ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

Diseño e Implementación de un prototipo de configuración para gestionar la red de un proveedor de internet de la ciudad de Guayaquil.

PROYECTO INTEGRADOR

Previo a la obtención del Título de:

INGENIERO EN TELECOMUNICACIONES

Autores:

Estuardo Ramiro Clavijo Perugachi

Ricardo Andrés Bravo Altamirano

GUAYAQUIL-ECUADOR

Año: 2023

DEDICATORIA

A mis queridos padres Rosa y Marcelo por enseñarme que todo logro requiere su cuota de esfuerzo, y que con perseverancia se puede alcanzar cualquier meta, porque siempre creyeron en mí y me apoyaron incesantemente a lo largo de esta carrera.

Estuardo Ramiro Clavijo Perugachi

DEDICATORIA

A mi familia, que siempre me apoyo en todo momento sin importar la dificultad y las adversidades.

Ricardo Andrés Bravo Altamirano

AGRADECIMIENTOS

Mi gratitud a toda mi familia por estar siempre presentes brindándome su apoyo incondicional en todo momento.

De igual manera mis agradecimientos a la Escuela Politécnica del Litoral, a la facultad de Facultad de Ingeniería en Electricidad y Computación, a mis profesores el PhD Washington Medina y a la Msc. Verónica Soto quienes han brindado todo los consejos y conocimientos necesarios para el desarrollo de este trabajo

Finalmente quiero expresar mis sinceros agradecimientos al Ing. Christian Jiménez quien con su amplia experiencia profesional cimentó las bases necesarias para que el presente proyecto haya podido realizarse.

Estuardo Ramiro Clavijo Perugachi

En primer lugar, agradecido con Dios, por bendecirme siempre y en todo lugar.

También agradecido con mi familia y amigos que siempre me apoyaron durante el camino, espcialmente agradezco a mis tíos Alfredo Altamirano e Inés Barahona, que estuvieron conmigo y me brindaron su apoyo en todo momento.

Al PhD Washington Medina y a la Msc. Verónica Soto por su paciencia y consejos que ayudaron a completar el desarrollo de este proyecto.

También estoy agradecido con la Escuela Superior Politécnica del Litoral, y con todos los profesores que tuve durante mi estancia aquí.

Ricardo Andrés Bravo Altamirano

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; Estuardo Ramiro Clavijo Perugachi y Ricardo Andrés Bravo Altamirano, damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual”

Estuardo Ramiro Clavijo Perugachi

Ricardo Andrés Bravo Altamirano

EVALUADORES

Firmado electrónicamente por:
WASHINGTON ADOLFO
MEDINA MOREIRA

Ph.D. Washington Adolfo Medina Moreira
PROFESOR DE LA MATERIA

Firmado electrónicamente por:
VERONICA ALEXANDRA
SOTO VERA

Msc. Verónica Alexandra Soto Vera
PROFESOR TUTOR

RESUMEN

En la actualidad, las redes informáticas ofrecen servicios como el internet, que hace posible que dispositivos de red puedan intercambiar información entre distintos puntos geográficos. Pese al avance de la tecnología, existen redes que se operan y mantienen de forma manual, es decir hay personal dedicado que se encarga de realizar un sin número de procedimientos o tareas a fin de mantener una red totalmente operativa. En el trabajo que se presenta a continuación, se desarrolló un método para automatizar procesos manuales en las redes mediante el uso de un lenguaje de programación. Se crearon algoritmos dedicados para realizar tareas específicas de forma automática, y todas las funcionalidades se probaron exhaustivamente en un entorno de red virtual que se construyó con el fin de obtener una respuesta lo más apegada a la realidad. Finalmente se demostró que los métodos manuales tradicionales para la actualización y configuración de equipos en una red son inferiores a la solución presentada, la cual fue desarrollada en un lenguaje de programación, siendo esta última aproximadamente un 91% más eficiente que el enfoque manual, obteniendo de esta manera beneficios para la empresa que implemente este tipo de soluciones.

ABSTRACT

Currently, computer networks offer services such as Internet, which makes it possible for network devices to exchange information between different geographical points. Despite the advance of technology, there are networks that are operated and maintained manually, that is, there are dedicated personnel who are in charge of carrying out a number of procedures or tasks in order to maintain a fully operational network. In the work presented below, a method was developed to automate manual processes in network operations through the use of a programming language. Dedicated algorithms were created to perform specific tasks automatically, and all functionalities were extensively tested in a virtual network environment that was built for the most realistic response. Finally, it was demonstrated that the traditional manual methods for updating and configuring equipment in a network are inferior to the solution presented, which was developed in a programming language, being approximately 91% more efficient than the manual approach, obtaining in this way benefits for the company that implements this type of solutions.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ANEXOS

CAPÍTULO 1

1. INTRODUCCIÓN

Las redes de comunicaciones se construyen a partir de una gran cantidad de dispositivos de red, como enrutadores, conmutadores y otros tipos de dispositivos intermedios (que manipulan el tráfico de la red, como los cortafuegos). Todos estos dispositivos interactúan entre sí mediante protocolos de comunicación.

Los administradores de red son los encargados de la configuración de los protocolos a fin de poder resolver las eventualidades que puedan presentarse dentro de la red. También tienen la labor de transformar políticas complejas en comandos de configuración básicos mientras se adaptan a los constantes cambios que ocurren en la red [1]. Con gran frecuencia, los administradores de red deben realizar estas complejas tareas a través de herramientas limitadas. Considerando lo anterior, la gestión y monitoreo de redes resultan en tareas de alta complejidad [1].

En general, las redes se operan y mantienen manualmente [2]. Los operadores de red ingresan a los distintos dispositivos uno por uno, realizan cambios en las configuraciones de los equipos de forma manual y luego salen de los dispositivos, y así continúan con los demás dispositivos.

Los procesos tradicionales para la configuración y la actualización de los dispositivos de red son manuales en su mayoría, lo cual implica que son procesos extremadamente lentos y propensos a errores [2], no cubren de manera efectiva los requisitos de transmisión de datos y aplicaciones que se encuentran en constante cambio [2]. Estos procesos imprácticos dificultan tareas como: ofrecer un servicio de alto nivel, implementar procedimientos para el registro de cambios en las configuraciones, gestionar efectivamente los inventarios de equipos de red, mantener los estándares de configuración, entre otros [2].

Debido a esto la automatización es una necesidad por sus beneficios, especialmente con el creciente número de dispositivos de red [3]. La palabra automático se define como actuar u operar de una manera independiente de la influencia externa y el control humano [4]. La automatización de redes ofrece miles de beneficios a las empresas [3]; permite la configuración de muchos dispositivos en cuestión de minutos [3], elimina la posibilidad de una configuración incorrecta originada por un error humano [3], por lo que puede reducir los gastos operativos, mejorar la seguridad, aumentar las tasas de productividad y, lo que es más importante, es un enfoque rentable que produce resultados consistentes y ofrece flexibilidad y resiliencia [5].

1.1. DESCRIPCIÓN DEL PROBLEMA

En el caso de una empresa proveedora de servicios de red de fibra óptica en la ciudad de Guayaquil, se requiere recopilar la información detallada acerca de los parámetros de red de todos los equipos CPE (Customer Premises Equipment o Equipo Local del Cliente), para posteriormente realizar un proceso de estandarización de configuraciones de los mismos. De momento, la empresa cuenta con una extensa red de fibra óptica de primer nivel con cobertura en todas las provincias del Ecuador continental y presencia en más de 150 ciudades, sumando en la actualidad un aproximado de 3000 equipos CPE que necesitan pasar por el proceso antes mencionado hasta el mes de febrero del año 2023. Al presente todo el proceso que conlleva la reconfiguración y levantamiento de información de cada equipo se realiza de forma manual con personal dedicado. Se estima que, si el proceso continúa realizándose de forma manual con el personal existente, este tomaría aproximadamente un año en completarse.

1.2. JUSTIFICACIÓN DEL PROBLEMA

El presente proyecto pretende encontrar alternativas al manejo y gestión manual de equipos en una red empresarial, los procesos de gestión manual de redes generan múltiples inconvenientes al momento de realizar ciertas tareas fundamentales , tales como: levantamiento de información de los equipos, implementar sistemas de control de cambios y configuraciones en los equipos de red [2], sostener la compatibilidad de las configuraciones entre equipos de plataformas de red distintas y mantener un servicio de alto nivel requerido por las empresas y los usuarios finales [2].

Una vez culminado el proceso que involucra todos los equipos CPE en la empresa, se logrará estandarizar los parámetros de la red, lo cual permitirá: documentar la red y acceder rápidamente a información acerca de la topología de red, mejorar la compatibilidad entre equipos de distintas marcas, identificar los servicios y las interfaces en la red según su función. Los aspectos anteriormente mencionados facilitarán la realización de: reportes, inventarios, reconfiguraciones de equipos, monitoreos de la red, etc.

1.3. OBJETIVOS

1.3.1. OBJETIVO GENERAL

Este proyecto tiene como objetivo general 'Automatizar procesos manuales usando lenguaje de programación de alto nivel', para recopilación de información de parámetros de red y posterior reconfiguración básica de dispositivos de una empresa proveedora de servicios de internet en Guayaquil". Para este propósito se detallan los siguientes objetivos específicos:

1.3.2. OBJETIVOS ESPECÍFICOS

- Emular una topología de red que con el fin de crear un entorno de trabajo controlado donde se puedan realizar pruebas exhaustivas a equipos y obtener una respuesta lo más cercana a la realidad.

- Desarrollar algoritmos e implementarlos mediante scripts (secuencia de comandos) que permitan automatizar procesos manuales en la gestión de redes.

- A partir de los algoritmos desarrollados, implementar cambios en las configuraciones de los equipos a nivel masivo.

- Desarrollar un estándar para la descripción de todas las interfaces de los equipos.

- Analizar y comparar los tiempos de configuración manual de los equipos con respecto a los tiempos de configuración del script.

1.4. METODOLOGÍA

Se necesita crear un entorno de trabajo controlado para testear las distintas funcionalidades que se realizaran a lo largo del proyecto para esto existen diversos softwares de simulación como: “GNS3” (Graphic Network Simulation o Simulación Gráfica de Redes) que imita o emula el hardware de un dispositivo y ejecuta imágenes reales en el dispositivo virtual.

Los scripts estarán desarrollados en algún lenguaje de programación de alto nivel como: C++, Fortran, Java, Perl, PHP, Python, entre otros. Estos programas se usan para un sin número de aplicaciones, sin embargo, Python resulta ser una combinación perfecta entre ser un lenguaje fácil de aprender con muchas muestras de código y utilidades que lo han convertido en un lenguaje de referencia para los ingenieros de redes.

1.5. ESTADO DEL ARTE

Hasta la fecha, se han desarrollado múltiples métodos para automatizar redes o sistemas, siendo el lenguaje de programación Python una opción popular para lograr esto [3]. Una investigación ha mostrado resultados y evidencias que muestran la viabilidad y factibilidad de desarrollar un programa de automatización de red en Python que ofrezca diferentes tipos de automatización en un entorno de múltiples proveedores [6]. Una investigación realizada en Suecia en el año 2020 demuestra que se puede usar la librería Netmiko en conjunto con el módulo de registro de Python para gestionar redes mediante SSH permitiendo así la comunicación con los dispositivos de red [6]. En los resultados obtenidos en el estudio anteriormente mencionado, se observó que el programa desarrollado logró realizar cambios de configuración exitosamente en equipos Cisco y Huawei, además de guardar copias de seguridad de las configuraciones de cada dispositivo [6].

Por otra parte, en 2017, en la Universidad Transilvania de Brasov, lograron crear una topología de red emulada en GNS3, teniendo como elemento principal unos puertos virtuales contenidos en una imagen de Ubuntu, con el rol de elemento controlador de red [7]. Con ello, lograron controlar los dispositivos de red, utilizando los paquetes de código abierto Netmiko y Paramiko, basados en Python. De este trabajo de investigación se concluyó que, al utilizar Python, ya no es necesario configurar cada dispositivo a la vez,

más bien se debe enfocar en la creación infraestructura de la red y en la correcta implementación del código [7].

También, el resultado de un estudio de 2021 demuestra que la automatización de procesos de red puede mejorar la eficiencia de las secuencias de comandos en la configuración de dispositivos de red, ofrece mayor velocidad y menor error en comparación con el método manual [3]. El rendimiento muestra que la automatización puede ahorrar hasta 5677 segundos de trabajo para configurar 36 dispositivos Cisco en un entorno real. El ahorro de tiempo es un elemento crucial que mejoraría la velocidad del trabajo realizado, aumentando así la productividad [3].

CAPÍTULO 2

2. MARCO TEÓRICO

En este capítulo desarrollaremos fundamentos teóricos para entender la propuesta de solución planteada al problema.

2.1. JERARQUÍA DE REDES

En redes informáticas, el diseño jerárquico consiste en dividir la red en niveles o capas [8]. Cada nivel posee funciones particulares dentro de la red. De esta manera un ingeniero en redes podrá optimizar y seleccionar características tanto en hardware y software que sean necesarios para realizar las funciones específicas de cada nivel en la red. Los modelos jerárquicos se aplican tanto a redes LAN (Local Área Network) como en redes WAN (Wide Area Network) [8].

Una red jerárquica facilita la gestión de la red. Además, se consigue brindar: redundancia, escalabilidad, rendimiento y capacidad de mantenimiento [9]. El diseño típico de red LAN jerárquica consta de tres capas: capa de acceso, capa de distribución, capa de núcleo [9].

2.1.1. CAPA DE ACCESO

La capa de acceso puede considerarse el punto de entrada a la red, aquí se conectan los dispositivos del usuario final [8]. Debido a la cantidad de dispositivos de usuario final que se conectan a la red, tiende a haber más switches en la capa de acceso que en cualquier

otra capa en un diseño de red jerárquico típico [9]. Requiere una alta densidad de puertos para admitir la gran cantidad de dispositivos conectados, pero generalmente no necesitan un alto rendimiento, ya que cada puerto se conecta a un solo dispositivo específico [10]. Estos dispositivos pueden ser cualquier dispositivo final que requiera una conexión de red, entre las más comunes están: computadoras portátiles, teléfonos inteligentes, tabletas e impresoras [10].

En esta capa se vincula al cliente a la red mediante “routers de acceso”, equipos que asignan IP a los clientes, limitan ancho de banda, brindan políticas de acceso entre otras funciones [9].

2.1.2. CAPA DE DISTRIBUCIÓN

En la capa de distribución generalmente se tendrá múltiples switches de capa 3 y enrutadores, cada uno conectado a uno o varios dispositivos en la capa de acceso [8]. La capa de distribución es la frontera entre el dominio de red de capa 2 y la red direccionada de capa 3 [9]. A la capa de distribución llegan los datos procedentes de los switches de la capa de acceso antes de que sean transferidos a la capa de núcleo para el direccionamiento hacia su destino final [9].

