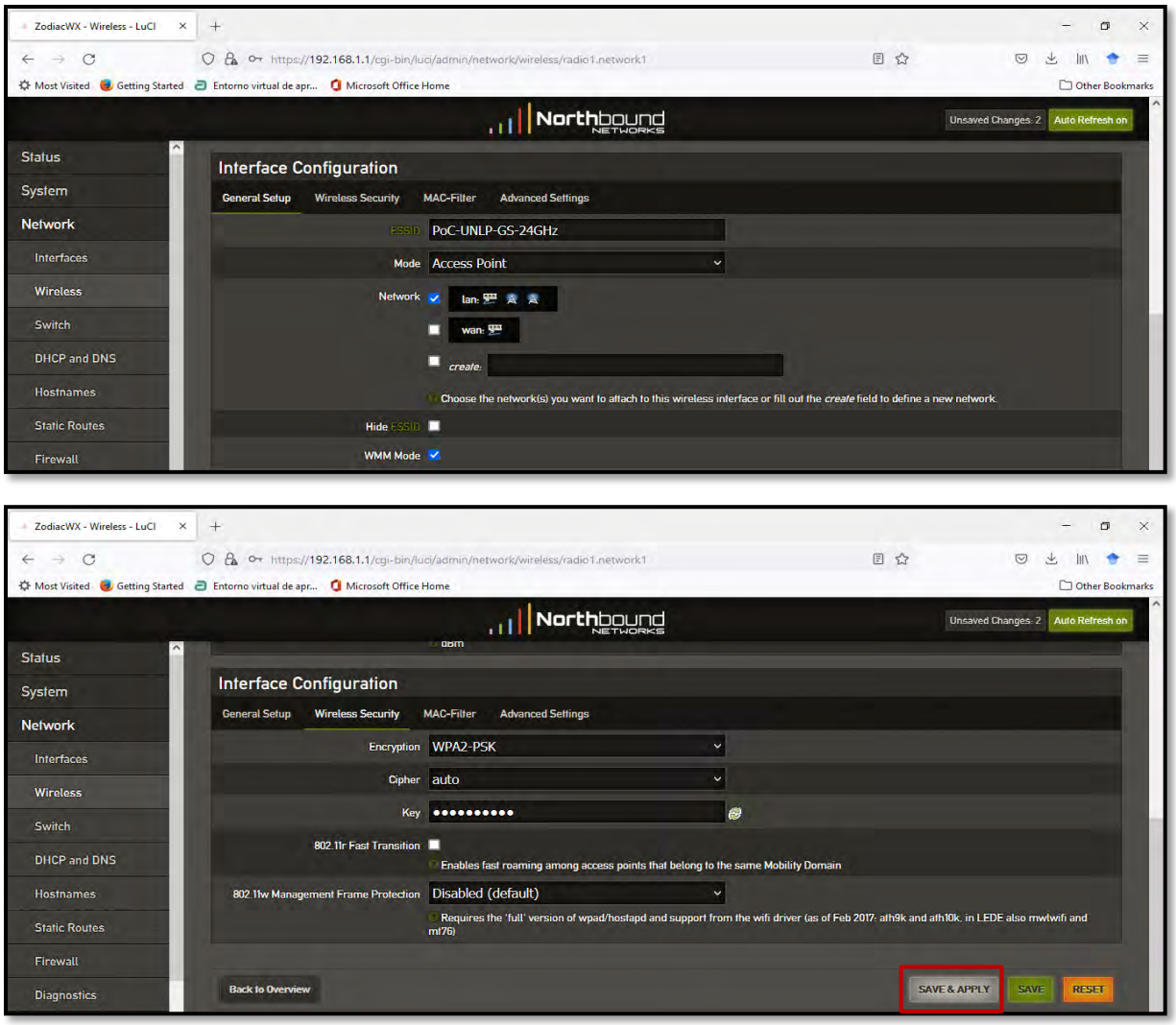
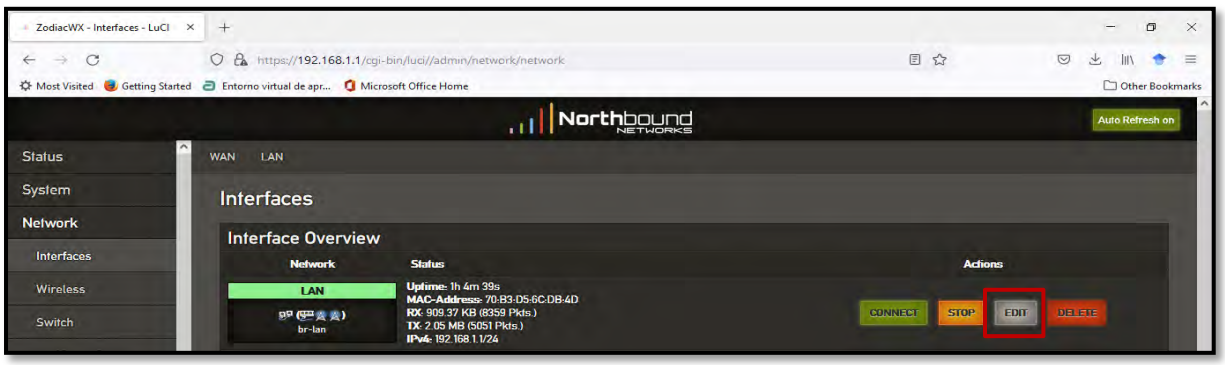


Figura H - 21 Cambios Configuración Red Inalámbrica – Zodiac WX
Fuente: *Autor*

Activación de *OpenFlow* en *Zodiac WX*

Al estar deshabilitado por defecto *OpenFlow*, se requiere siga estos pasos:

- Cambiar la dir. IP del equipo a **192.168.85.20/24**
  Ingrese a *Network – Interfaces – Edit* en la interfaz LAN



