

MANUAL TÉCNICO

MPLStudio

MANUAL TÉCNICO

Contenido

OVERVIEW.....	4
INTRODUCCIÓN	4
OBJETIVO Y ALCANCES DEL SISTEMA.....	4
NORMAS, POLÍTICAS Y PROCEDIMIENTOS	4
RECURSOS DE SOFTWARE.....	5
PYTHON.....	5
PARAMIKO	5
PYQT	5
GNS3	6
RECURSOS DE SIMULACIÓN.....	6
RECURSOS DE HARDWARE	6
SWITCH CATALYST 2960.....	6
ENRUTADOR CISCO 2811	6
CASO DE ESTUDIO	7
DIAGRAMA Y DIRECCIONAMIENTO DE LA RED	7
CONFIGURACIONES PROPUESTAS.....	8
Configuración de direccionamiento entre dispositivos PE.....	8
Configuración de un IGP (Interior Gateway Protocol) en los PE1 y PE2	9
Comunicación entre PE's	9
Intercambio de clientes entre PE1 y PE2	10
Disminución de carga de procesamiento utilizando el protocolo CEF	10
Creación de VRFs y asociación de sub-interfaces en los PE	11
Configuración de la comunicación entre PE y CE.....	11
Redistribución de rutas del aprendidas por el CE	12
Configuración de direccionamiento en Provider	12
Configuración de un IGP (Interior Gateway Protocol) para la comunicación entre PE1 y PE2	13
Disminución de carga de procesamiento utilizando el protocolo CEF	13
Configuración de los Customer Edge (Gye_CFR y Cue_AlmacenesJR)	14
CÓDIGO FUENTE	15
LOGIN.PY	15

REMOTE.PY	16
SELECTORLOCAL.PY	18
CONFIGURACIONBASICA.PY	18
DIRECCIONAMIENTOLOCAL.PY	19
SELECTORCREDENCIALES.PY	21
ANADIRUSUARIO.PY	22
PLANTILLA.PY	22
SELECTORCONFIGURACIONMPLS.PY	23
CONFIGURARCE.PY	24
CONFIGURARPE2.PY	25
FUNCIONES2.PY	28
DATABASE.PY	39
SOPORTE	40

Overview

Introducción

MPLStudio es un aplicativo de escritorio desarrollado para Windows que permite realizar conexiones remotas por medio del protocolo *SSH*. Con este aplicativo se puede automatizar la configuración de dispositivos intermedios del backbone de una red MPLS-VPN-L3 de un ISP. Dentro de esta configuración se incluyen aspectos como enrutamiento dinámico, redistribución de rutas, etiquetado de paquetes bajo *LDP*, configuración de rutas estáticas y creación de *VRFs* para clientes del ISP.

Además, se incluyen un apartado que permite la configuración de direccionamiento de las interfaces disponibles en el dispositivo (enrutador), y la opción de configurar parámetros básicos del equipo como nombres de dominio, dns, hostname y credenciales de usuarios.

Objetivo y alcances del sistema

- Automatizar la configuración de dispositivos intermediarios en una red de datos.
- Mejorar la interacción hombre-máquina mediante interfaces gráficas amigables.
- Facilitar el trabajo del administrador de redes mediante ingreso de parámetros de la red por teclado.

Normas, políticas y procedimientos

El ingreso a la aplicación está controlado por un login inicial el cual se valida con las credenciales de usuarios previamente autorizados y registrados en la base de datos. Para más información ver

Se espera que en todo momento la conexión hacia el dispositivo a configurar sea estable, por lo cual se recomienda cerrar cualquier otro tipo de aplicación que realice conexiones de este tipo durante el uso del aplicativo.

Todos los campos de ingreso de datos dentro de la aplicación están debidamente validados, sin embargo, MPLStudio no verifica que la información que el usuario ingrese es correcta, es decir las fallas humanas de tipo diseño no están validadas.

Recursos de software

Dentro de esta sección se describen las librerías y frameworks de terceros usados durante la implementación de MPLStudio.

Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma. Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Paramiko

Paramiko es una implementación de Python (2.7, 3.4+) del protocolo SSHv2, que proporciona funcionalidad de cliente y servidor. Mientras que aprovecha una extensión de Python C para la criptografía de bajo nivel (criptografía), Paramiko en sí es una interfaz pura de Python alrededor de los conceptos de redes SSH.

PyQT

PyQt es uno de los dos productos de Python más populares para el Qt cross-platform GUI/XML/SQL C++. PyQt desarrollado por Riverbank Computing Limited. PyQt contiene más de 620 clases que cubren interfaces gráficas de usuario, manejo de XML, comunicación de red, bases de datos SQL, navegación web y otras tecnologías disponibles en Qt.

Esta librería junto con su Framework *PyQt Designer* fueron usadas para la creación y diseño de ventanas e interfaces graficas del aplicativo.

GNS3

GNS3 es un simulador gráfico de red que te permite diseñar topologías de red complejas y poner en marcha simulaciones sobre ellos.

Para permitir completar simulaciones, GNS3 está estrechamente vinculada con:

- Dynamips, un emulador de IOS que permite a los usuarios ejecutar imágenes binarias IOS de Cisco System.
- Dynagen, un front-end basado en texto para Dynamips
- Qemu y VirtualBox, para permitir utilizar máquinas virtuales como un firewall PIX.
- VPCS, un emulador de PC con funciones básicas de networking
- IOU (IOS on Unix), compilaciones especiales de IOS provistas por Cisco para correr directamente en sistemas UNIX y derivados.

GNS3 es una excelente herramienta complementaria a los verdaderos laboratorios para los administradores de redes de Cisco o las personas que quieren pasar sus CCNA, CCNP, CCIE DAC o certificaciones.

Recursos de Simulación

Las pruebas realizadas sobre MPLStudio en su etapa de desarrollo fueron llevadas a cabo bajo el programa **GNS3**, el cual permite la simulación de sistemas de redes creando interfaces virtuales lo cual facilita al programador a testear configuraciones reales en un ambiente controlado.

El IOS usado para simular los enrutadores corresponde a [c7200-adventerprisek9-mz.153-3.XB12](#) el cual representa a los enrutadores 7200 de Cisco. Es importante resaltar que en esta versión de IOS los enrutadores no cuentan con *Modo Usuario* por lo cual para es necesario tomar correctivos en el código al momento de usar otra versión del IOS.

Recursos de Hardware

Switch Catalyst 2960

Soportan voz, video, datos y acceso seguro. Disponen de 24 puertos 10/100 más dos puertos SFP. Capacidad de configurar LAN virtuales, función Power over Ethernet que permite implementar fácilmente nuevas funciones como comunicaciones por voz e inalámbricas sin necesidad de realizar nuevas conexiones. Seguridad integrada.

Enrutador cisco 2811

Memoria RAM de 256 MB instalados, soportando hasta 768 MB, memoria FLASH de 64 MB instalados soportando hasta 256 MB. Utiliza IPSec como protocolo de transporte, protección firewall, cifrado del hardware,

soporte de MPLS, diseño modular, criptografía de 128 bits, asistencia técnica VPN, filtrado de URL, cifrado de 256 bits y cumple con las normas de la IEEE 802.3af.