2.1.3. CAPA DE NÚCLEO

La capa de núcleo también se conoce como “backbone de red” consta de dispositivos de red de alta velocidad [9], diseñados para conmutar paquetes lo más rápido posible. El núcleo debe ser confiable y eficiente para maximizar el rendimiento y también estar disponible en todo momento [8]. Debe diseñarse con redundancias para que no

tenga un solo punto de falla. Si ocurriera un problema catastrófico, la recuperación debe ser rápida [8].

La capa central debe escalarse a través de la calidad (mejor equipo) en lugar de la cantidad (número de dispositivos), debe ser lo más delgada posible para minimizar el potencial de falla y maximizar la eficiencia [9].

2.2. MODELO ISP DE 3 NIVELES

Los ISP (Proveedor de servicios de internet) se clasifican en un modelo de 3 niveles, el cual los clasifica según el tipo de servicio de Internet proporcionado [11].

2.2.1. TIER 1 (NIVEL 1)

Los proveedores de Internet de nivel 1 se los denomina proveedores de red troncal [11]. Estos ISP construyen infraestructura a gran escala como es el tendido de cables submarino de Internet. Estos se encargan de proveer tráfico a todos los demás ISP, mas no a los usuarios finales [11]. Entre los principales ISP tier 1 se encuentran AT&T, Verizon, Sprint, NTT, Singtel, entre otros [11].

Un ISP de nivel 1 intercambia tráfico de Internet únicamente con otros proveedores de nivel 1 de forma no comercial a través de interconexiones privadas sin acuerdos [11]. También se interconectarán en Puntos de Intercambio de Internet (IXP). Los ISP de nivel 1 pueden ofrecer el mejor rendimiento de red a través de la red troncal de Internet y brindar conexiones privadas porque son

dueños de su infraestructura de red y tienen control directo sobre cómo fluye el tráfico a través de estas conexiones [12].

2.2.2. TIER 2 (NIVEL 2)

Un ISP de nivel 2 es un proveedor de servicios que utiliza una combinación de tránsito pagado a través de un ISP de nivel 1 junto con otros ISP de nivel 2 para entregar tráfico de Internet a los clientes finales a través de ISP de nivel 3 [11]. Los ISP de nivel 2 suelen ser proveedores regionales o nacionales. Solo unos pocos ISP de nivel 2 pueden brindar servicio a clientes en más de dos continentes. A menudo, tendrán velocidades de acceso más lentas que los ISP de nivel 1 y están al menos a un enrutador de distancia de la red troncal de Internet [12].

2.2.3. TIER 3 (NIVEL 3)

Un ISP de nivel 3 es un proveedor que compra estrictamente tránsito de Internet. Por definición, un proveedor de nivel 3 se dedica principalmente a brindar acceso a Internet a los clientes finales [11]. Los ISP de nivel 3 se enfocan en las condiciones del mercado comercial y de consumo local. Proporcionan el acceso local o a Internet para los clientes finales, a través de redes de acceso por cable, DSL (Digital Subscriber Line o Línea de Abonado Digital), fibra óptica o enlaces inalámbricos [12]. Su cobertura está limitada a países o subregiones específicos, como un área metropolitana. Los ISP de nivel 3 utilizan y pagan a los ISP de nivel superior por el acceso al resto de Internet [12].

2.3. MODELO OSI (OPEN SYSTEMS INTERCONNECT)

Es un estándar que divide el funcionamiento de una red en capas o niveles [13]. Las capas de este modelo son: capa física, capa de enlace de datos, capa de red, capa de transporte, capa de sesión, capa de presentación y capa de aplicación. Cada una de las capas tiene una función específica dentro de la red [13]. El modelo OSI sirve como referencia para fabricantes a fin de permitir la comunicación o interoperabilidad entre equipos de distintos proveedores o fabricantes [14].

2.4. PROTOCOLOS DE RED

Consisten en una serie de políticas donde se definen: procedimientos, formatos y condicionamientos que rigen el intercambio de información entre 2 o más terminales [15].

2.4.1. SSH (SECURE SHELL)

Consiste en un protocolo cuya función principal consiste en acceder de forma remota un servidor [16]. Cabe destacar que el acceso mediante SSH es totalmente seguro, debido a que la información se encuentra codificada, evitando cualquier tipo de filtración de información [17].

2.4.2. OSPF (OPEN SHORTEST PATH FIRST)

Consiste en un protocolo creado a partir del algoritmo de estado de Dijkstra, que calcula el camino más corto a cada uno de los dispositivos de la red a través de una métrica denominada como costo de la ruta [18]. A menor costo, se tiene una ruta más corta hacia el destino [18].

2.5. REDES LAN

Las redes LAN (Local Area Network) consisten redes con áreas físicas limitadas [19]. Redes domésticas o de pequeñas empresas son claros ejemplos de redes LAN [20]. Estas redes se conforman con mínimo 2 equipos conectados, y pueden llegar hasta abarcar hasta miles de equipos conectados, sin embargo, estos deben estar ubicados dentro de un área pequeña [20].

2.6. REDES WAN

Las redes WAN (Wide Area Network) consisten en redes con áreas físicas extensas, en el orden de países o continentes [21]. Generalmente una red WAN consiste en la unión de varias redes LAN a través de medios cableados como la fibra óptica, pudiendo intercomunicar usuarios situados a miles de kilómetros de distancia con altas tasas de transmisión [22].

2.7. LENGUAJES DE PROGRAMACIÓN

Consisten en palabras o instrucciones muy próximas al lenguaje de los humanos [23]. Estos no son dependientes de un hardware en particular, por lo tanto, los códigos elaborados en lenguajes de alto nivel pueden ser ejecutados en otros equipos con distinto hardware [23].

Los lenguajes de alto nivel permiten de forma muy sencilla manejar estructuras de datos en variables, y desempeñar acciones complicadas a través de pocas líneas de código [23]. Los lenguajes de programación más utilizados para automatizar redes son: Python, C/C++, Java [24].

2.7.1. LENGUAJES DE PROGRAMACIÓN DE BAJO NIVEL

Consisten en instrucciones, palabras o símbolos específicos, diseñados especialmente para un hardware en particular. Al ser creados solo para un hardware en concreto, son ejecutados rápidamente y permiten un máximo aprovechamiento de las especificaciones y características del mismo [25]. Sin embargo, al ser lenguajes dedicados para una sola máquina o arquitectura, estos no son utilizables en otros hardware [25].

2.7.2. LENGUAJES DE PROGRAMACIÓN DE ALTO NIVEL

Consisten en palabras o instrucciones muy próximas al lenguaje de los humanos [25]. Estos no son dependientes de un hardware en particular, por lo tanto, los códigos elaborados en lenguajes de alto nivel pueden ser ejecutados en otros equipos con distinto hardware [25].

Los lenguajes de alto nivel permiten de forma muy sencilla manejar estructuras de datos en variables, y desempeñar acciones complicadas a través de pocas líneas de código [23]. Los lenguajes de programación más utilizados para automatizar redes son: Python, C/C++, Java [23].

2.7.2.1. PYTHON

Es un lenguaje de programación multipropósito, muy sencillo de aprender y con un entorno amigable con los usuarios [26]. Cuenta con soporte para todos los paradigmas

de programación y es un lenguaje popular para automatizar redes [26]. Múltiples herramientas y librerías de redes como: Netmiko, Ansible, Napalm, Salt y pyATS están desarrolladas con Python [26].

2.7.2.2. C/C++

Son lenguajes de programación de propósito general [24]. Tienen una mayor complejidad de aprendizaje, sin embargo, los programas codificados con C/C++ son ejecutados en menor tiempo respecto de otros lenguajes [24]. Múltiples herramientas de redes definidas por software están desarrolladas en C/C++ [24].

2.7.2.3. JAVA

Consiste en un lenguaje de programación orientado a objetos [24]. Es un lenguaje de sencillo aprendizaje, sin embargo, no es tan sencillo como Python [27]. Es un lenguaje popular para el desarrollo de aplicaciones web y cuenta con herramientas para networking [24].

CAPÍTULO 3

3. DISEÑO DE LA SOLUCIÓN

En la actualidad, existe una tendencia en aumento de implementar la automatización en procesos manuales en el manejo de redes debido a los múltiples beneficios brindados como: reducción de tiempo en los procesos, menor propensión a errores, menor coste económico y demás. Sin embargo, adquirir alguno de los sistemas de automatización de redes existentes en el mercado resulta extremadamente costoso, debido a que utilizan software y hardware dedicado e importado de los países del primer mundo.

La empresa proveedora de servicios de red, objeto de este proyecto cuenta con una infraestructura de red de alto nivel, con aproximadamente 3000 equipos CPE desplegados a lo largo de las regiones costa, sierra y Amazonía del Ecuador. La empresa proveedora desea realizar una estandarización de las configuraciones de todos sus equipos de cliente, por lo tanto, se debe hacer un levantamiento de información para construir una base de datos actualizada de todos los equipos y con ello proceder a la reconfiguración de los equipos, a fin de cumplir con los nuevos estándares propuestos por la empresa.

3.1. AUTOMATIZACIÓN DE PROCESOS MANUALES

El presente proyecto de tesis tiene como finalidad automatizar procesos manuales en una empresa proveedora de servicios de internet, para esto se hará uso de un lenguaje de programación. El problema como tal ha sido descrito con más detalle en el capítulo 1 de este documento. Antes de ahondar más en la solución, es necesario describir el proceso a automatizarse, para esto se presenta el diagrama de flujo que describe dicho proceso.

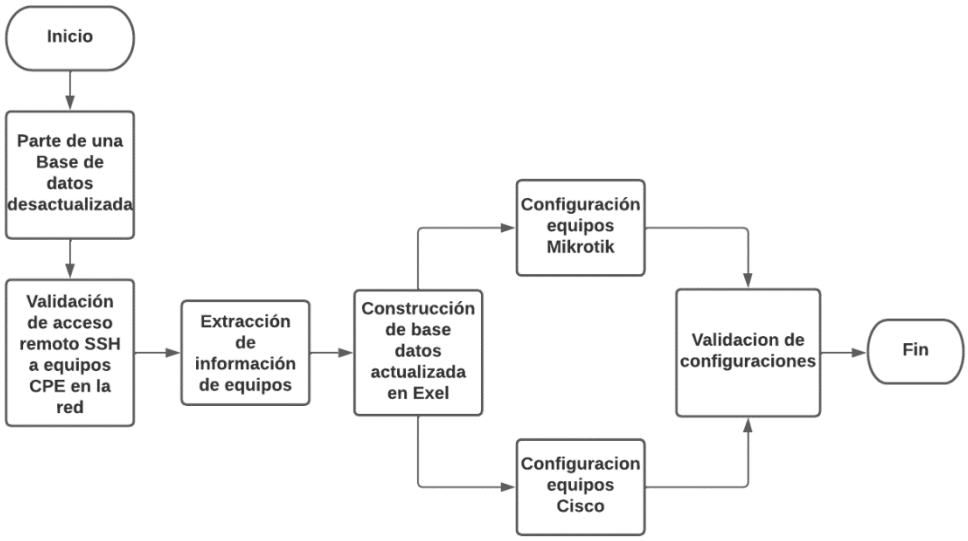

Figura 3-1. Diagrama de flujo de los procesos a automatizarse

En la figura 3-1 observamos el diagrama de procedimientos a llevar cabo por la empresa. El objetivo en esencia es configurar masivamente equipos de cliente que se encuentran en toda la red. Para esto se parte de una base de datos desactualizada, en la que se encuentra información de direccionamiento de equipos. Al realizar la validación de acceso a los equipos, si el acceso es exitoso, se extraerá información esencial del equipo con el fin de construir una base de datos actualizada.

La siguiente etapa en el proceso tiene que ver con la configuración de equipos, la empresa entrega a sus clientes equipos de marca Cisco o MikroTik. Por ello, la siguiente etapa del proceso se divide en dos, dado que ambas marcas manejan sistemas operativos distintos, el manejo de configuración tendrá que ser dedicado. Una vez se haya realizado las configuraciones se hace una validación de las mismas.

3.1.1. DISEÑO DE ALGORITMOS

La primera parte del proceso comprende la construcción de una base de datos, el escenario real es que la empresa cuenta con una base datos desactualizada, el fin del algoritmo es obtener información de cada equipo, y construir una nueva base de datos. La información que se obtiene es esencial para un administrador de red y facilita las labores de soporte técnico.

El algoritmo tiene como entrada una lista de direcciones IP de acceso a equipos de cliente. El primer paso será validar el acceso remoto a estos equipos, si el acceso es exitoso, entonces un script dedicado para cada marca ejecutara comandos en la terminal del equipo. Por cada comando que se envíe, el equipo retorna información, dicha información se guarda en formato de texto.

Luego, la información es tratada para finalmente obtener el parámetro de interés, todos los parámetros obtenidos de cada equipo se guardan de manera ordenada en un archivo csv y también exportara un archivo de Excel para una mejor visualización de la data. El diagrama de flujo que resume la ejecución del algoritmo se presenta en la figura 3-2.

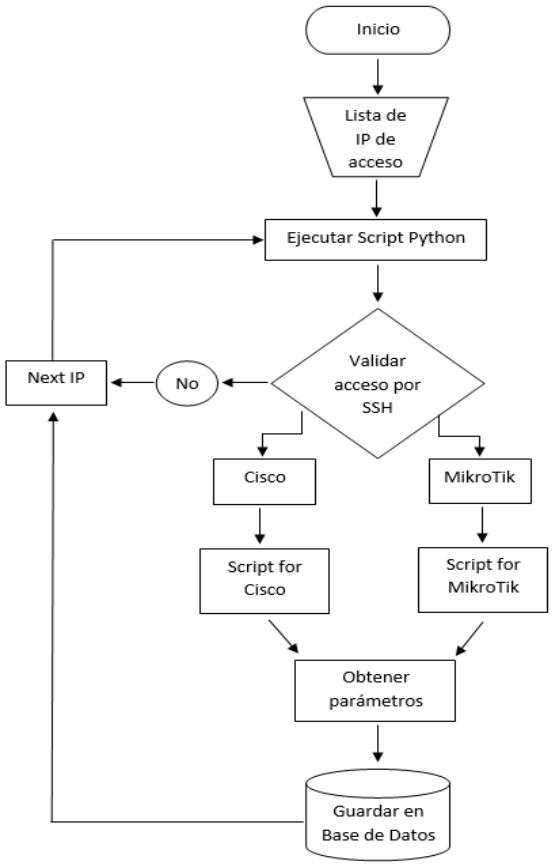

Figura 3-2. Algoritmo empleado para construcción de base de datos.

La segunda parte del proceso comprende realizar configuración de equipos, cualquier tipo de configuración que se vaya a realizar seguirá el diagrama de flujo presentado en la figura 3-3.