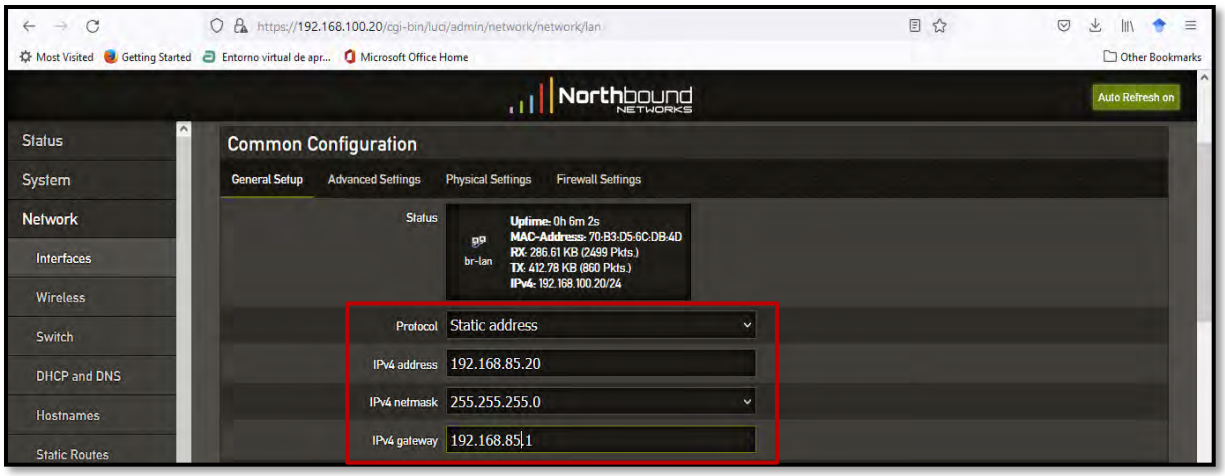


Figura H - 22 Cambio de Dir. IP – Zodiac WX
Fuente: *Autor*

Es importante mencionar que *Zodiac WX* tiene capacidad *dual-stack* (soporta IPv4 e IPv6).

- Vuelva a ingresar a la GUI de *Zodiac WX* cambiando la dir. IP de la tarjeta de red de la PC/Laptop huésped de la VM del controlador a una que permita tener conectividad

```
C:\Users\Gustavo Salazar>ping 192.168.85.20

Pinging 192.168.85.20 with 32 bytes of data:
Reply from 192.168.85.20: bytes=32 time=1ms TTL=64
Reply from 192.168.85.20: bytes=32 time=1ms TTL=64
Reply from 192.168.85.20: bytes=32 time=1ms TTL=64
Reply from 192.168.85.20: bytes=32 time=1ms TTL=64

Ping statistics for 192.168.85.20:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 1ms, Average = 1ms
```

Figura H - 23 Cambio de Dir. IP – Zodiac WX
Fuente: *Autor*

- Habilitar *OpenFlow* y configure la dir. IP del controlador SDN
  Ingrese a *Control – Settings*