Caso de estudio

Diagrama y Direcccionamiento de la Red

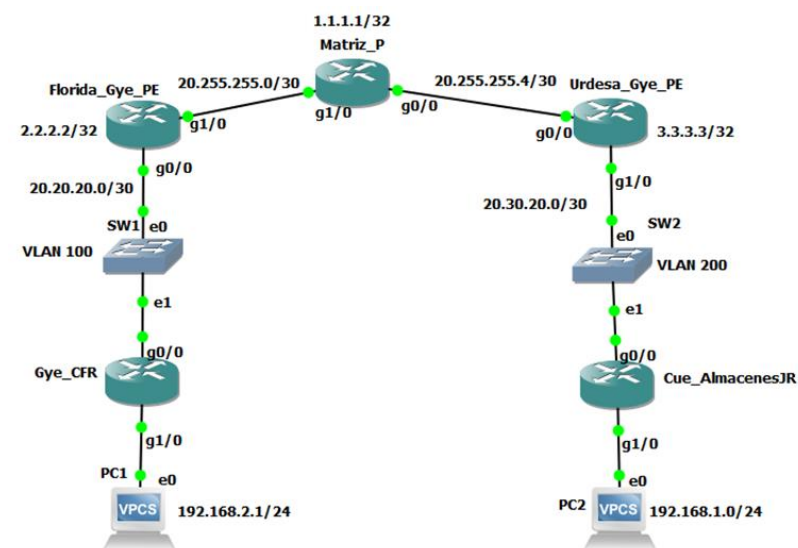


Ilustración 1

Bajo el diagrama de la red propuesto en la *Ilustración 1* se realizaron las configuraciones en dispositivos simulados para probar el correcto funcionamiento de la aplicación, obteniendo resultados favorables y replicables en cualquier otro escenario. En la *Ilustración 2* se incluye el direccionamiento en ipv4 proporcionado a cada dispositivo.

Dispositivo	Dirección red/CIDR	de Interfaz	Dirección IP	Máscara subred	de Puerta de enlace predeterminada
Matriz_P	20.255.255.4/30	G0/0	20.255.255.5	255.255.255.252	N/A
	20.255.255.0/30	G0/1	20.255.255.1	255.255.255.252	N/A
	1.1.1.0/32	Lo0	1.1.1.1	255.255.255.255	N/A
Florida_Gye_PE	20.20.20.0/30	G0/0.100	20.20.20.1	255.255.255.252	N/A
	20.255.255.0/30	G0/1	20.255.255.2	255.255.255.252	N/A
	2.2.2.0/32	Lo0	2.2.2.2	255.255.255.255	N/A
Urdesa_Gye_PE2	20.255.255.0/30	G0/0	20.255.255.6	255.255.255.252	N/A
	20.30.20.0/30	G0/1.200	20.30.20.1	255.255.255.252	N/A
	3.3.3.0/32	Lo0	3.3.3.3	255.255.255.255	N/A
Gye_CFR	20.20.20.0/30	G0/0	20.20.20.2	255.255.255.252	N/A
Cue_AlmacenesJR	20.30.20.0/30	G0/0	20.20.20.2	255.255.255.252	N/A

Ilustración 2

Configuraciones propuestas

Este apartado muestra las configuraciones propuestas para los dispositivos que toman parte en la red MPLS-VPN L3.

Configuración de direccionamiento entre dispositivos PE

Nombre de dispositivo: Florida_GYE_PE


Descripción: Se configura el direccionamiento de las interfaces Loopback0 y GigabitEthernet1/0. Adicionalmente se habilita el protocolo mpls en esta última interfaz ya que se encuentra conectada a un dispositivo tipo P.





```
Florida_GYE_PE(config)# interface Loopback0
Florida_GYE_PE (config-if)# ip address 2.2.2.2
255.255.255.255
Florida_GYE_PE (config-if)# interface GigabitEthernet1/0
Florida_GYE_PE (config-if)# description HACIA-P
Florida_GYE_PE (config-if)# ip address 20.255.255.2
255.255.255.252
Florida_GYE_PE (config-if)# duplex full
Florida_GYE_PE (config-if)# mpls ip
Florida_GYE_PE (config-if)# exit
```

Nombre de dispositivo: Urdesa_GYE_PE


Descripción: Se configura el direccionamiento de las interfaces Loopback0 y GigabitEthernet0/0. Adicionalmente se habilita el protocolo mpls en esta última interfaz ya que se encuentra conectada a un dispositivo tipo P.