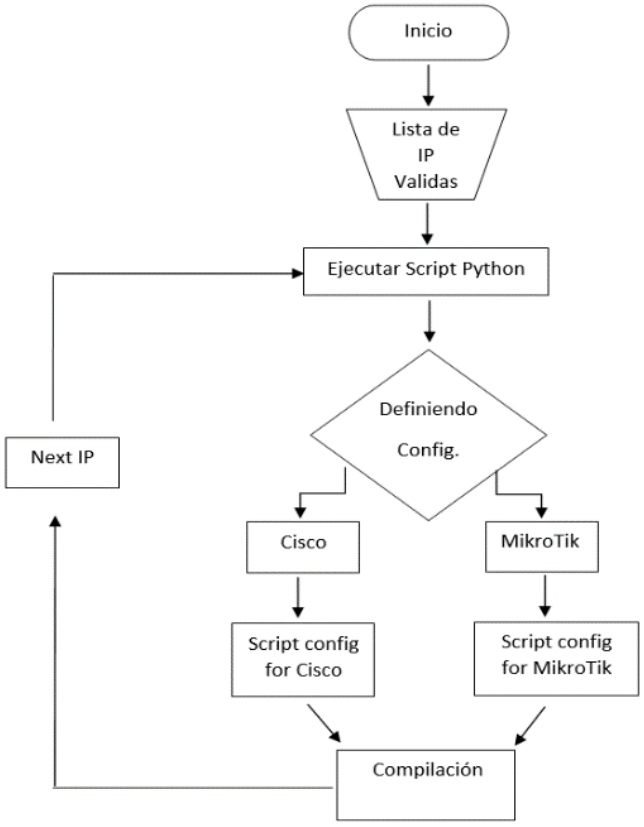

Figura 3-3. Algoritmo empleado para configurar equipos CPE.

En la primera parte se validó acceso remoto y se obtuvo información de cada equipo, por este motivo la entrada para este algoritmo de configuración será una lista de IP válidas. El tipo de configuración a realizarse puede variar, la configuración puede partir de algo básico como cambiar un hostname y escalar en dificultad como, configurar descripciones en las interfaces, configurar listas de acceso (ACL), crear usuarios, publicar redes, entre otras. Se ejemplificarán algunas de estas configuraciones en el capítulo 4 de este documento.

3.1.2. CONSTRUCCIÓN DE TOPOLOGÍA DE PRUEBA

En primer lugar, se necesita crear un entorno de trabajo controlado para testar las distintas funcionalidades que se realizarán a lo largo del proyecto. Para este fin, se utilizará el software GNS3, que permite emular equipos de red, es decir recrea el hardware de un dispositivo y ejecuta imágenes reales, obteniendo entonces un entorno cercano a la realidad.

Para poder simular la topología de red se necesita tomar recursos en hardware del ordenador donde se instalará GNS3. Cada imagen simulada consume memoria RAM y espacio en nuestro disco duro, los requerimientos recomendados para una simulación en GNS3 se detallan en la tabla 3-1.

Tabla 3-1. Requerimientos mínimos para poder simular en GNS3.

Ítem	Requerimiento
Sistema Operativo	Windows 7 (64 bit) o superior, GNS3 también soporta distribuciones de Linux Ubuntu/Debian
Procesador	4 o más núcleos lógicos - AMD-V / RVI Series o Intel VT-X / EPT
Virtualización	Extensiones de Virtualización (VMware; VirtualBox)
Memoria	16 GB RAM
Almacenamiento	SSD con al menos 35GB de espacio libre

La topología de red a utilizar para ese proyecto se verá limitada por los recursos de nuestro ordenador. Mientras más equipos se agregue a la topología, más memoria RAM necesitaremos y por ende más espacio en el disco duro. El consumo de RAM de las imágenes disponibles para este proyecto se detalla en la siguiente tabla:

Tabla 3-2. Lista de equipos a ser emulados en GNS3.

Equipo virtual	**Sistema Operativo**	**Consumo memoria RAM**
Cisco Nexus 9000 C9300v	NXOS versión 9.3(3)	8096 MB
Cisco CSR1000V (VXE)	Cisco IOS-XE	3072 MB
Cisco 3725	Cisco IOS	200 MB
Cisco 3745	Cisco IOS	256 MB
Cisco 7200	Cisco IOS	512 MB
MikroTik	RouterOS	384 MB
Cisco IOSv	Cisco IOS	800 MB
VPC	-	2 MB

Asumiendo que solo se empleara una imagen de cada equipo virtual, necesitaremos al menos 17GB de RAM para poder construir una topología simple. El proyecto abarca la simulación de una topología más elaborada que solo se verá limitada por las características de nuestro ordenador que se adjunta en la tabla 3-3.

Tabla 3-3. Especificaciones técnicas del computador donde se ejecuta GNS3

Ítem	Requerimiento
Sistema Operativo	Ubuntu 20.04.5 LTS
Procesador	Intel Core i7 10ma generación, 8 núcleos, Frecuencia Base 3.80GHz
Virtualización	VMware Workstation 16 Pro
Memoria	32 GB RAM
Almacenamiento	SSD de 512 GB

Es de suma importancia indicar esto, debido a que el diseño de la topología de la red de prueba va en estrecha relación con los recursos que se encuentran disponibles.

En la tabla 3-4 se muestra los equipos virtuales a ser empleados en la simulación del proyecto y el consumo de memoria RAM que implica simular dichas imágenes.

Tabla 3-4. Características de los equipos utilizados en la topología de prueba.

Equipo virtual	**Sistema Operativo**	**Cantidad**	**Consumo memoria RAM**	**Funcionalidad**	**Nombre en Topología**
Cisco Nexus 9000 C9300v	NXOS versión 9.3(3)	1	8096 MB	Switch de distribución capa 3 (principal)	GYQ-NORTE
Cisco Nexus 9000 C9300v	NXOS versión 9.3(3)	1	8096 MB	Switch de distribución capa 3 (backup)	GYQ-SUR
Cisco CSR1000V (VXE)	Cisco IOS-XE	2	6144 MB	Router de acceso (Equipo Cliente)	CPEa, CPEe
Cisco 3725	Cisco IOS	2	400 MB	Router de acceso (Equipo Cliente)	CPEc,CPEg
Cisco 3745	Cisco IOS	1	256 MB	Router de acceso (Equipo Cliente)	CPEi
Cisco 7200	Cisco IOS	1	512 MB	Router de acceso (Equipo Cliente)	CPEd
Mikrotik	RouterOS	5	1920 MB	Router de acceso (Equipo Cliente)	CPEb,CPEf, CPEh,CPEj, CPEk
Cisco IOSv	Cisco IOS	2	1600 MB	Switches capa 2 para redundancia	SW1, SW2
VPCs	-	11	22 MB	PCs virtuales representa red LAN cliente	LANa,LANb, LANc…LANf
TOTAL		**26**	**27046 MB**		

El diseño de la topología de prueba tiene estrecha relación con el consumo de recursos de los equipos a simular (ver tabla 3-4). Considerando lo anteriormente mencionado, se llegó a una topología con un total de 26 dispositivos. De todos ellos, 4 constituyen la capa de núcleo y distribución de la red, 11 equipos actúan como equipos de cliente y finalmente 11 máquinas virtuales simulan la red LAN del cliente.

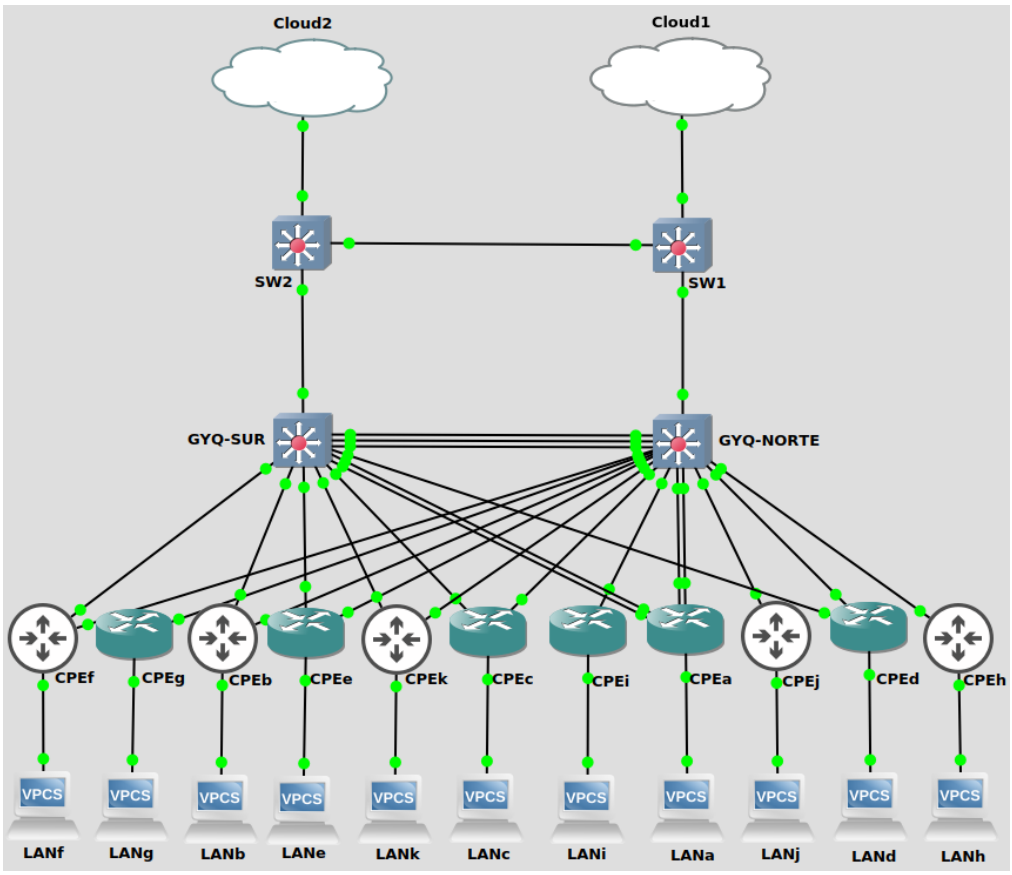

Figura 3-4. Topología de red de prueba

Observando la figura 3-4 de los 11 equipos de cliente mostrados en la topología (CPEa, CPEb, …CEPk), 6 equipos son de la marca Cisco y 5 equipos son de la marca MikroTik. La topología en general cuenta con redundancia, a fin de garantizar la conectividad en todo momento. Los equipos CPE junto con las PC virtuales conforman la capa de acceso de esta topología de red. Adicional, tenemos dos switches de capa 3 (capa de red del modelo OSI), que conforman la capa de distribución y núcleo de la red. Los elementos Cloud1 y Cloud2 representan la conexión hacia una red exterior a la topología. Para poder interconectar los equipos en la topología, se ha empleado direccionamiento estático y protocolo de enrutamiento OSPF.

En la topología se emplean dos equipos Cisco Nexus nombrados GYQ-NORTE y GYQ-SUR, estos reciben todo el tráfico que generan los CPE en la capa de acceso y lo distribuyen a capas superiores del modelo jerárquico. El direccionamiento que fue asignado a las interfaces de estos equipos se detalla en las siguientes tablas:

Tabla 3-5. Direccionamiento asignado al equipo GYQ-NORTE.

Dispositivo	Interfaz	IP	Conexión con
GYQ-NORTE	Po1	172.16.10.1/30	CPEa
	Po20	172.16.100.1/30	GYE-SUR
	Eth1/3	172.16.10.5/30	CPEb
	Eth1/4	172.16.10.9/30	CPEc
	Eth1/5	172.16.10.13/30	CPEd
	Eth1/6	172.16.10.17/30	CPEe
	Eth1/7	172.16.10.21/30	CPEf
	Eth1/8	172.16.10.25/30	CPEg
	Eth1/9	172.16.10.29/30	CPEh
	Eth1/10	172.16.10.33/30	CPEi
	Eth1/11	172.16.10.37/30	CPEj
	Eth1/12	172.16.10.41/30	CPEk

Tabla 3-6. Direccionamiento asignado al equipo GYQ-SUR

Dispositivo	Interfaz	IP	Conexión con
GYQ-SUR	Po1	172.16.11.1/30	CPEa
	Po20	172.16.100.2/30	GYE-NORTE
	Eth1/3	172.16.11.5/30	CPEb
	Eth1/4	172.16.11.9/30	CPEc
	Eth1/5	172.16.11.13/30	CPEd
	Eth1/6	172.16.11.17/30	CPEe
	Eth1/7	172.16.11.21/30	CPEf
	Eth1/8	172.16.11.41/30	CPEk

3.1.3. DESARROLLO DE SCRIPTS

Existen diversos lenguajes de programación para automatizar redes, entre ellos: C/C++, Java, Python, Perl, entre otros. Sin embargo, el lenguaje más destacado para este fin es Python, esto debido a que es un lenguaje de programación fácil de aprender y tiene múltiples referencias de código disponibles, lo que lo ha convertido en un lenguaje de referencia para la ingeniería en redes. Actualmente, existen herramientas para el manejo de redes desarrolladas en Python, entre ellas podemos mencionar las librerías más conocidas que son: Netmiko y Paramiko, que nos permiten simplificar el proceso de conexión de los dispositivos de red además de facilitar el manejo y gestión de los equipos de red. Por todos los factores anteriormente mencionados, se decidió desarrollar los scripts en el lenguaje de programación Python.

CAPÍTULO 4

4. RESULTADOS Y ANÁLISIS

En este apartado analizaremos los resultados obtenidos del sistema de automatización de procesos manuales para el levantamiento de información y estandarización de configuraciones de los equipos CPE de la empresa proveedora de servicios de red. Para las pruebas correspondientes, se utilizó la topología de red de la figura 3-4.

4.1. IMPLEMENTACIÓN DE TOPOLOGÍA DE PRUEBA

El montaje del laboratorio de prueba se ajusta al sistema que muestra la figura 4-1, las imágenes de los equipos se encuentran virtualizadas en el programa VMware Workstation. La topología se construye en GNS3 y un servidor montado en Ubuntu será el encargado de todo el procesamiento de los equipos virtuales. Para tener control y acceso a la topología virtual se utiliza programas como PuTTY, Winbox, Python y Visual Studio Code. Todos estos programas se ejecutan dentro de un computador con sistema operativo Windows.

Para poder comunicar la topología virtual alojada en el servidor Ubuntu con la PC Windows, se usa un router físico Mikrotik como medio de interconexión. Con este sistema se consigue controlar la topología virtual alojada en un servidor Ubuntu de manera remota desde un sistema operativo Windows.