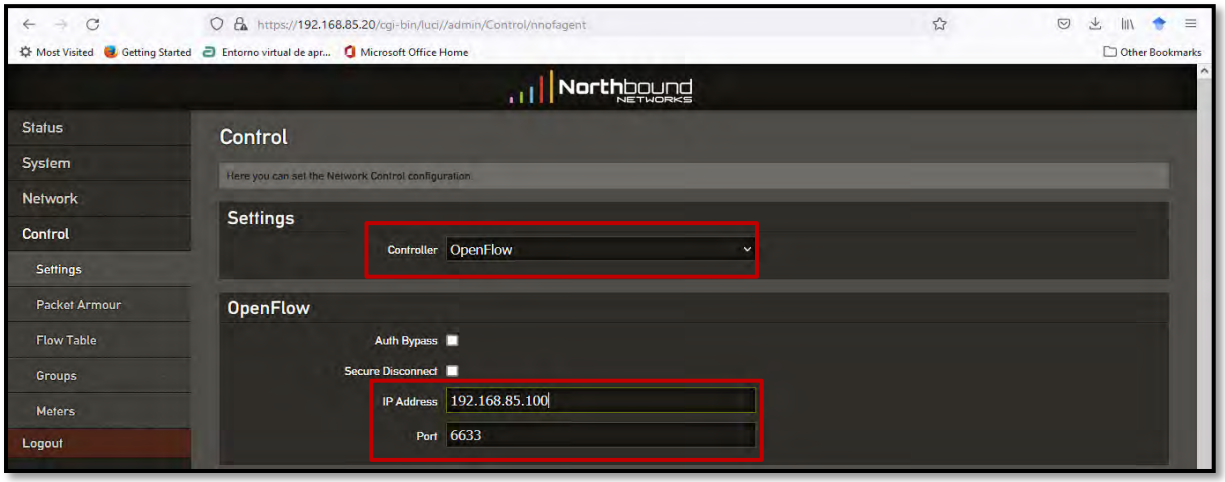
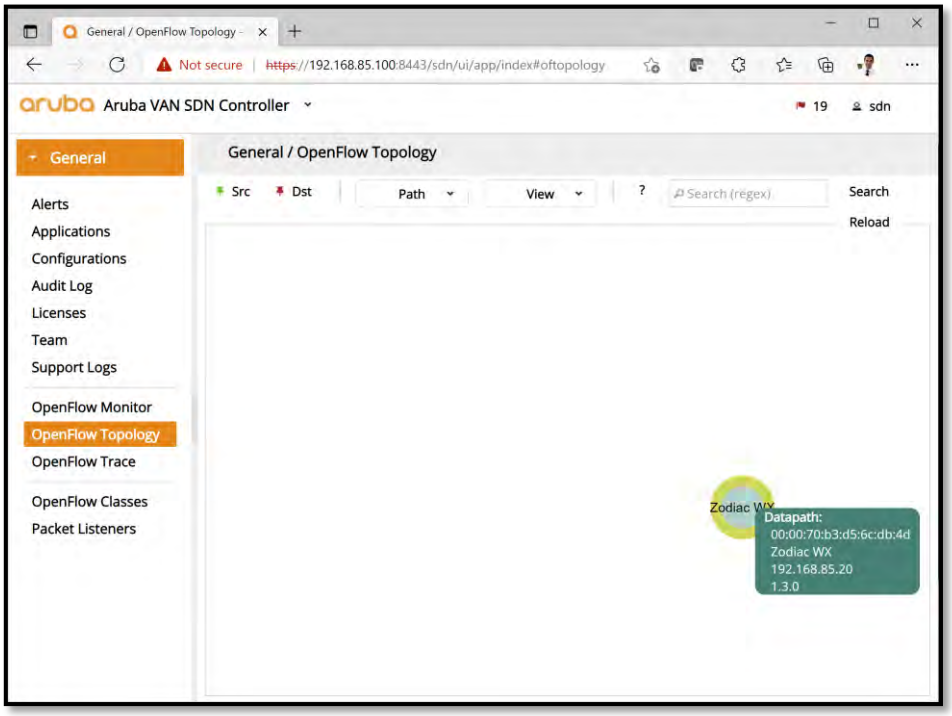
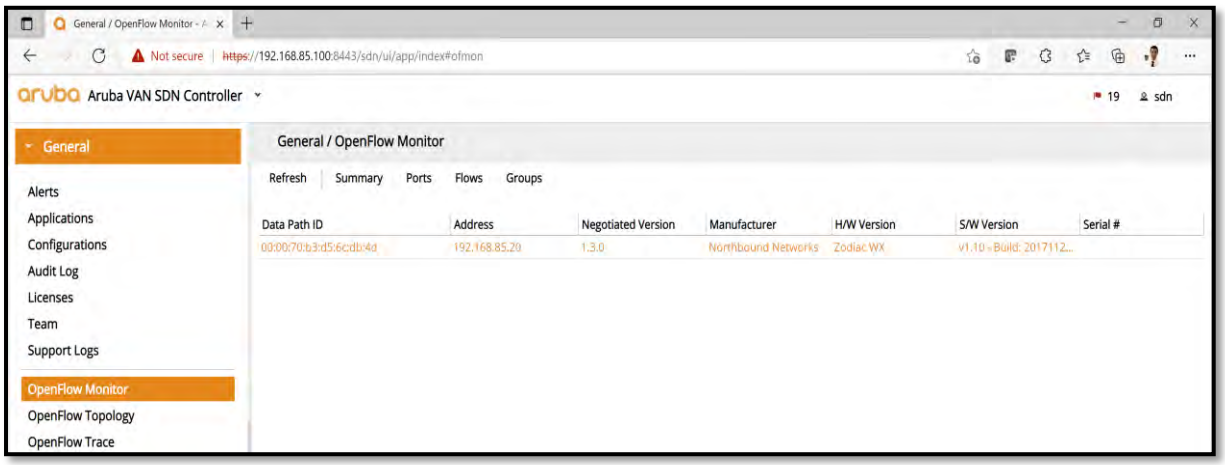
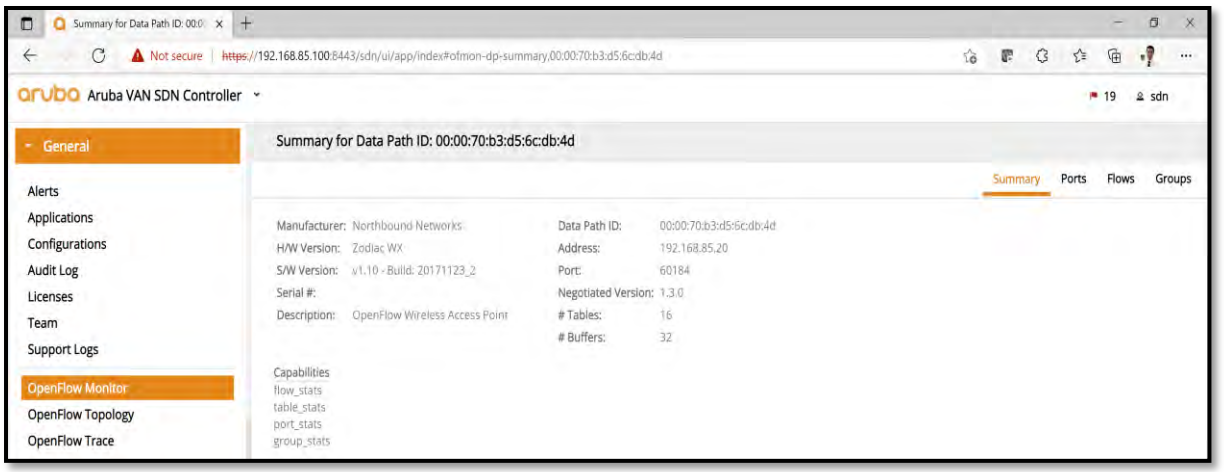


Figura H - 24 Activar OpenFlow – Zodiac WX
Fuente: *Autor*

Si *Aruba VAN SDN Controller* está funcional, el controlador podrá reconocer a *Zodiac WX*