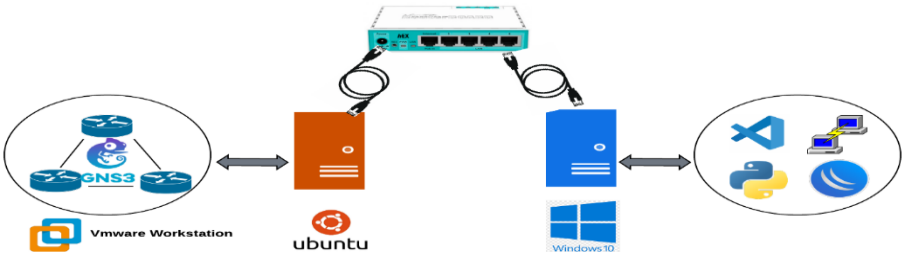

Figura 4-1. Esquemático de la implementación de Laboratorio de pruebas.

4.2. TOPOLOGÍA FUNCIONAL

Como se detalló anteriormente, un router físico será el medio de intercomunicación por el cual se accede y gestiona los equipos virtuales. Un router físico maneja todas las rutas para poder alcanzar los equipos virtuales desde la computadora Windows hacia la topología virtual en Ubuntu. De esta manera se logra tener control remoto de la red por medio de SSH usando programas como PuTTY y Visual Studio Code.

En la figura 4-2 se muestra la tabla de enrutamiento del router físico, dicha tabla aloja todas las rutas hacia los equipos virtuales.

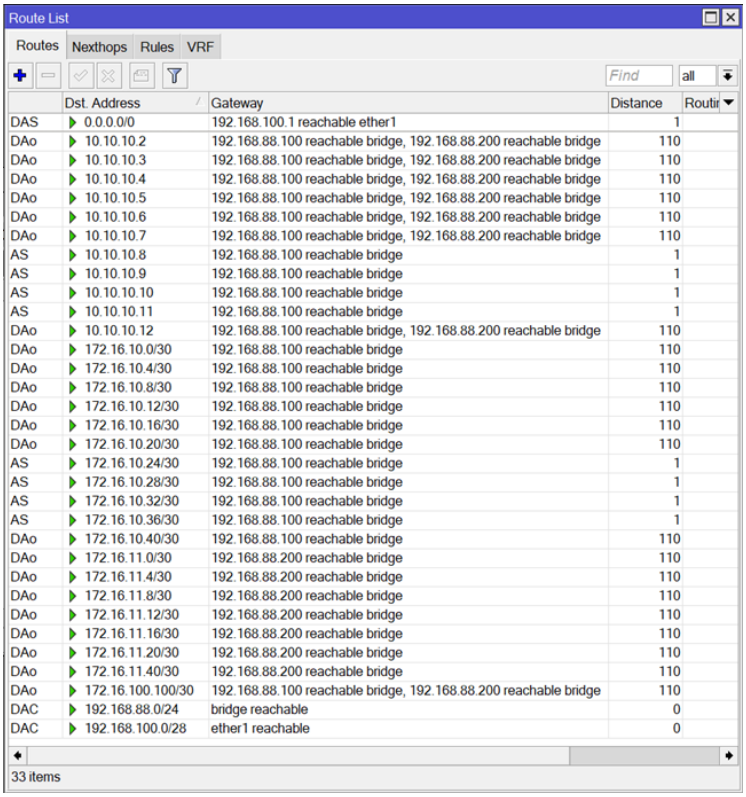

Figura 4-2. Tabla de enrutamiento del router físico MikroTik

La administración de los equipos virtuales se hace por protocolo SSH desde la PC Windows. El router físico MikroTik es el encargado de dar una ruta hacia la topología virtual, con la conexión ya establecida en la figura 4-3, se aprecia una prueba de conectividad y acceso al equipo virtual CPEa, para esta prueba la conexión se establece desde la terminal CMD de la

computadora Windows. De esta manera se comprueba el acceso remoto a los equipos virtualizados en el servidor Ubuntu.

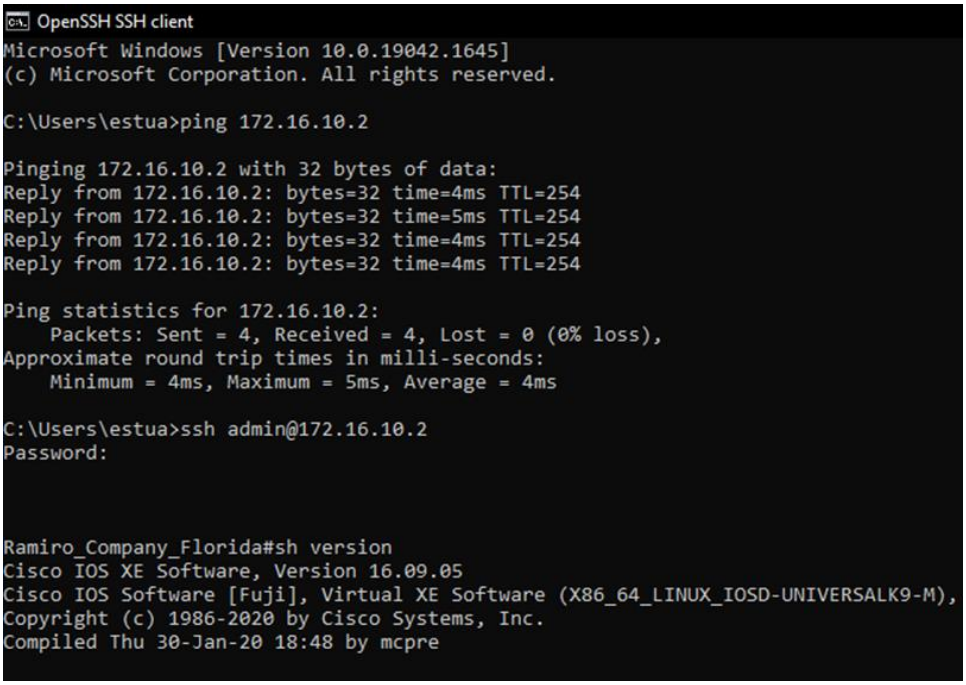

Figura 4-3. Validación de acceso mediante SSH desde PC Windows hacia topología en servidor Ubuntu.

4.3. CONSTRUCCIÓN DE BASE DE DATOS

El algoritmo implementado para la construcción de base de datos empieza validando acceso remoto a los equipos. En la figura 4-4 se observa cómo se encuentran programadas las sesiones SSH. En esta primera parte, el código es capaz de autentificar usuario/contraseña y discrimina si el equipo es de la marca Cisco o MikroTik.

```
#SSH session parameters CISCO
cisco_router = {
    'device_type': 'cisco_ios',
    'host': ip,
    'username': 'admin',
    'password': 'test',
    'port': 22,
}
try:
    net_connect = netmiko.ConnectHandler(**cisco_router)
    router="cisco"
except:

#SSH session parameters MIKROTIK
    mk_router = {
        'device_type': 'mikrotik_routeros',
        'host': ip,
        'username': 'admin',
        'password': 'test',
        'port': 22,
    }
    net_connect = netmiko.ConnectHandler(**mk_router)
    router="mikrotik"
```

Figura 4-4. Parte del código que permite validar acceso remoto a los equipos y determinar si el dispositivo es Cisco o MikroTik

Si algoritmo accede a un equipo, los comandos a enviarse se encuentran previamente programados. Estos comandos se envían hacia la terminal de cada equipo y la información que nos despliega cada comando se guarda en un archivo de texto plano. En esta parte no se realiza ninguna clase de configuración, es por esto que los comandos que se envían hacia los equipos son puramente informativos del tipo “show”. En la figura 4-5 se muestra los comandos programados para extraer información básica de un equipo Cisco

```
if router=="cisco":

    #Comandos para obtener inf
    C_version  ='show version'
    C_ospf     ='show run | section ospf'
    C_hostname ='show running-config | include hostname'
    C_Lo10     ='show ip int brief | i Loopback10'

    #Enviamos los comandos al equipo CISCO
    GET_IOS_MODEL= net_connect.send_command(C_version)
    comando_ospf = net_connect.send_command(C_ospf)
    GET_HOSTNAME = net_connect.send_command(C_hostname)
    GET_int_Lo10 = net_connect.send_command(C_Lo10)
```

Figura 4-5. Comandos utilizados para obtener información de equipos Cisco.

La figura 4-6 muestra la información que despliega el comando “show versión” en un equipo Cisco, esta información se guarda en un archivo de texto dentro del directorio Windows donde se está ejecutando el código. Este proceso se realiza para cada comando ejecutado en la terminal de un equipo.

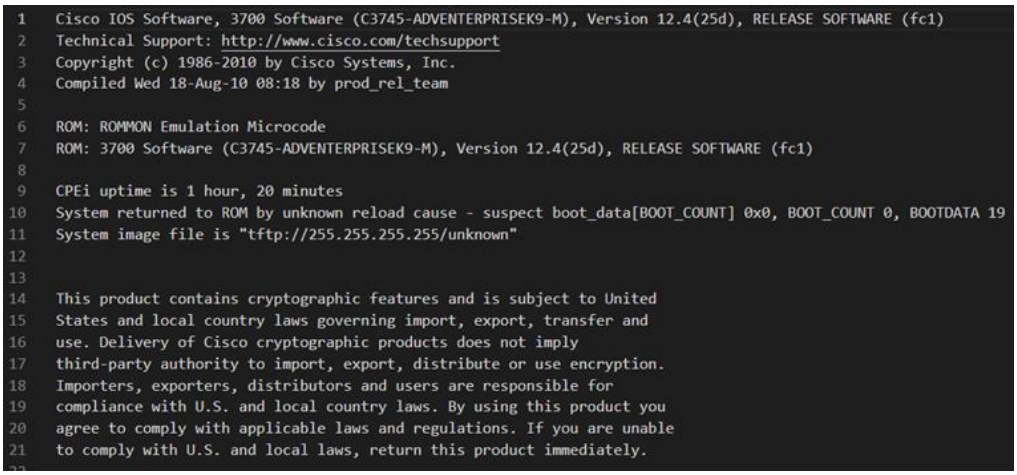

Figura 4-6. Información en texto que retorna equipo Cisco

Con la información extraída de cada equipo y almacenada en archivos de texto, el siguiente paso en el algoritmo es buscar los parámetros de interés dentro de estos archivos, para esto se desarrolla código capaz de identificar y extraer solo el parámetro de interés. En la figura 4-7 se observa parte del código que es capaz de discriminar el modelo de un equipo Cisco

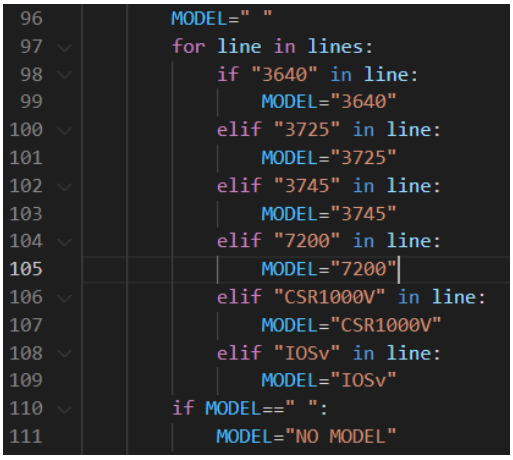

Figura 4-7. Parte de código utilizado para obtener modelo de equipos Cisco.

La información que se obtiene de un equipo se guarda en un diccionario, que contendrá de manera ordenada cada uno de los parámetros obtenidos. Se adjunta en la figura 4-8 parte del código que construye el diccionario.

```
170         Diccionario_info = {"IP":ip,
171                             "HOSTNAME":GET_HOSTNAME,
172                             "IOS":IOS,
173                             "MODEL":MODEL,
174                             "ROUTING":routing,
175                             "Loopback10":Lo10
176         }
177         List_Dct.append(Diccionario_info)
178
179         net_connect.disconnect()
180
```

Figura 4-8. Parte de código utilizado para crear diccionarios con información de equipos.

Conforme se extrae información de cada equipo, esta se imprime en pantalla y empieza a construirse la base de datos. En la figura 4-9 se observa la información que se obtuvo de dos equipos: uno marca Cisco y uno marca MikroTik.

```
IOS:     cisco_xe
ROUTING:  ospf
MODEL:   CSR1000V
Loopback:  10.10.10.2
####################--------------DATA_BASE-------------####################
            0                     1         2         3        4           5
0           IP               HOSTNAME       IOS     MODEL  ROUTING  Loopback10
1  172.16.10.2  Ramiro_Company_Florida  cisco_xe  CSR1000V     ospf  10.10.10.2
################################################################################
mikrotik
add address=10.10.10.3 interface=Loopback10 network=10.10.10.3
################################
IP:   172.16.10.6
HOSTNAME ConstructoraSA_centro
MODEL:  RouterOSv
ROUTING: ospf
################################
####################--------------DATA_BASE-------------####################
            0                     1              2         3       4           5
0           IP               HOSTNAME             IOS     MODEL  ROUTING  Loopback10
1  172.16.10.2  Ramiro_Company_Florida        cisco_xe  CSR1000V     ospf  10.10.10.2
2  172.16.10.6   ConstructoraSA_centro  RouterOS 6.48.6  RouterOSv     ospf  10.10.10.3
################################################################################
```

Figura 4-9. Información obtenida de un equipo Cisco y un MikroTik

Cuando el script haya finalizado obtendremos un archivo csv como el que detalla la figura 4-10. En este archivo se encuentra contenida la información de todos los equipos CPE de la topología virtual.

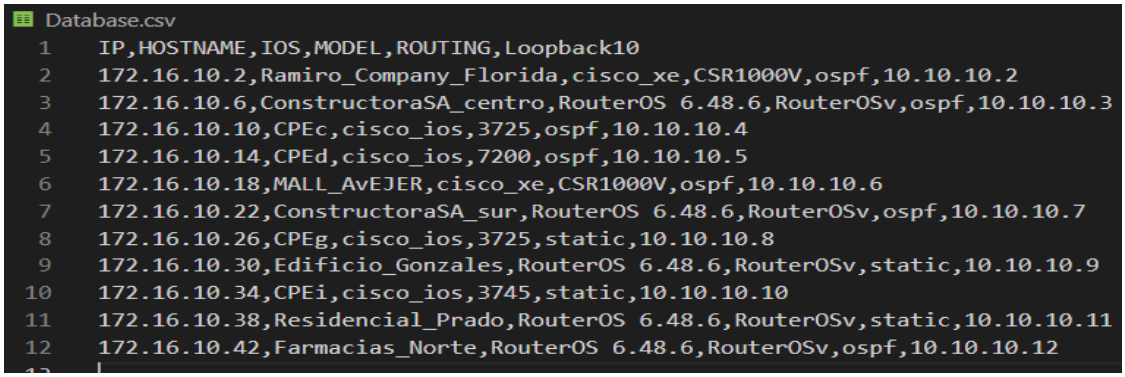

Figura 4-10. Base de datos de los equipos CPE en formato .csv

El script también retorna un archivo en formato Excel, con el que se puede apreciar de mejor manera la base de datos obtenida.