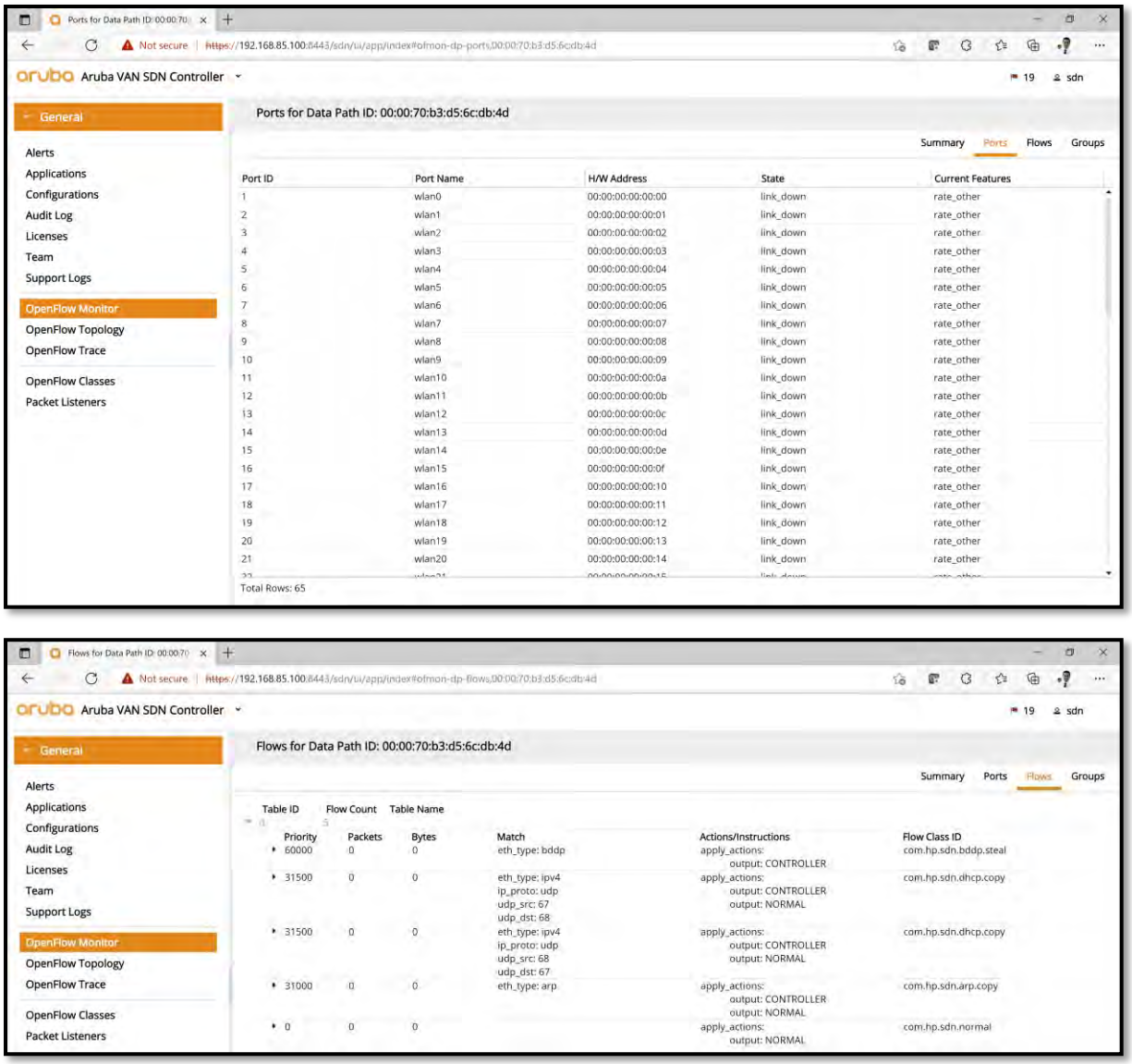


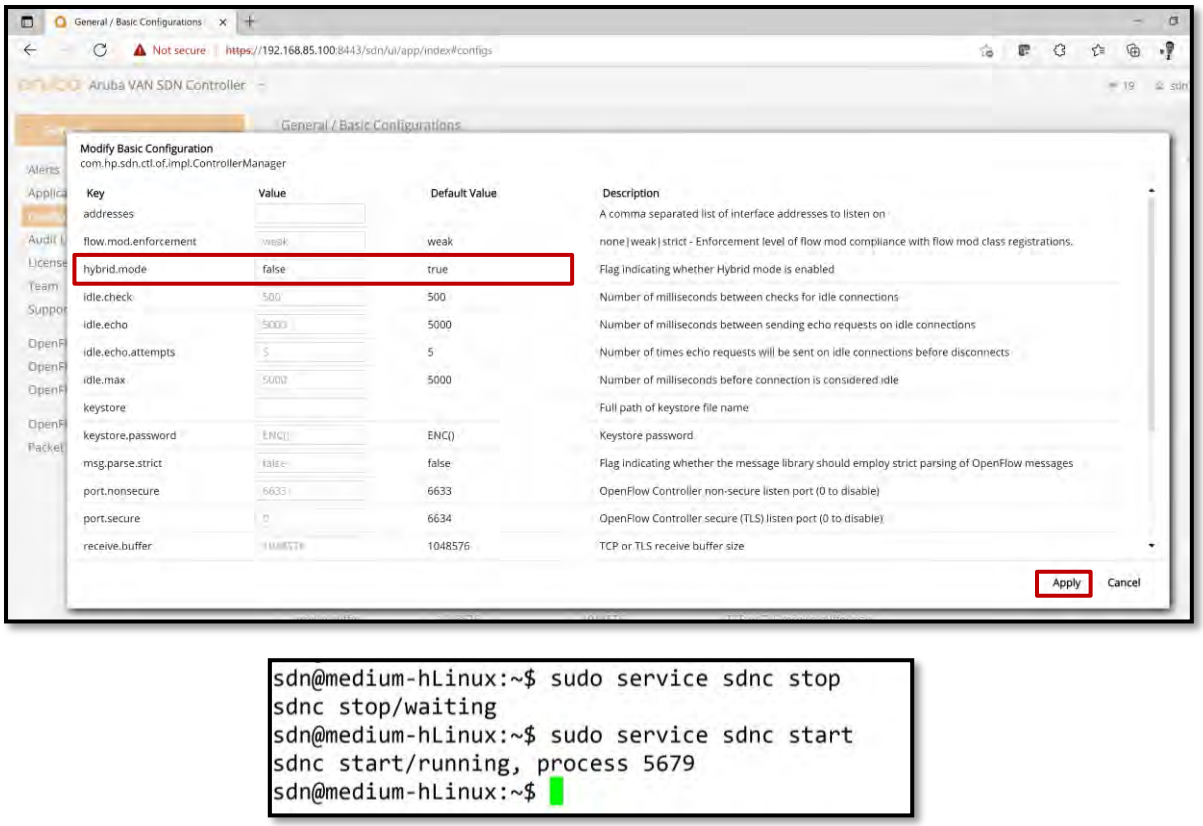
Figura H - 25 Reconocimiento de Zodiac WX por Aruba VAN SDN Controller
Fuente: *Autor*

La penúltima imagen de la *Fig. H-25* indica que es posible conectar hasta 64 *hosts* inalámbricos.

Limitaciones de *Aruba VAN SDN Controller*

Para el adecuado funcionamiento del Controlador *Aruba VAN*, es necesario deshabilitar el mode *Hybrid Mode* para escribir flujos y controlar la comunicación.

Para quitar ese modo, se ingresa al controlador en *Configuration - Controller Manager – Modify – Hybrid Mode – false*




```
sdn@medium-hLinux:~$ sudo service sdnc stop
sdnc stop/waiting
sdn@medium-hLinux:~$ sudo service sdnc start
sdnc start/running, process 5679
sdn@medium-hLinux:~$
```

Figura H - 26 Deshabilitar modo Híbrido – Aruba VAN SDN Controller
Fuente: *Autor*

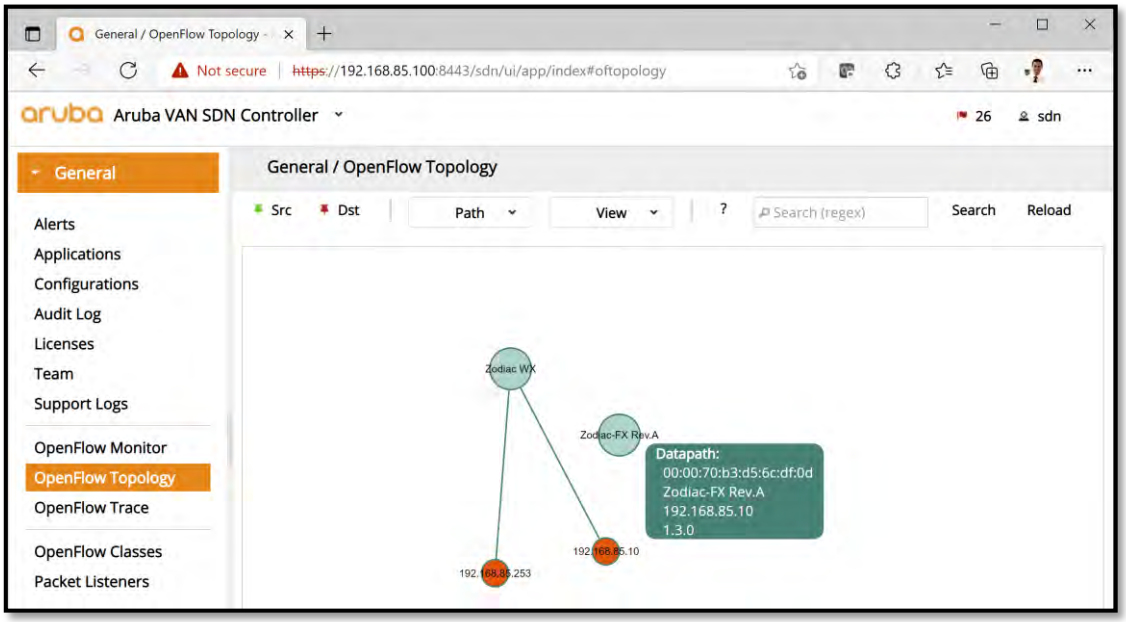


Figura H - 27 Visualización de Zodiac FX, Zodiac WX y hosts – Aruba VAN SDN Controller
Fuente: *Autor*

A través de *Aruba VAN SDN Controller* se aprecia el descubrimiento de los equipos, sin embargo, el manejo de flujos no cuenta aún con una aplicación compatible capaz de permitir la manipulación de la comunicación entre usuarios, por ello, se plantea el uso de RYU y *FlowManager*, controlador que es capaz de convertir el *switch Zodiac FX* en *Firewall, Router* y por supuesto en *Switch*.

RYU SDN Controller y FlowManager

Instalación de RYU.
RYU es NOS abierto licenciado bajo *Apachev2.0* que soporta *OpenFlow* como protocolo de comunicación *Southbound*, el cual escucha el puerto 6633 por defecto.
Para la adecuada instalación del Controlador SDN RYU, se lo hará en un VM con **Ubuntu 20.04.**

- Realizar una actualización de Ubuntu Linux (**sudo apt-get update**)
- Para instalar RYU existen dos maneras, una a través de **pip3** y otra clonando el repositorio (forma manual). En el PoC del Capítulo 5 se usa la forma manual con los siguientes comandos:

Figura H - 28 Comandos para instalación de RYU Controller
Recuperado de (ProgrammerSought, 2021)

En caso de que algún error aparezca al verificar la versión instalada de RYU (con el comando **ryu-manager --version**), es necesario bajar la versión de eventlet:

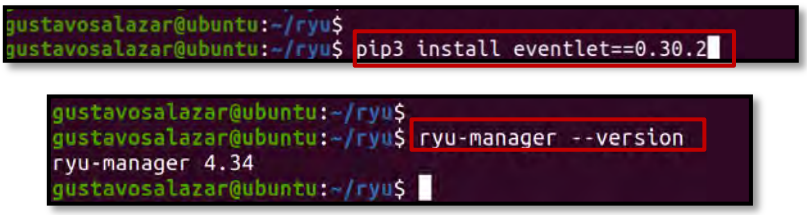


Figura H - 29 Verificación de instalación de RYU
Fuente: *Autor*

Entre las Apps que tiene RYU para el manejo de mensajes en un equipo están:

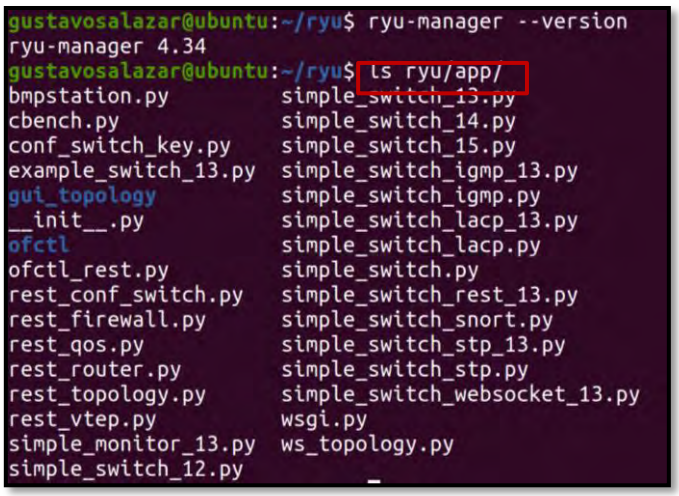


Figura H - 30 Aplicaciones en RYU
Fuente: *Autor*

- Instalación de *FlowManager*

  FlowManager es una aplicación adicional de RYU que permite tener visibilidad y control de flujo de paquetes enviados en la infraestructura *OpenSDN* fácil de instalar y de usar. Para su instalación se debe clonar *FlowManager* de *GitHub*

```
gustavosalazar@ubuntu:~/ryu$ git clone https://github.com/martimy/flowmanager
Cloning into 'flowmanager'...
remote: Enumerating objects: 881, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 881 (delta 3), reused 8 (delta 3), pack-reused 873
Receiving objects: 100% (881/881), 2.31 MiB | 4.14 MiB/s, done.
Resolving deltas: 100% (583/583), done.
```