	IP	HOSTNAME	IOS	MODEL	ROUTING	Loopback10
2	172.16.10.2	Ramiro_Company_Florida	cisco_xe	CSR1000V	ospf	10.10.10.2
3	172.16.10.6	ConstructoraSA_centro	RouterOS 6.48.6	RouterOSv	ospf	10.10.10.3
4	172.16.10.10	CPEc	cisco_ios	3725	ospf	10.10.10.4
5	172.16.10.14	CPEd	cisco_ios	7200	ospf	10.10.10.5
6	172.16.10.18	MALL_AvEJER	cisco_xe	CSR1000V	ospf	10.10.10.6
7	172.16.10.22	ConstructoraSA_sur	RouterOS 6.48.6	RouterOSv	ospf	10.10.10.7
8	172.16.10.26	CPEg	cisco_ios	3725	static	10.10.10.8
9	172.16.10.30	Edificio_Gonzales	RouterOS 6.48.6	RouterOSv	static	10.10.10.9
10	172.16.10.34	CPEi	cisco_ios	3745	static	10.10.10.10
11	172.16.10.38	Residencial_Prado	RouterOS 6.48.6	RouterOSv	static	10.10.10.11
12	172.16.10.42	Farmacias_Norte	RouterOS 6.48.6	RouterOSv	ospf	10.10.10.12

Figura 4-11. Base de datos de los equipos CPE en formato Excel

El tiempo que toma extraer la información y añadirla a una base de datos, para 11 equipos CPE en la topología construida, se midió mediante contadores implementados en el código, el tiempo de compilación se detalla en la siguiente tabla

Tabla 4-1. Tiempos de compilación del script por cada equipo CPE en la topología

IP del Equipo	Tiempo de compilación
172.16.10.2	22.29 s
172.16.10.6	22.22 s
172.16.10.10	3.97 s
172.16.10.14	22.31 s
172.16.10.18	4.40 s
172.16.10.22	22.31 s
172.16.10.26	11.45 s
172.16.10.30	6.89 s
172.16.10.34	7.83 s
172.16.10.38	23.43 s
172.16.10.42	11.86 s
TOTAL	159.01 s

El proceso manual implica entrar a cada equipo por SSH, escribir y ejecutar comandos en la terminal del equipo para obtener la información que se necesita, para luego almacenarla manualmente en un archivo Excel o csv.

El tiempo que se emplea para realizar la misma acción de manera manual depende en parte del modelo de equipo y de la habilidad propia de cada operante o trabajador. Luego de haber realizado pruebas de gestión manual a lo largo del proyecto, el tiempo promedio por modelo de equipo para obtener los mismos 5 parámetros y construir la base de datos de manera manual se detalla en la tabla 4-2.

Tabla 4-2. Tiempos de construcción de base datos de forma manual por modelo de equipo

Modelo Equipo	Cantidad de equipos en topología	Tiempo aproximado por equipo	Tiempo de obtención manual de información
CSR1000V	2	180 s	360 s
RouterOSv	4	150 s	600 s
3725	2	200 s	400 s
7200	1	250 s	250 s
3745	1	220 s	220 s
TOTAL			1830 s

En definitiva, si se construye la misma base de datos de manera manual emplearíamos 1830 segundos (aproximadamente 30.5 minutos). Por otro lado, utilizando el script desarrollado, solo se emplean 159 s (aproximadamente 2,65 minutos). En sí, este es un gran resultado, si comparamos ambos tiempos existe una ventaja del script sobre el proceso manual de aproximadamente 27.85 minutos.

Comparando ambos resultados se puede calcular la eficiencia del script sobre el proceso manual con la ecuación siguiente.

$$\%Eficiencia = \left(1 - \frac{Tiempo\ de\ compilación\ script}{Tiempo\ de\ obtención\ manual}\right) \cdot 100\% = \left(1 - \frac{2.65}{30.85}\right) \cdot 100\% = 91.3\%$$

Se evidencia que el script resulta ser un 91% más rápido sobre el proceso manual. Ahora extrapolando resultados y aplicando a la problemática de la empresa proveedora de servicios de internet a fin a este proyecto, si lo que se requiere es obtener información de 3000 equipos y el número de parámetros de interés incrementa a 20, por regla de tres compuesta podemos estimar con mucha precisión el tiempo que emplearía un script desarrollado con las mismas técnicas presentadas para realizar esta acción.

$$x = \frac{(20)(3000)(159.01s)}{(5)(11)} = 173465.45\ s \approx 48\ h$$

Con el mismo método podemos estimar el tiempo que tomaría la construcción manual de una base datos para 3000 equipos.

$$x = \frac{(20)(2000)(1830.01s)}{(5)(11)} = 1996363.64\ s \approx 554h$$

Con los resultados presentados podemos asegurar que el código desarrollado reducirá el tiempo al menos 10 veces lo que tome hacer el mismo proceso, pero de forma manual.

4.4. ESTANDARIZACIÓN DE CONFIGURACIONES

Llevar un estándar en las configuraciones de equipos ayuda a los administradores de la red a brindar una mejor calidad de soporte y resume en gran medida el tiempo de solución de problemas de un equipo.

4.4.1. ESTANDARIZACIÓN DE DESCRIPCIONES

Cuando un administrador brinda soporte o mantenimiento a un equipo, entre los primeros pasos está el ejecutar el comando “show interfaces description” para equipos Cisco o su equivalente en equipos Mikrotik “interfaces print”. La información que despliegan estos comandos sirve para conocer el estado actual de la interfaz (up/down), mientras que la descripción de la interfaz da información acerca de la función que podría estar cumpliendo dicha interfaz en el equipo. La figura 4-12, muestra la información desplegada por un equipo Cisco 3725, simulado en la topología, luego de ejecutarse el comando “sh int description”.

```
CPEc#sh int description
Interface                      Status         Protocol Description
Fa0/0                          up             up       WAN_GYQ_NORTE
Se0/0                          admin down     down     -
Fa0/1                          admin down     down     -
Se0/1                          admin down     down     -
Se0/2                          admin down     down     -
Fa1/0                          up             up       WAN_GYQ_SUR
Fa2/0                          up             up       LAN_CLIENTE
Lo10                           up             up       administracion
CPEc#
```

Figura 4-12. Descripciones no estandarizadas de las interfaces de un equipo Cisco 3725

Si se observa la figura 4-12, es posible apreciar las descripciones de las interfaces. Si se toma, por ejemplo, la interfaz Fa0/0, se observa que dicha interfaz se encuentra operativa (“up”), es decir en funcionamiento. También se observa la descripción “WAN_GYQ_NORTE”, da una referencia muy importante debido a que indica hacia dónde va conectada dicha interfaz. El escenario real de un proveedor de servicios de internet es que las descripciones de las interfaces llegan a ser muy variables, puesto que cada persona, administrador o ingeniero en soporte puede describir las interfaces como crea conveniente. Como consecuencia de ello, existen conflictos para llevar una correcta documentación de la red o para obtener información básica de la topología de la red. De la misma manera el trabajo de una persona que realiza soporte se ve afectado por la falta de un orden o un estándar.

Lo que se presenta en la tabla 4-3, es un estándar de descripciones que se implementó en todos los equipos de nuestra topología usando algoritmo desarrollados en Python.

Tabla 4-3. Estándar desarrollado para descripción de interfaces en equipos CPE en la topología

Tipo de interfaz /función	**Descripción**
Red interna al cliente, red (LAN)	"RIC-IDC:"+IDENTIFICADOR_CLIENTE+"-DES:"
Red externa al cliente, red (WAN)	"REC-INTF:LY3-ID:"+IDENTIFICADOR_CLIENTE+"-DES:"
Interfaz lógica para gestión (Loopback)	"INTL: GESTION-DES: ACCESO_CPE"
Interfaz down	INACTIVA

El código desarrollado para estandarizar descripciones de interfaces cuenta con un total de 430 líneas de código para equipos Cisco y 200 líneas de código para equipos MikroTik (el código completo estará adjunto al final del documento). El fragmento del código que se observa en la figura 4-13, muestra en esencia lo que

realiza el script. Una vez el código haya extraído información acerca de las interfaces de un equipo, esta información se guarda en un archivo de texto plano, en este archivo se encuentra información detallada acerca del estado en que se encuentran todas las interfaces de un equipo además de las descripciones actuales de cada interfaz.

Lo que realiza el código es analizar a detalle la información para decidir qué tipo de descripción debe llevar cada interfaz, es decir si reconoce que la interfaz pertenece a la red WAN la describirá conforme a la tabla 4-3, y así para los distintos casos.

```
#Creando una Lista con todas las int down

f = open("fileName3.txt", "r")        #Leyendo el archivo
lineas = f.readlines()                #Cada linea una lista

List_UNUSED=[]                        #Lista down
List_USED=[]                          #Lista up
List_comment=[]                       #Lista de comentarios int up
List_comment_down=[]                       #Lista de comentarios int up
for i in lineas:
    i=i.strip()                       #Elimina salto de linea
    count_down=i.count("down")        #Contando int down
    if count_down >= 2:               #Si es mayor igual 2
        ns = " ".join( i.split() )     #Quita los espacios demas
        list_ns=ns.split(" ")          #Crea una lista
        List_UNUSED.append(list_ns[0]) #añadiendo Lista int down
        if len(list_ns)>=4:            #Si mayor 4 append comentario
                List_comment_down.append(list_ns[3])
        else:
                List_comment_down.append(" ") #Caso contrario append " "
    else:
        ns = " ".join( i.split() )
        list_ns=ns.split(" ")
        List_USED.append(list_ns[0])
        if len(list_ns)>=4:          #Si mayor 4 append comentario
            List_comment.append(list_ns[3])
        else:
            List_comment.append(" ") #Caso contrario append " "
```

Figura 4-13. Fragmento del código utilizado para estandarizar descripciones en equipos Cisco

Una vez ejecutado el código que describe interfaces para un equipo Cisco, se obtienen las siguientes descripciones:

```
Interface                      Status         Protocol Description
Fa0/0                          up             up       REC-INTF:LY3-ID:GYQ2730CSCO-DES:
WAN_GYQ_NORTE
Se0/0                          admin down     down     INACTIVA
Fa0/1                          admin down     down     INACTIVA
Se0/1                          admin down     down     INACTIVA
Se0/2                          admin down     down     INACTIVA
Fa1/0                          up             up       REC-INTF:LY3-ID:GYQ2730CSCO-DES:
WAN_GYQ_SUR
Fa2/0                          up             up       RIC-IDC:GYQ2730CSCO-DES:LAN_CLIE
NTE
Lo10                           up             up       INTL:GESTION-DES:ACCESO_CPE
CPEc#
```

Figura 4-14. Descripciones estandarizadas del equipo Cisco 3725 (CPEc)

El estándar de descripciones también se realiza para equipos de la marca MikroTik. Parte del código desarrollado se muestra en la figura 4-15, el código difiere en gran medida al presentado en la figura 4-13 puesto que, en un sistema operativo totalmente distinto, pero la finalidad del código es exactamente la misma, describir interfaces según el estándar descrito en la tabla 4-3

```
101  #-----------------INTERFACE_MK----------------------#
102  ########################################################
103
104  file1 = open("f4.txt", "r")      #Leyendo txt
105  lines = file1.readlines()        #Lista de lineas
106  lines=lines[2:]
107  #print(lines[8])
108
109  ether_list=[]
110  List_up=[]
111  List_dow=[]
112  List_up_comment=[]
113  for i in lines:
114      i=i.strip()             #Elimina salto de linea
115      splt=i.split()          #Quita los espacios demas
116      print(splt)
117      if "R" in splt[1]:      #Valido R como up
118          List_up.append(splt[0])
119          List_up_comment.append(splt[3])
120      else:                   #Los demas estan down
121          List_dow.append(splt[0])
122
123  List_del_dow=[]
```

Figura 4-15. Fragmento del código utilizado para estandarizar descripciones en equipos MikroTik

En la figura 4-16 se muestra una captura de las interfaces del equipo MikroTik CPEb simulado en la topología.

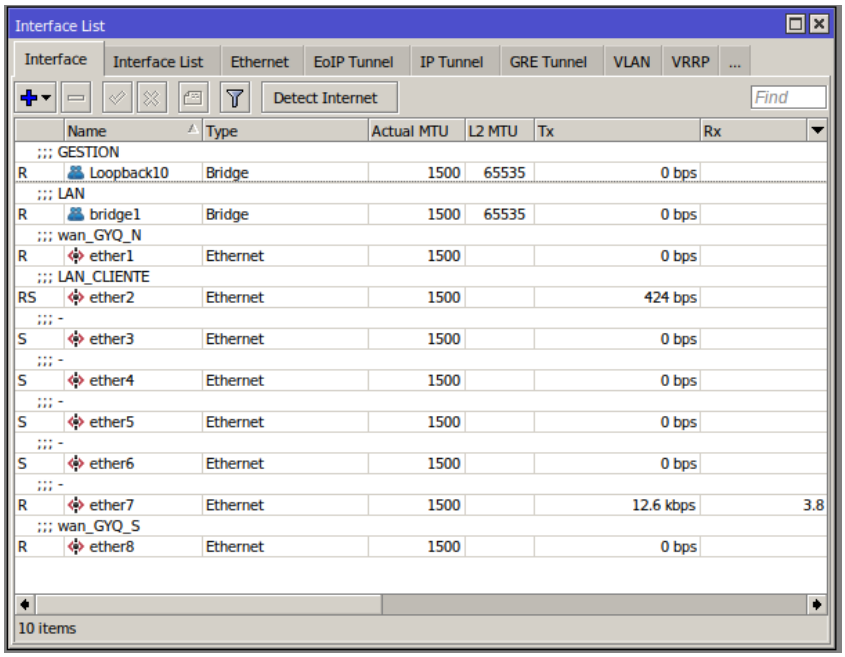

Figura 4-16. Descripciones de interfaces MikroTik

Luego de ejecutarse el Código que describe interfaces para un equipo MikroTik se obtiene el resultado que muestra la figura 4-17

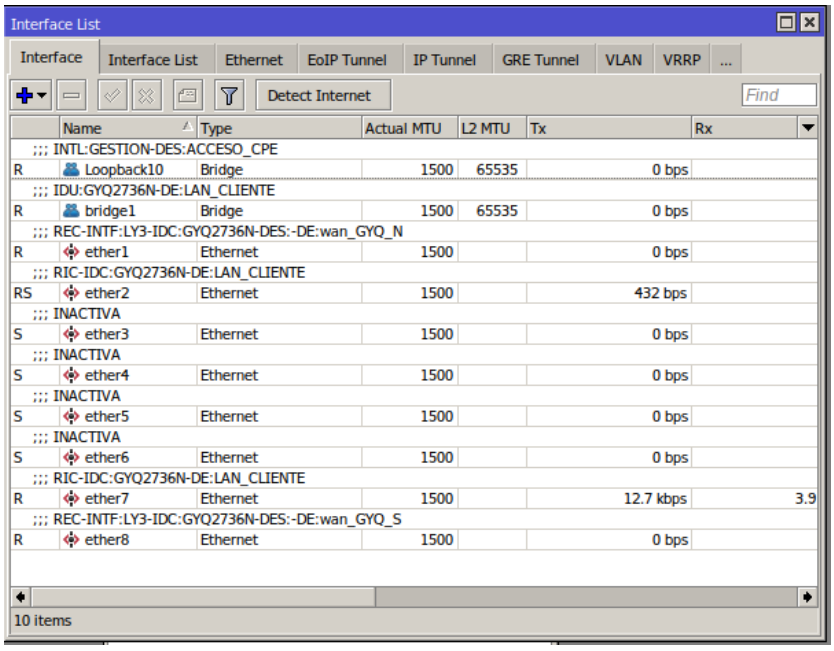

Figura 4-17. Descripción de interfaces estandarizadas del equipo MikroTik CPEb.

4.4.2. ESTÁNDAR DE CONFIGURACIÓN PARA PARÁMETROS DE INTERÉS

Si se necesita que todos los equipos cumplan con un estándar de configuraciones, estas pueden ser diversas. Se ha mostrado como primer ejemplo, cómo es posible llevar un estándar para la descripción de interfaces. El algoritmo de configuración que se presentó en la figura 3-3 es muy flexible, basta con decidir qué tipo de configuración se va a implementar, se valida un código que permita realizar dicha configuración y se envía a todos los equipos.

En este apartado realizaremos tres tipos de configuraciones para ejemplificar la flexibilidad del algoritmo de configuración, la primera será crear una interfaz de red virtual Loopback20 en el rango 10.20.20.x, se añadirán listas de acceso, y se cambiará el hostname del equipo por un hostname estandarizado.

En la figura 4-18 se observa fragmento específico del código que será el encargado de configurar una nueva Loopback en equipos Cisco.

```
#----------------------#
#------CONFIG_Lo20------#
#----------------------#

i=str(ip+1)
Lo20="10.20.20."+i

file = open("config_Lo20.txt","w+") #Creando el script que se enviara

file.write("Configure terminal")
file.write("\n")
file.write("interface Loopback20")
file.write("\n")
file.write("ip address"+" "+Lo20+" "+"255.255.255.255")
file.write("\n")
file.write("end")
file.write("\n")
```

Figura 4-18. Código para configurar una interfaz Loopback

En la figura 4-19 se observa fragmento específico del código que será el encargado de configurar las listas de acceso en equipos Cisco.

```
#----------------------#
#---------ACL----------#
#----------------------#

file.write("Configure terminal")
file.write("\n")
file.write("ip access-list extended ssh-access")
file.write("\n")
file.write("permit ip 172.16.10.0 0.0.0.255 any")
file.write("\n")
file.write("permit ip 162.16.10.0 0.0.0.255 any")
file.write("\n")
file.write("permit ip 152.16.10.0 0.0.0.255 any")
file.write("\n")
file.write("permit ip 192.168.88.0 0.0.0.255 any")
file.write("\n")
file.write("permit ip host 10.10.10.1 any")
```

Figura 4-19. Código para configurar listas de acceso

En la figura 4-20 se observa fragmento específico del código que será el encargado de cambiar el hostname actual por uno que siga un estándar, para equipos Cisco.

```
#----------------------#
#--------HOSTNAME-------#
#----------------------#

file.write("hostname"+" "+IDENTIFICADOR_CLIENTE)
file.write("\n")
file.write("end")
file.write("\n")
file.close()
```

Figura 4-20. Código para estándar de hostname

Conforme se va realizando la configuración a todos los equipos se ha decido que el script imprimirá en pantalla el tiempo de compilación individual de cada equipo, además de imprimir las direcciones IP asignadas a cada interfaz a fin de constatar que la nueva Loopback se esté configurando correctamente.

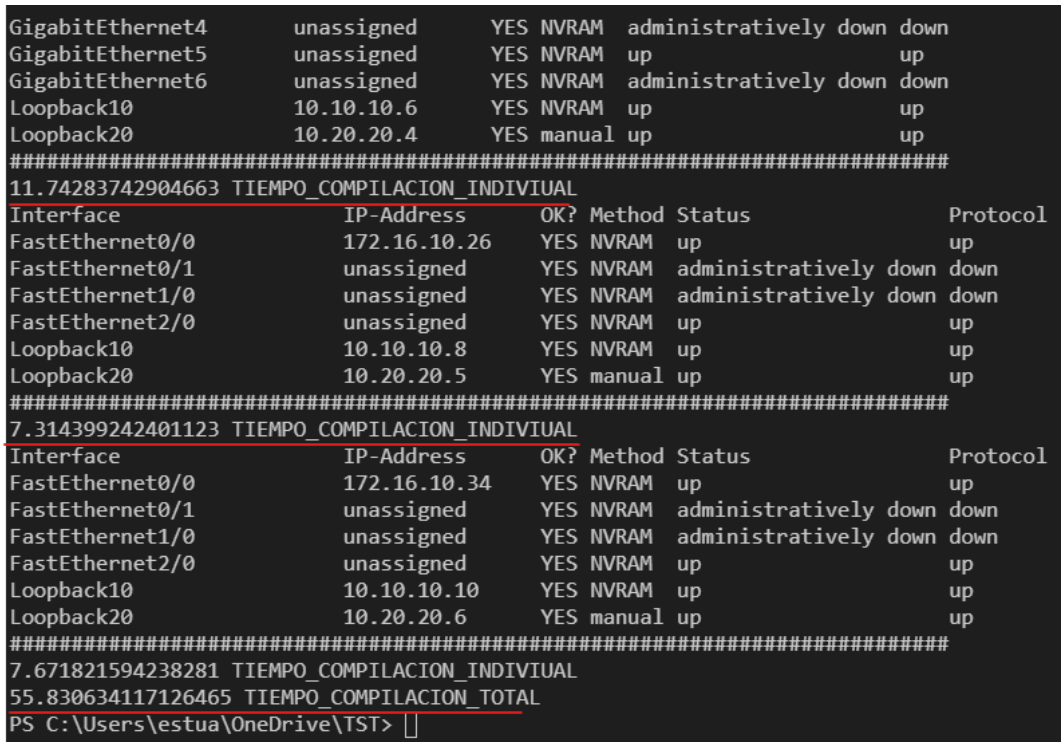

Figura 4-21. Tiempo de Compilación de script

Como se observa en la figura 4-21, realizar estos 3 cambios de configuración a los 6 equipos Cisco en la topología toma un total de 55.83s, podemos aproximar y decir que toma alrededor de 10 s configurar cada uno de los 6 equipos. Al comparar el tiempo de compilación del script con el tiempo que tardaría una persona en realizar los mismos cambios a todos los equipos, el resultado del proceso manual podría escalar en tiempo como mínimo unas 10 veces, es decir 10 minutos como se ilustró en el apartado 4.3 de este capítulo.

4.5. CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

Se pudo emular de manera exitosa una topología de red en GNS3 utilizando imágenes virtuales de equipos reales, con lo cual se creó un entorno controlado para poder probar las distintas funcionalidades de los scripts desarrollados en Python.

Se logró desarrollar a completitud scripts de Python totalmente funcionales para equipos Cisco y MikroTik, con lo que se pudo automatizar procesos manuales en la gestión de redes como: ingreso de credenciales, automatización de configuraciones a través de comandos de línea, extracción de valores de parámetros de red y creación de base de datos.

Se demostró que se puede configurar y tener gestión de varios equipos a la vez a través de scripts, logrando así alcanzar una estándar de configuraciones.

Se redujo en al menos un 91% el tiempo de obtención de los parámetros de red, construcción de una base de datos y configuración de equipos con ayuda de los scripts, porcentaje obtenido al comparar los 159.01 segundos que toma al script recopilar la información de los 11 equipos CPE de la topología de red respecto a los 1830 segundos que se tarda obtener la misma información de forma manual.

RECOMENDACIONES

El desarrollo de algoritmos y desarrollo de código podría resultar más eficiente usando técnicas matemáticas y de programación más avanzadas de este modo se puede llegar a un algoritmo y un tiempo de compilación de código mucho más eficiente.

Si se requiere realizar trabajos semejantes al presentado, se recomienda ampliamente implementar el laboratorio de prueba dentro de un sistema operativo basado en Linux, dado que las imágenes de los equipos en un entorno Windows generan innumerables problemas.

BIBLIOGRAFÍA

[1] Bruno Astuto A. Nunes Marc Mendonca, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks", *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, vol. 16, n.º 3, 2014.

[2] Red Hat Ansible, "Automatización de Red para todos", 2022, ma-network-automation-for-everyone-e-book-f14954-201812-a4-es, Estados Unidos. Accedido el 16 de octubre de 2022. [En línea]. Disponible:

[3] A. Mahmood Mazin. "Performance Analysis on Network Automation Interaction with Network Devices Using Python". IEEE Xplore. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9431823 (accedido el 16 de octubre de 2022).

[4] M. Dyer et al., "Deployment support network," in European Conference on Wireless Sensor Networks, 2007: Springer, pp. 195-211.

[5] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," IEEE Communications surveys & tutorials, vol. 16, no. 3, pp. 1617- 1634, 2014.

[6] J. Larsson, "Network Automation in a Multi-vendor Environment," ed: Digitala Vetenskapliga Arkivet, 2020.

[7] P. Mihăilă, T. Bălan, R. Curpen, and F. Sandu, "Network Automation and Abstraction using Python Programming Methods," MACRo 2015, vol. 2, no. 1, pp. 95-103, 2017

[8] A. Walton. "Diseño Jerárquico de Redes". CCNA desde cero. https://ccnadesdecero.es/diseno-jerarquico-de redes/#31_Capa_de_acceso (accedido el 14 de noviembre de 2022).

[9] "Hierarchical Network Design Overview (1.1) > Cisco Networking Academy Connecting Networks Companion Guide: Hierarchical Network Design | Cisco Press". Cisco Press: Source for Cisco Technology, CCNA, CCNP, CCIE Self-Study |CiscoPress. https://www.ciscopress.com/articles/article.asp?p=2202410&seqNum=4 (accedido el 16 de noviembre de 2022).

[10] "Hierarchical Network Design Overview (1.1) > Cisco Networking Academy Connecting Networks Companion Guide: Hierarchical Network Design | Cisco Press". Cisco Press: Source for Cisco Technology, CCNA, CCNP, CCIE Self-Study | Cisco Press. https://www.ciscopress.com/articles/article.asp?p=2202410&seqNum=4 (accedido el 16 de noviembre de 2022).

[11] "Internet Service Provider 3-Tier Model | ThousandEyes". Digital Experience Monitoring | ThousandEyes. https://www.thousandeyes.com/learning/techtorials/isp-tiers#:~:text=A%20Tier%203%20ISP%20is,business%20and%20consumer%20market%20conditions (accedido el 16 de noviembre de 2022).

[12] "Tier 1, Tier 2 and Tier 3 ISP: The Three Tiers of ISPs » Network Interview". Network Interview. https://networkinterview.com/tier-1-tier-2-and-tier-3-isp/ (accedido el 14 de noviembre de 2023).

[13] "¿Qué es el modelo OSI?" IONOS Digital Guide. https://www.ionos.es/digitalguide/servidores/know-how/el-modelo-osi-un-referente-para-normas-y-protocolos/ (accedido el 6 de febrero de 2023).

[14] "Modelo OSI - Concepto, Como funciona, para qué sirve y capas". Concepto. https://concepto.de/modelo-osi/ (accedido el 6 de febrero de 2023).

[15] "Protocolos de redes: guía completa con los protocolos básicos". RedesZone. https://www.redeszone.net/tutoriales/internet/protocolos-basicos-redes/ (accedido el 7 de febrero de 2023).

[16] "SSH: qué es y cómo funciona este protocolo - Blog de arsys.es". Blog de arsys.es. https://www.arsys.es/blog/ssh (accedido el 7 de febrero de 2023).

[17] "Qué es y para qué sirve el SSH". RedesZone. https://www.redeszone.net/tutoriales/internet/protocolo-ssh-usos/ (accedido el 7 de febrero de 2023).

[18] "Qué es OSPF y Cómo Funciona OSPF - CCNA Desde Cero". CCNA Desde Cero. https://ccnadesdecero.com/curso/ospf/ (accedido el 7 de febrero de 2023).

[19] "Red LAN - Concepto, tipos, topologías y qué es Internet". Concepto. https://concepto.de/red-lan/ (accedido el 7 de febrero de 2023).

[20] "LAN — Red de área local: la tecnología de un vistazo". IONOS Digital Guide. https://www.ionos.es/digitalguide/servidores/know-how/lan/ (accedido el 7 de febrero de 2023).

[21] "Guía de redes WAN: configuración y características - Tokio School". Tokio School. https://www.tokioschool.com/noticias/guia-de-redes-wan-configuracion-y-caracteristicas/ (accedido el 7 de febrero de 2023).

[22] "¿Qué es una red de área amplia (WAN)?" IONOS Digital Guide. https://www.ionos.es/digitalguide/servidores/know-how/wan/ (accedido el 7 de febrero de 2023).

[23] "Qué es un lenguaje de programación". OpenWebinars.net. https://openwebinars.net/blog/que-es-un-lenguaje-de-programacion/ (accedido el 7 de febrero de 2023).

[24] "12 Hot Programming Languages For Infrastructure Pros". Network Computing. https://www.networkcomputing.com/data-centers/12-hot-programming-languages-infrastructure-pros (accedido el 7 de febrero de 2023).

[25] "Tipos de lenguajes de programación". Home de DesarrolloWeb.com. https://desarrolloweb.com/articulos/2358.php (accedido el 7 de febrero de 2023).

[26] R. Donato. "Network Automation 101 - Skills and Technologies". Packet Coders. https://www.packetcoders.io/network-automation-101-skills-and-technologies/ (accedido el 7 de febrero de 2023).

[27] "Los 5 lenguajes de programación más fáciles de aprender - campusMVP.es". campusMVP.es. https://www.campusmvp.es/recursos/post/los-5-lenguajes-de-programacion-mas-faciles-de-aprender.aspx (accedido el 7 de febrero de 2023).