Figura H - 31 Instalación FlowManager - RYU
Fuente: *Autor*

- Arranque de RYU y *FlowManager*

  Para que RYU corra junto con *FlowManager* es necesario el uso del comando

  sudo ryu-manager --observe-links ~/ryu/flowmanager/flowmanager.py

  Abra un navegador Web e ingrese esta URL para abrir *FlowManager*:

  http://localhost:8080/home/

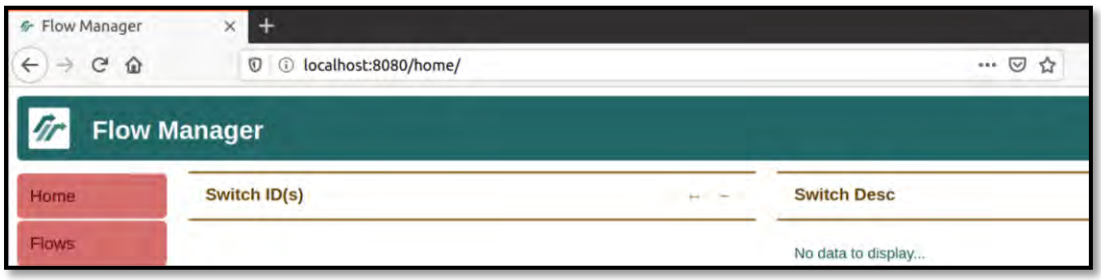


Figura H - 32 GUI de FlowManager - RYU
Fuente: *Autor*

- Control del Plano de Datos por RYU SDN *Controller*

  Verifique exista conectividad entre el Controlador SDN y la infraestructura de prueba:

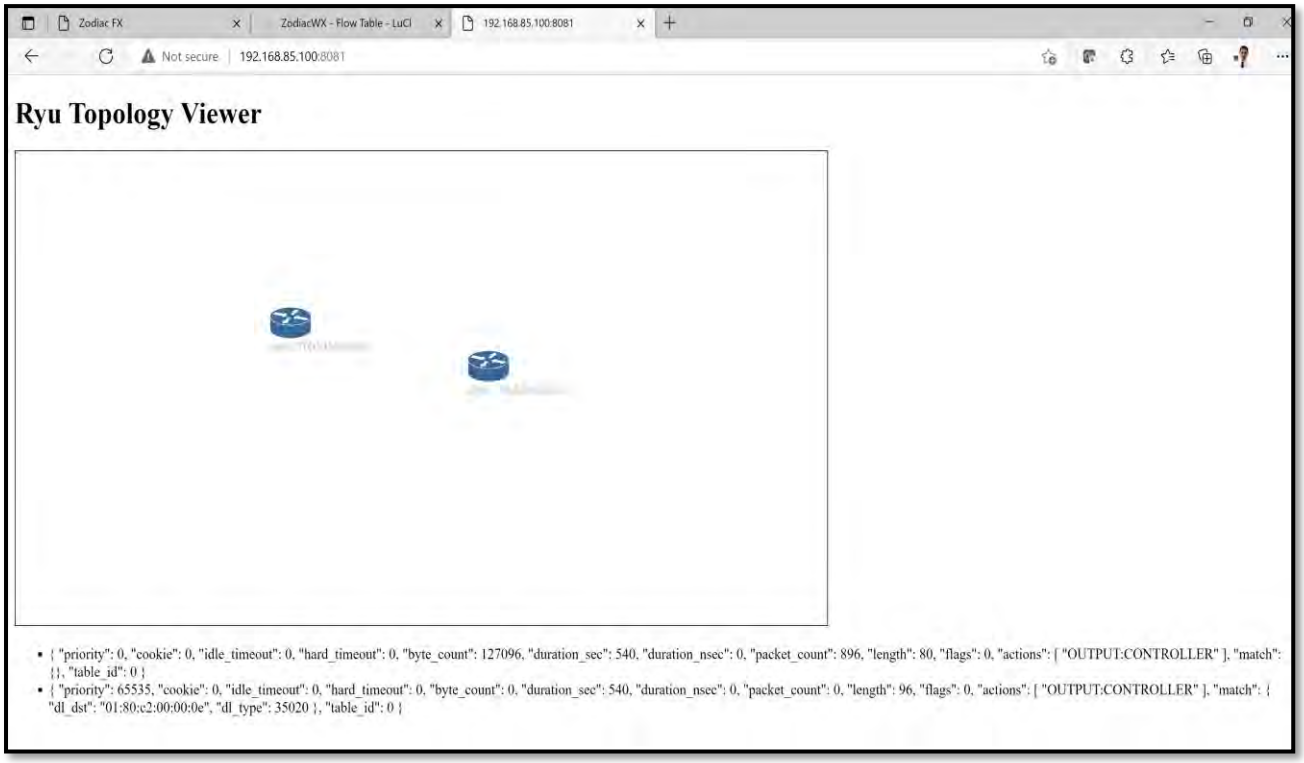
```
gustavosalazar@ubuntu:~/ryu$ ping 192.168.85.10
PING 192.168.85.10 (192.168.85.10) 56(84) bytes of data.
64 bytes from 192.168.85.10: icmp_seq=1 ttl=255 time=0.644 ms
64 bytes from 192.168.85.10: icmp_seq=2 ttl=255 time=0.667 ms
^C
--- 192.168.85.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1031ms
rtt min/avg/max/mdev = 0.644/0.655/0.667/0.011 ms
gustavosalazar@ubuntu:~/ryu$
gustavosalazar@ubuntu:~/ryu$ ping 192.168.85.20
PING 192.168.85.20 (192.168.85.20) 56(84) bytes of data.
64 bytes from 192.168.85.20: icmp_seq=1 ttl=64 time=0.833 ms
64 bytes from 192.168.85.20: icmp_seq=2 ttl=64 time=0.825 ms
64 bytes from 192.168.85.20: icmp_seq=3 ttl=64 time=0.773 ms
^C
--- 192.168.85.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2034ms
rtt min/avg/max/mdev = 0.773/0.810/0.833/0.026 ms
```

Figura H - 33 Conectividad entre RYU y equipos OpenFlow (Zodiac FX/WX)
Fuente: *Autor*

Para activar al controlador RYU, se usa el comando ***ryu-manager*** junto con el uso de una aplicación que le dotará a los equipos la funcionalidad de *Switches L2:* ***simple_switch_13,*** aplicación que hace referencia al *Switch* que se comunica con el controlador con *OpenFlow 1.3*.

Antes de usar *FlowManager* RYU también cuenta con un visualizador de redes basado en el servicio WSGI (*Web Server Gateway Interface*) conocido como RYU *Topology Viewer*.

```
gsalazar@ubuntu: ~/ryu
gsalazar@ubuntu:~/ryu$
gsalazar@ubuntu:~/ryu$
gsalazar@ubuntu:~/ryu$ sudo ryu-manager --ofp-tcp-listen-port 6633 --wsapi-port 8081 --observe-links
--app-lists ryu.app.simple_switch_13 ryu.app.ofctl_rest ryu.app.gui_topology.gui_topology
[sudo] password for gsalazar:
loading app ryu.app.simple_switch_13
loading app ryu.app.ofctl_rest
loading app ryu.app.gui_topology.gui_topology
loading app ryu.controller.ofp_handler
loading app ryu.app.rest_topology
loading app ryu.app.ofctl_rest
loading app ryu.app.ws_topology
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app None of Switches
creating context switches
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.app.ofctl_rest of RestStatsApi
instantiating app ryu.app.gui_topology.gui_topology of GUIServerApp
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.app.rest_topology of TopologyAPI
instantiating app ryu.app.ws_topology of WebSocketTopology
(11912) wsgi starting up on http://0.0.0.0:8081
(11912) accepted ('127.0.0.1', 34146)
127.0.0.1 - - [26/Jul/2021 16:23:12] "GET / HTTP/1.1" 200 515 0.008096
127.0.0.1 - - [26/Jul/2021 16:23:12] "GET /ryu.topology.css HTTP/1.1" 200 514 0.000571
(11912) accepted ('127.0.0.1', 34148)
127.0.0.1 - - [26/Jul/2021 16:23:12] "GET /ryu.topology.js HTTP/1.1" 200 8652 0.000476
```