ANEXOS

Anexo 1. Código para construcción de base de datos

```
import csv
import pandas as pd
import time
import netmiko

LST=["172.16.10.6","172.16.10.10","172.16.10.14","172.16.10.18",
"172.16.10.22","172.16.10.26","172.16.10.30","172.16.10.34","172.16.10.38","
172.16.10.42"]

inicio=time.time()      #Contador para time compilacion
List_Dct=[]             #Lista de dic con info de cada equipo
for ip in LST:
    t_inicio=time.time()
    #SSH session parameters CISCO
    cisco_router = {
        'device_type': 'cisco_ios',
        'host': ip,
        'username': 'admin',
        'password': 'test',
        'port': 22,
    }
    try:
        net_connect = netmiko.ConnectHandler(**cisco_router)
        router="cisco"
    except:

    #SSH session parameters MIKROTIK
        mk_router = {
            'device_type': 'mikrotik_routeros',
            'host': ip,
            'username': 'admin',
            'password': 'test',
            'port': 22,
        }
        net_connect = netmiko.ConnectHandler(**mk_router)
        router="mikrotik"

    print(router)
    ##################
    #--SCRIPT_CISCO--#
```

```
##################

if router=="cisco":

    #Comandos para obtener inf
    C_version  ='show version'
    C_ospf     ='show run | section ospf'
    C_hostname ='show running-config | include hostname'
    C_Lo10     ='show ip int brief | i Loopback10'

    #Enviamos los comandos al equipo CISCO
    GET_IOS_MODEL= net_connect.send_command(C_version)
    comando_ospf = net_connect.send_command(C_ospf)
    GET_HOSTNAME = net_connect.send_command(C_hostname)
    GET_int_Lo10 = net_connect.send_command(C_Lo10)

    #Creando archivos txt con info comandos

    # Archivo txt con info para obtener version y modelo
    file = open("GET_IOS_MODEL.txt","w")
    file.write(GET_IOS_MODEL)
    file.close()

    # Archivo txt con info de BGP summary
    file = open("fileName1.txt","w")
    file.write(comando_ospf)
    file.close()

    # Archivo con info de "show ip interface brief"

    file = open("fileName2.txt","w")
    file.write(GET_int_Lo10)
    file.close()

    ###################---IOS----##############################

    f = open("GET_IOS_MODEL.txt", "r")
    lines = f.readlines()
    linea = lines[0].rstrip()
    f.close()

    #Discriminando IOS

    if "Cisco IOS Software" in linea:
        IOS="cisco_ios"
```

```
        elif "Cisco IOS XE Software" in linea:
            IOS="cisco_xe"
        elif "Cisco Nexus" in linea:
            IOS="cisco_nxos"
        else:
            IOS="ESTE EQUIPO NO ES CISCO!!!"

        #Discriminando MODELO

        lines=lines[20:]
        MODEL=" "
        for line in lines:
            if "3640" in line:
                MODEL="3640"
            elif "3725" in line:
                MODEL="3725"
            elif "3745" in line:
                MODEL="3745"
            elif "7200" in line:
                MODEL="7200"
            elif "CSR1000V" in line:
                MODEL="CSR1000V"
            elif "IOSv" in line:
                MODEL="IOSv"
        if MODEL==" ":
            MODEL="NO MODEL"

        # ####################---ROUTING----############################

        #Leo la 1era linea y le quito el salto de linea
        f = open("fileName1.txt", "r")
        lines = f.readline()
        linea = lines.rstrip()
        f.close()

        if "ospf" in linea:
            routing = "ospf"
        else:

            routing = "static"

        # ###################-LOOPBACK10-##########################

        ns = " ".join( GET_int_Lo10.split() ) #Quita los espacios demas
        list_ns=ns.split(" ")              #Crea una lista
```

```
        Lo10=list_ns[1]

        # ###################---HOSTNAME----######################

        GET_HOSTNAME=GET_HOSTNAME[9:]

        print("###################################")
        print("IP:      "  ,  ip)
        print("HOSTNAME:" ,  GET_HOSTNAME)
        print("###################################")
        print("IOS:     "  ,  IOS)
        print("ROUTING: " , routing)
        print("MODEL:   "  ,  MODEL)
        print("Loopback: ",  Lo10)

        Diccionario_info = {"IP":ip,
                            "HOSTNAME":GET_HOSTNAME,
                            "IOS":IOS,
                            "MODEL":MODEL,
                            "ROUTING":routing,
                            "Loopback10":Lo10
        }
        List_Dct.append(Diccionario_info)

        net_connect.disconnect()
    ######################
    #--SCRIPT_MIKROTIK--#
    ######################

    else:
        #Comandos para show
        routerboard_export= net_connect.send_command('system routerboard
export') #Para obtener MODEL
        comando_ospf      = net_connect.send_command('routing bgp instance
print')#Para obtener AS
        command_hostname  = net_connect.send_command('system identity
print')     #Para obtener hostname
        routing_table     = net_connect.send_command('ip route
print')            #Para obtener static route
        ip_address        = net_connect.send_command('ip address
export')         #Para obtener Lo10

        #Creando archivo con info para MODEL
        file = open("f0.txt","w")      #Creando txt
        file.write(routerboard_export) #Escribiendo comando en txt
```

```
        file.close()

        #Creando archivo con info para AS
        file = open("f1.txt","w")
        file.write(comando_ospf)
        file.close()

        #Se crea un archivo que contiene la informacion del comando
        file = open("f6.txt","w")
        file.write(routing_table)
        file.close()

        #Creando archivo con info para INTERFACE_ETHER
        file = open("f4.txt","w")      #Creando txt
        file.write(ip_address) #Escribiendo comando en txt
        file.close()

        ##########################################################
        #-----------------MODEL_MIKROTIK--------------------#
        ##########################################################

        file = open("f0.txt", "r")     #Leyendo txt

        linea_os = file.readline()
        linea_os = linea_os.rstrip()
        linea_os = linea_os.split()
        IOS=linea_os[-2]+" "+linea_os[-1]

        lines = file.readlines()       #Lista de lineas

        #print(lines)
        MODEL="RouterOSv"
        for line in lines:
            line=line.strip()          #Elimina salto de linea
            if "model" in line:        #Linea con modelo de equipo
                line=line.split(' ')   #Lista de linea
                MODEL=line[-1]         #Obteniendo modelo

        ##########################################################
        #-----------------VALIDANDO_ESTATICA-----------------#
        ##########################################################

        routing="ospf"
        file = open("f6.txt", "r")     #Leyendo txt
        lines = file.readlines()       #Lista de lineas
```

```
        for line in lines:
            line=line.strip()          #Elimina salto de linea
            if "0 A S" in line:        #Linea con ruta estatica
                print(line, "ESTATICO")
                routing="static"
        # ##################---HOSTNAME----######################

        GET_HOSTNAME=command_hostname[6:]

        ###########################################################
        #--------------------Loopback10----------------------#
        ###########################################################

        file1 = open("f4.txt", "r")     #Leyendo txt
        lines = file1.readlines()       #Lista de lineas
        lines=lines[2:]

        for i in lines:
            if "Loopback10" in i:     #Valido R como up
                print(i)
                i=i.strip()            #Elimina salto de linea
                splt=i.split()         #Quita los espacios demas

        ip_add_lo=splt[1]
        Lo10=ip_add_lo[8:]

        print("#################################")
        print("IP:  ",ip)
        print("HOSTNAME",GET_HOSTNAME)
        print("MODEL: ",MODEL)
        print("ROUTING:",routing)

        print("#################################")

        names =
{"IP":ip,"HOSTNAME":GET_HOSTNAME,"IOS":IOS,"MODEL":MODEL,"ROUTING":routing,"
Loopback10":Lo10}
        List_Dct.append(names)

        net_connect.disconnect()

    with open ("Database.csv",mode= "w") as csvfile:
        fieldnames=List_Dct[0].keys()
        writer=csv.DictWriter(csvfile,fieldnames=fieldnames)
        writer.writeheader()
```

```
        writer.writerows(List_Dct)

    #Solo para quitar lineas en blanco
    DATA=pd.read_csv('Database.csv',header=None)
    df = pd.DataFrame(DATA)
    df.to_csv('Database.csv', index=False,header=None)
    df.to_excel('Database.xlsx',index=False,header=None)
    print("####################--------------DATA_BASE-------------
####################")
    print(DATA)
    print("#########################################################################
###############")
    t_fin=time.time()
    print(t_fin-t_inicio,"TIEMPO_COMPILACION_porEquipo")

fin = time.time()
print(fin-inicio,"TIEMPO_COMPILACION")
```

Anexo 2. Código que describe interfaces equipos Cisco

```
import copy
import netmiko
import pandas as pd

# #SHH & COMMANDS

#Creando una lista de todas las ip

IPs=pd.read_csv('IP_csc.csv',header=None)

for index in range(IPs.shape[1]):
    colum_list=IPs.iloc[: , index].values
IP_csco=list(colum_list)
print(IP_csco)

LST=IP_csco

for ip in range(len(LST)):

    cisco_router = {
            'device_type': 'cisco_ios',
            'host': LST[ip],
            'username': 'admin',
            'password': 'test',
            'port': 22,
```

```
    }

net_connect = netmiko.ConnectHandler(**cisco_router)

IDENTIFICADOR_CLIENTE="GYQ273"+str(ip)+"CSCO"

#Comandos
comando_IOS = net_connect.send_command('show version')
interfaces  = net_connect.send_command('show interface description')
ip_int      = net_connect.send_command('show ip int brief')

# Se crea un archivo que contiene info del comando
file = open("fileName2.txt","w")
file.write(comando_IOS)
file.close()

#Creando un archivo txt con la info del comando
file = open("fileName3.txt","w")
file.write(interfaces)
file.close()

#Creando un archivo txt con la info del comando
file = open("fileName4.txt","w")
file.write(ip_int)
file.close()

print(interfaces)
##############################----------##############################
#------------------------------IOS-----------------------------#
##############################----------##############################

#Leo la 1era linea y le quito el salto de linea

f = open("fileName2.txt", "r")
lines = f.readlines()
linea = lines[0].rstrip()
f.close()

#Discriminando IOS

if "Cisco IOS Software" in linea:
    IOS="cisco_ios"
elif "Cisco IOS XE Software" in linea:
    IOS="cisco_xe"
else:
```

```
        IOS="ESTE EQUIPO NO ES CISCO!!!"

    #Discriminando MODELO

    lines=lines[30:]
    MODEL=" "
    for line in lines:
        if "ASR-920-12SZ-A" in line:
            MODEL="ASR-920-12SZ-A"
        elif "ASR-920-4SZ-A" in line:
            MODEL="ASR-920-4SZ-A"
        elif "WS-C4948E-F" in line:
            MODEL="WS-C4948E-F"
        elif "WS-C4948E" in line:
            MODEL="WS-C4948E"
        elif "C1111-4P" in line:
            MODEL="C1111-4P"
    if MODEL==" ":
        MODEL="NO MODEL"

    ##############################----------##############################
    #-------------------------DESCRIPCIONES-----------------------#
    ##############################----------##############################

    #Creando una Lista con todas las int down

    f = open("fileName3.txt", "r")        #Leyendo el archivo
    lineas = f.readlines()                #Cada linea una lista

    List_UNUSED=[]                        #Lista down
    List_USED=[]                          #Lista up
    List_comment=[]                       #Lista de comentarios int up
    List_comment_down=[]                       #Lista de comentarios int up
    for i in lineas:
        i=i.strip()                       #Elimina salto de linea
        count_down=i.count("down")        #Contando int down
        if count_down >= 2:               #Si es mayor igual 2
            ns = " ".join( i.split() )     #Quita los espacios demas
            list_ns=ns.split(" ")          #Crea una lista
            List_UNUSED.append(list_ns[0]) #añadiendo Lista int down
            if len(list_ns)>=4:            #Si mayor 4 append comentario
                    List_comment_down.append(list_ns[3])
            else:
                    List_comment_down.append(" ") #Caso contrario append " "
        else:
```

```
            ns = " ".join( i.split() )
            list_ns=ns.split(" ")
            List_USED.append(list_ns[0])
            if len(list_ns)>=4:          #Si mayor 4 append comentario
                List_comment.append(list_ns[3])
            else:
                List_comment.append(" ") #Caso contrario append " "

    for i in range(len(List_UNUSED)):     #Borrando int Vi
        if "Vi" in List_UNUSED[i]:
            del List_UNUSED[i]
            del List_comment_down[i]

    print(List_comment_down,"LSIT_COMMENT_DOWN")
    print(List_UNUSED,"list_UNUSED")

    #Creando lista solo int up , delete Lo
    ALL_UP=List_USED
    List_USED=List_USED[1:]             #delete 1er item word "Interface"
    List_comment=List_comment[1:]
    List_up=[]
    List_index=[]
    for j in range(len(List_USED)):     #Borrando Loopbacks de lista_USED
        if not "Lo" in List_USED[j]:
            List_up.append(List_USED[j])
            List_index.append(j)

    List_up_comment=[]
    for k in List_index:
        List_up_comment.append(List_comment[k])

    #print(List_index)
    print(List_up,"List_up")
    print(List_up_comment,"List_up_comment")

    ########################
    #----IP_INTERFACES----#
    ########################

    file = open("fileName4.txt","r")

    lineas = file.readlines()              #Cada linea una lista
    List_UNUSED_int=[]                     #Lista ip down
    List_ip_UNUSED =[]
    List_USED_int  =[]                     #Lista int up
```

```
List_ip_USED   =[]
for i in lineas:
    i=i.strip()                      #Elimina salto de linea
    count_down=i.count("down")       #Contando int down
    if count_down >= 2:              #Si es mayor igual 2
        ns = " ".join( i.split() )     #Quita los espacios demas
        list_ns=ns.split(" ")          #Crea una lista
        List_UNUSED_int.append(list_ns[0]) #añadiendo Lista int down
        List_ip_UNUSED.append(list_ns[1]) #
    else:
        ns = " ".join( i.split() )
        list_ns=ns.split(" ")
        List_USED_int.append(list_ns[0])
        List_ip_USED.append(list_ns[1])

print(List_UNUSED_int,"PRINT_LIST_INT")
print(List_ip_UNUSED,"PRINT_LIST_IP_UNUSED")
print(ALL_UP,"PRINT_LIST_ALLUP")
print(List_USED_int,"PRINT_LIST_UP")
print(List_ip_USED,"PRINT_LIST_IP_UP")


################
#----INACTIVA----#
################

#Si la interfaz esta down pero detecta que es de WAN no
#cambia por INACTIVA conserva la descripcion

index_down=[]
List_UNUSED_index=[]
for l in range(len(List_comment_down)):
    if  "WAN" in List_comment_down[l] :
        index_down.append(l)
        #print(l,"INDEX")
    elif  "Ro0" in List_comment_down[l] :
        index_down.append(l)
        #print(l,"INDEX")
    elif  "To_R" in List_comment_down[l] :
        index_down.append(l)
        #print(l,"INDEX")
    elif  "To_" in List_comment_down[l] :
        index_down.append(l)
        #print(l,"INDEX")
    elif  "TO_G" in List_comment_down[l] :
```

```
            index_down.append(l)
            #print(l,"INDEX")
        else:
            List_UNUSED_index.append(l)

    #Creando un archivo txt con todos los comandos a enviarse para down

    file = open("send_des.txt","w+") #Creando el script que se enviara
    file.write("configure terminal"+"\n")

    for i in List_UNUSED_index:
        int_down="interface"+" "+List_UNUSED[i]
        des_int="description INACTIVA"
        sh="shutdown"
        exit_next="exit"
        file.write(int_down+"\n")
        file.write(des_int+"\n")
        file.write(sh+"\n")
        file.write(exit_next+"\n")

    ################
    #-----LO10------#
    ################


    file.write("interface Lo10"+"\n")
    file.write("description INTL:GESTION-DES:ACCESO_CPE"+"\n")
    file.write("exit"+"\n")


    ################
    #--LAN_CLIENTE--#
    ################

    index_LAN=[]
    for l in range(len(List_up_comment)):  #En la lista de all up busco UNI
        if " " in List_up_comment[l] :
            index_LAN.append(l)
            #print(l,"INDEX")
        elif "LAN" in List_up_comment[l] :
            index_LAN.append(l)
            #print(l,"INDEX")
        elif "DATOS" in List_up_comment[l] :
            index_LAN.append(l)
            #print(l,"INDEX")
```

```
        elif "Datos" in List_up_comment[l] :
            index_LAN.append(l)
            #print(l,"INDEX")
        elif "CLIENTE" in List_up_comment[l] :
            index_LAN.append(l)
            #print(l,"INDEX")
    print(index_LAN,"index UNI")


    LAN_description="RIC-IDC:"+IDENTIFICADOR_CLIENTE+"-DES:"

    #RIC= RED INTERNA CLIENTE
    #IDC= IDENTIFICADOR CLIENTE
    #DES= DESCRIPCION

    index_List_LAN=[]
    List_LAN=[]
    for u in index_LAN:
        if "Fa" in List_up[u]:
            List_LAN.append(List_up[u])
            index_List_LAN.append(u)
            print(List_up[u],"EUREKA")
        if "Gi0/0/" in List_up[u]:
            List_LAN.append(List_up[u])
            index_List_LAN.append(u)
            print(List_up[u],"EUREKA")
        elif "Gi1/" in List_up[u]:
            List_LAN.append(List_up[u])
            index_List_LAN.append(u)
            print(List_up[u],"EUREKA")
        elif "Gi" in List_up[u]:
            List_LAN.append(List_up[u])
            index_List_LAN.append(u)
            print(List_up[u],"EUREKA")

    print(List_LAN,"List_UNI")
    print(index_List_LAN,"index_List_UNI")


    for o in index_List_LAN:

            int_UNI="interface"+" "+List_up[o]
            if List_up_comment[o]==" ":
                List_up_comment[o]="LAN_CLIENTE"
```

```
                des_int="description"+" "+
LAN_description+List_up_comment[o]
                print(int_UNI,"int_UNI")
                exit_next="exit"
                file.write(int_UNI+"\n")
                file.write(des_int+"\n")
                file.write(exit_next+"\n")

            else :
                des_int="description"+" "+
LAN_description+List_up_comment[o]
                print(int_UNI,"int_UNI")
                exit_next="exit"
                file.write(int_UNI+"\n")
                file.write(des_int+"\n")
                file.write(exit_next+"\n")

    ################
    #-----WAN------#
    ################

    index_WAN=[]
    for l in range(len(List_up_comment)):  #En la lista de all up busco WAN
        if "WAN" in List_up_comment[l] :
            index_WAN.append(l)
        elif "TO_" in List_up_comment[l] :
            index_WAN.append(l)
        elif "GYQ" in List_up_comment[l] :
            index_WAN.append(l)

            #print(l,"INDEX")
    print(index_WAN,"index UNI")


    WAN_description="REC-INTF:LY3-ID:"+IDENTIFICADOR_CLIENTE+"-DES:"

    index_List_WAN=[]
    List_WAN=[]
    for u in index_WAN:
        if "Fa" in List_up[u]:
            List_WAN.append(List_up[u])
            index_List_WAN.append(u)
            print(List_up[u],"EUREKA")
        if "Gi0/0/" in List_up[u]:
            List_WAN.append(List_up[u])
```

```
            index_List_WAN.append(u)
            print(List_up[u],"EUREKA")
        elif "Te1/" in List_up[u]:
            List_WAN.append(List_up[u])
            index_List_WAN.append(u)
            print(List_up[u],"EUREKA")
        elif "Gi1/" in List_up[u]:
            List_WAN.append(List_up[u])
            index_List_WAN.append(u)
            print(List_up[u],"EUREKA")
        elif "Gi" in List_up[u]:
            List_WAN.append(List_up[u])
            index_List_WAN.append(u)
            print(List_up[u],"EUREKA")

    print(List_WAN,"List_WAN")
    print(index_List_WAN,"index_List_WAN")



    for o in index_List_WAN:

            int_UNI="interface"+" "+List_up[o]
            if List_up_comment[o]==" ":
                List_up_comment[o]="WAN"
                des_int="description"+" "+
WAN_description+List_up_comment[o]
                print(int_UNI,"int_UNI")
                exit_next="exit"
                file.write(int_UNI+"\n")
                file.write(des_int+"\n")
                file.write(exit_next+"\n")

            else :
                des_int="description"+" "+
WAN_description+List_up_comment[o]
                print(int_UNI,"int_UNI")
                exit_next="exit"
                file.write(int_UNI+"\n")
                file.write(des_int+"\n")
                file.write(exit_next+"\n")


    #Creando un archivo txt con todos los comandos a enviarse para up
```

```
    file = open("send_des_up.txt","w+") #Creando el script que se enviara
    file.write("configure terminal"+"\n")

    for i in range(len(List_up)):
        int_down="interface"+" "+List_up[i]
        des_int="description"+" "+List_up_comment[i]
        exit_next="exit"
        file.write(int_down+"\n")
        file.write(des_int+"\n")
        file.write(exit_next+"\n")

    #Enviando descripciones

    send_description=net_connect.send_config_from_file('send_des.txt',read_t
imeout=90)

    interfaces  = net_connect.send_command('show interface description')
    print(interfaces)
    print("##########################################################################
###########")

    net_connect.disconnect()
```