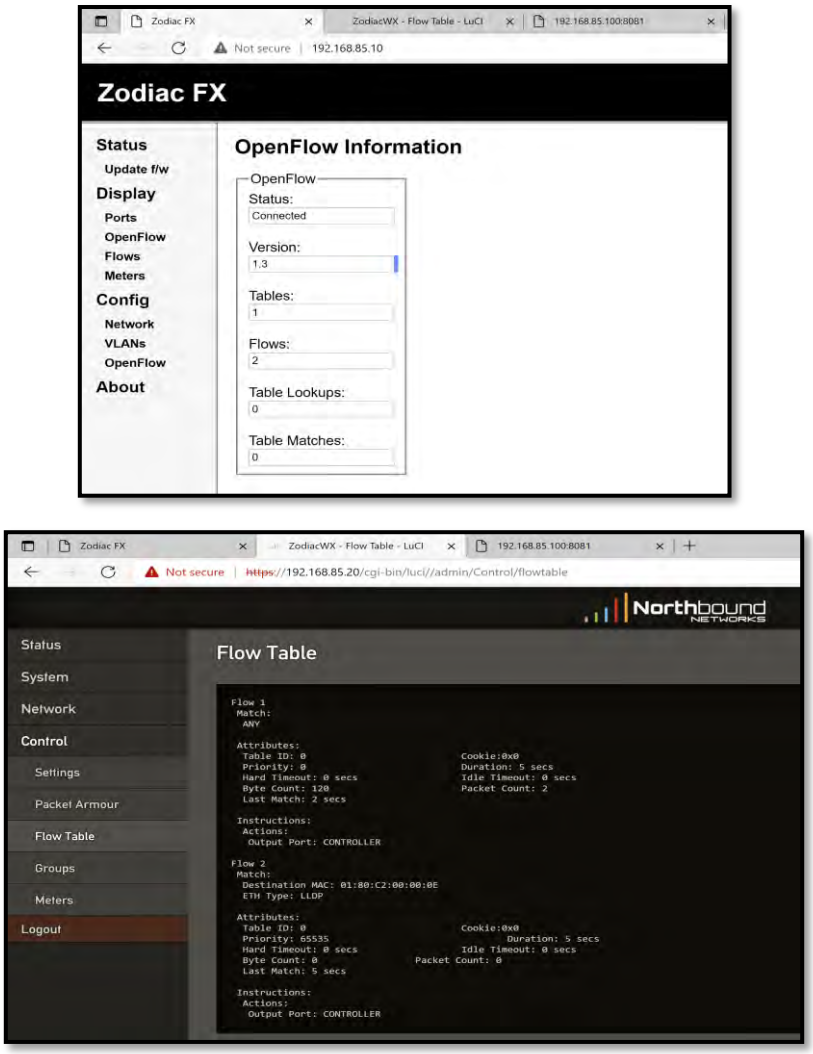


Figura H - 34 RYU Topology Viewer de Zodiac FX/WX
Fuente: *Autor*

Para activar *RYU* junto con *FlowManager* se requiere del siguiente comando:

```
gsalazar@ubuntu:~/ryu$ sudo ryu-manager --ofp-tcp-listen-port 6633 --observe-links --app-lists ~/ryu/
flowmanager/flowmanager.py ryu.app.simple_switch_13
[sudo] password for gsalazar:
loading app /home/gsalazar/ryu/flowmanager/flowmanager.py
You are using Python v3.8.10.final.0
loading app ryu.app.simple_switch_13
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
creating context wsgi
instantiating app None of DPSet
creating context dpset
instantiating app /home/gsalazar/ryu/flowmanager/flowmanager.py of FlowManager
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
(12533) wsgi starting up on http://0.0.0.0:8080
(12533) accepted ('192.168.85.253', 52173)
(12533) accepted ('192.168.85.253', 60108)
192.168.85.253 - - [26/Jul/2021 19:08:54] "GET /home/ HTTP/1.1" 200 1162 0.002203
192.168.85.253 - - [26/Jul/2021 19:08:54] "GET /home/css/style.css?v2 HTTP/1.1" 200 4550 0.002564
192.168.85.253 - - [26/Jul/2021 19:08:54] "GET /home/css/cards.css?v2 HTTP/1.1" 200 2652 0.003429
(12533) accepted ('192.168.85.253', 50930)
(12533) accepted ('192.168.85.253', 63969)
(12533) accepted ('192.168.85.253', 57960)
(12533) accepted ('192.168.85.253', 56475)
192.168.85.253 - - [26/Jul/2021 19:08:54] "GET /home/css/table.css?v2 HTTP/1.1" 200 2582 0.001997
192.168.85.253 - - [26/Jul/2021 19:08:54] "GET /home/css/menu.css?v2 HTTP/1.1" 200 1688 0.000610
192.168.85.253 - - [26/Jul/2021 19:08:54] "GET /home/js/jquery-3.3.1.min.js HTTP/1.1" 200 87068 0.005
```

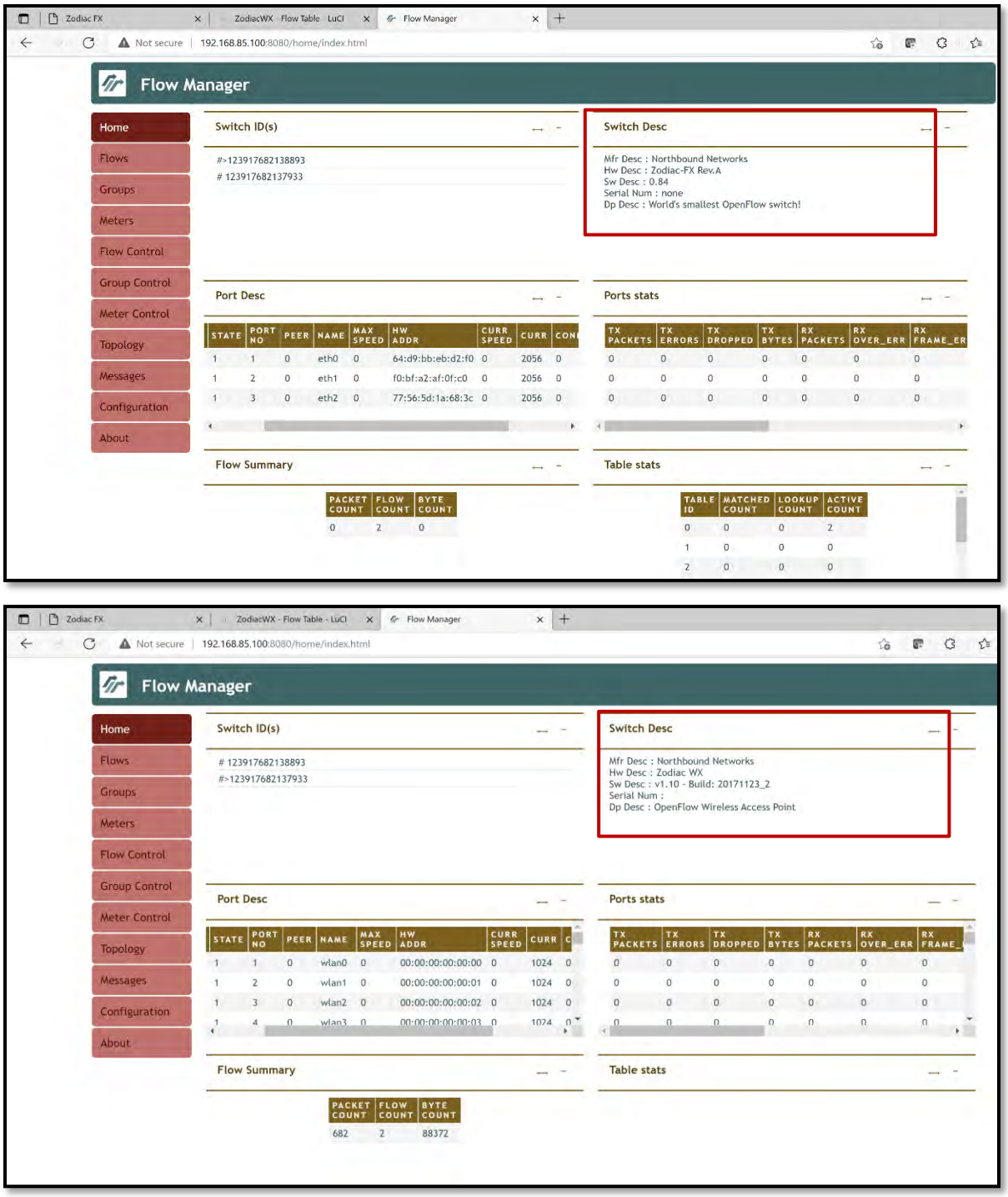


Figura H - 35 Flow Manager RYU SDN Controller - Zodiac FX/WX
Fuente: *Autor*

Las *Fig. H-34* y *H-35* para que pueda observarse adecuadamente la topología del Plano de Datos se sugiere un navegador distinto a Mozilla Firefox (para RYU *Topology Viewer* y *FlowManager*).

Página intencionalmente en blanco