Anexo 3. Código que describe interfaces equipos Mikrotik

```
import netmiko
import time
import pandas as pd
#SSH

# #SHH & COMMANDS

#Creando una lista de todas las ip

IPs=pd.read_csv('IP_mk.csv',header=None)

for index in range(IPs.shape[1]):
    colum_list=IPs.iloc[: , index].values
IP_mk=list(colum_list)
print(IP_mk)

LST=IP_mk

#hostname="ECGUEEMEMFRCPUFNMANFRE01"
```

```
for ip in range(len(LST)):

    mk_router = {
                'device_type': 'mikrotik_routeros',
                'host': LST[ip],
                'username': 'admin',
                'password': 'test',
                'port': 22,
            }
    net_connect = netmiko.ConnectHandler(**mk_router)

    IDENTIFICADOR_CLIENTE="GYQ273"+str(ip)+"MK"

    #COMANDOS_PRINT

    routerboard_export= net_connect.send_command('system routerboard
export')      #Para obtener model

    #Creando archivo con info para MODEL
    file = open("f0.txt","w")      #Creando txt
    file.write(routerboard_export) #Escribiendo comando en txt
    file.close()

    ############################################################
    #-----------------MODEL_MIKROTIK--------------------#
    ############################################################

    file = open("f0.txt", "r")     #Leyendo txt
    lines = file.readlines()       #Lista de lineas
    for line in lines:
        line=line.strip()          #Elimina salto de linea
        if "model" in line:        #Linea con modelo de equipo
            line=line.split(' ')   #Lista de linea
            model=line[-1]         #Obteniendo modelo

    #COMANDOS

    ACL_export        = net_connect.send_command('ip firewall address-list
export')#Para obtener ACL
    ACL_print         = net_connect.send_command('ip firewall address-list
print') #Para obtener ACL
    interace_ethernet = net_connect.send_command('interface ethernet
export')      #Para obtener interfaces
    int_eth_print     = net_connect.send_command('interface ethernet
print')       #Para obtener interfaces
```

```
route_print       = net_connect.send_command('ip route print')


#Creando archivo con info para ACL
file = open("f1.txt","w")      #Creando txt
file.write(ACL_export) #Escribiendo comando en txt
file.close()

#Creando archivo con info para ACL
file = open("f2.txt","w")      #Creando txt
file.write(ACL_print) #Escribiendo comando en txt
file.close()

#Creando archivo con info para INTERFACE_ETHER
file = open("f3.txt","w")      #Creando txt
file.write(interace_ethernet) #Escribiendo comando en txt
file.close()

#Creando archivo con info para INTERFACE_ETHER
file = open("f4.txt","w")      #Creando txt
file.write(int_eth_print) #Escribiendo comando en txt
file.close()

#Creando archivo con info para ROUTE PRINT
file = open("f5.txt","w")      #Creando txt
file.write(route_print) #Escribiendo comando en txt
file.close()



##########################################################
#------------------INTERFACE_MK----------------------#
##########################################################

file1 = open("f4.txt", "r")     #Leyendo txt
lines = file1.readlines()       #Lista de lineas
lines=lines[2:]
#print(lines[8])

ether_list=[]
List_up=[]
List_dow=[]
List_up_comment=[]
for i in lines:
    i=i.strip()            #Elimina salto de linea
```

```
        splt=i.split()          #Quita los espacios demas
        print(splt)
        if "R" in splt[1]:      #Valido R como up
            List_up.append(splt[0])
            List_up_comment.append(splt[3])
        else:                   #Los demas estan down
            List_dow.append(splt[0])

    List_del_dow=[]
    for i in List_dow:
        #print(i)
        if len(i)>2:
            List_del_dow.append(i)

    for k in List_del_dow:
        List_dow.remove(k)


    print(List_up,"up")
    print(List_dow,"dow")

    ######---UNUSED---######

    file = open("send_des_Mk.txt","w+") #Creando el script que se enviara

    List_UNUSED=[]
    for ether in List_dow:
        if (ether != "0") and (ether!= "7"):
            List_UNUSED.append(ether)
    print(List_UNUSED,"UNUSED")

    for i in List_UNUSED:
        int_down="interface ethernet"+" "+"comment"+" "+i+" "+"INACTIVA"
        file.write(int_down+"\n")
        #int_disable="interface ethernet disable"+" "+i
        #file.write(int_disable+"\n")

    ######---UNI---######

    List_UNI=[]
    for ether in List_up:
        if (ether != "0") and (ether!= "7"):
            List_UNI.append(ether)
    print(List_UNI,"UNI")
```

```
    for i in List_UNI:
        UNI_description=("interface ethernet"+" "+"comment"+" "+i
                         +" "+"RIC-IDC:"+IDENTIFICADOR_CLIENTE+"-
DE:LAN_CLIENTE")
        file.write(UNI_description+"\n")

    ######---IDU---######

    IDU_description=("interface bridge set
comment="+"IDU:"+IDENTIFICADOR_CLIENTE+
                     "-DE:LAN_CLIENTE"+" "+"bridge1")
    file.write(IDU_description+"\n")

    ###--Loopback593--####

    Lo_description=("interface bridge set comment="
                     +"INTL:GESTION-DES:ACCESO_CPE"+" "+"Loopback10")

    file.write(Lo_description+"\n")


    ################
    #-----WAN------#
    ################

    print(List_up)
    print(List_up_comment)


    index_WAN=[]
    for l in range(len(List_up_comment)):  #En la lista de all up busco WAN
        if "WAN" in List_up_comment[l] :
            index_WAN.append(l)
        elif "TO_G" in List_up_comment[l] :
            index_WAN.append(l)
        elif "GYQ" in List_up_comment[l] :
            index_WAN.append(l)

            #print(l,"INDEX")


    WAN_des="REC-INTF:LY3-IDC:"+IDENTIFICADOR_CLIENTE+"-DES:"

    for o in index_WAN:

        WAN_description=("interface ethernet"+" "+"comment"+" "+List_up[o]
```

```
                        +" "+WAN_des+"-DE:"+List_up_comment[o])
        file.write(WAN_description+"\n")


    file.close()

    print(index_WAN,"index WAN")


    #-----------------------#
    #--SEND_COMMANDS--#
    #-----------------------#

    #Interfaces description
    send=net_connect.send_config_from_file('send_des_Mk.txt',read_timeout=90
)


    net_connect.disconnect()
```

Anexo 4. Código para configuraciones adicionales

```
import copy
import netmiko
import pandas as pd
import time

# #SHH & COMMANDS

t_inicio=time.time()
#Creando una lista de todas las ip

IPs=pd.read_csv('IP_csc.csv',header=None)

for index in range(IPs.shape[1]):
    colum_list=IPs.iloc[: , index].values
IP_csco=list(colum_list)
print(IP_csco)

LST=IP_csco

for ip in range(len(LST)):

    inicio=time.time()
```

```
    IDENTIFICADOR_CLIENTE="GYQ273"+str(ip)+"CSCO"
    cisco_router = {
            'device_type': 'cisco_ios',
            'host': LST[ip],
            'username': 'admin',
            'password': 'test',
            'port': 22,
        }

    net_connect = netmiko.ConnectHandler(**cisco_router)

    #-----------------------#
    #------CONFIG_Lo20------#
    #-----------------------#

    i=str(ip+1)
    Lo20="10.20.20."+i

    file = open("config_Lo20.txt","w+") #Creando el script que se enviara

    file.write("Configure terminal")
    file.write("\n")
    file.write("interface Loopback20")
    file.write("\n")
    file.write("ip address"+" "+Lo20+" "+"255.255.255.255")
    file.write("\n")
    file.write("end")
    file.write("\n")

    #-----------------------#
    #---------ACL-----------#
    #-----------------------#

    file.write("Configure terminal")
    file.write("\n")
    file.write("ip access-list extended ssh-access")
    file.write("\n")
    file.write("permit ip 172.16.10.0 0.0.0.255 any")
    file.write("\n")
    file.write("permit ip 162.16.10.0 0.0.0.255 any")
    file.write("\n")
    file.write("permit ip 152.16.10.0 0.0.0.255 any")
    file.write("\n")
    file.write("permit ip 192.168.88.0 0.0.0.255 any")
    file.write("\n")
```

```
    file.write("permit ip host 10.10.10.1 any")
    file.write("\n")
    file.write("end")
    file.write("\n")
    file.write("Configure terminal")
    file.write("\n")
    file.write("line vty 0 4")
    file.write("\n")
    file.write("access-class ssh-access in")
    file.write("\n")
    file.write("end")
    file.write("\n")
    file.write("Configure terminal")
    file.write("\n")

    #-----------------------#
    #--------HOSTNAME-------#
    #-----------------------#

    file.write("hostname"+" "+IDENTIFICADOR_CLIENTE)
    file.write("\n")
    file.write("end")
    file.write("\n")
    file.close()

    send_description=net_connect.send_config_from_file('config_Lo20.txt',rea
d_timeout=90)

    interfaces  = net_connect.send_command('show ip int brief')
    print(interfaces)
    print("##########################################################################
############")

    net_connect.disconnect()
    fin = time.time()
    print(fin-inicio,"TIEMPO_COMPILACION_INDIVIUAL")

t_fin = time.time()
print(t_fin-t_inicio,"TIEMPO_COMPILACION_TOTAL")
```