	<pre> Urdesa_GYE_PE(config)# interface Loopback0 Urdesa_GYE_PE (config-if)# ip address 3.3.3.3 255.255.255.255 Urdesa_GYE_PE (config-if)# interface GigabitEthernet0/0 Urdesa_GYE_PE (config-if)# description HACIA-P Urdesa_GYE_PE (config-if)# ip address 20.255.255.6 255.255.255.252 Urdesa_GYE_PE (config-if)# mpls ip Urdesa_GYE_PE (config-if)# exit </pre>
---	--

Configuración de un IGP (interior Gateway Protocol) en los PE1 y PE2

Nombre de dispositivo: Florida_GYE_PE Descripción: Se anuncian las redes 2.2.2.2/32 y la 20.255.255.0/30 por el protocolo de enrutamiento OSPF con AS 1 y en el área de backbone 0.	
	<pre> Florida_GYE_PE (config)# router ospf 1 Florida_GYE_PE (config)# network 2.2.2.2 0.0.0.0 area 0 Florida_GYE_PE (config)# network 20.255.255.0 0.0.0.3 area 0 </pre>
Nombre de dispositivo: Urdesa_GYE_PE Descripción: Se anuncian las redes 3.3.3.3/32 y la 20.255.255.0/30 por el protocolo de enrutamiento OSPF con AS 1 y en el área de backbone 0.	
	<pre> Urdesa_GYE_PE (config)# router ospf 1 Urdesa_GYE_PE (config)# network 3.3.3.3 0.0.0.0 area 0 Urdesa_GYE_PE (config)# network 20.255.255.0 0.0.0.3 area 0 </pre>

Comunicación entre PE's

Nombre de dispositivo: Florida_GYE_PE Descripción: Se configura al router vecino iBGP con la ip de loopback 3.3.3.3 según la tabla de direccionamiento.	
	<pre> Florida_GYE_PE (config)# router bgp 1 Florida_GYE_PE (config-router)# neighbor 3.3.3.3 remote-as 1 Florida_GYE_PE (config-router)# neighbor 3.3.3.3 update- source Loopback0 Florida_GYE_PE (config-router)# neighbor 3.3.3.3 next-hop- self Florida_GYE_PE (config-router)# no auto-summary Florida_GYE_PE (config-router)# exit </pre>

Nombre de dispositivo: Urdesa_GYE_PE

Descripción: Se configura al router vecino iBGP con la ip de loopback 2.2.2.2 según la tabla de direccionamiento.



```
Urdesa_GYE_PE (config)# router bgp 1
Urdesa_GYE_PE (config-router)# neighbor 2.2.2.2 remote-as 1
Urdesa_GYE_PE (config-router)# neighbor 2.2.2.2 update-
source Loopback0
Urdesa_GYE_PE (config-router)# neighbor 2.2.2.2 next-hop-
self
Urdesa_GYE_PE (config-router)# no auto-summary
Urdesa_GYE_PE (config-router)# exit
```

Intercambio de clientes entre PE1 y PE2

Nombre de dispositivo: Florida_GYE_PE

Descripción: Se configura el protocolo MP-BGP, anunciando como vecino al router con la ip de loopback 3.3.3.3 según la tabla de direccionamiento.



```
Florida_GYE_PE (config)# router bgp 1
Florida_GYE_PE (config-router)# address-family vpnv4
Florida_GYE_PE (config-router-af)# neighbor 3.3.3.3 activate
Florida_GYE_PE (config-router-af)# neighbor 3.3.3.3 send-
community extended
Florida_GYE_PE (config-router-af)# exit-address-family
```

Nombre de dispositivo: Urdesa_GYE_PE

Descripción: Se configura el protocolo MP-BGP, anunciando como vecino al router con la ip de loopback 2.2.2.2 según la tabla de direccionamiento.



```
Urdesa_GYE_PE (config)# router bgp 1
Urdesa_GYE_PE (config-router)# address-family vpnv4
Urdesa_GYE_PE (config-router-af)# neighbor 2.2.2.2 activate
Urdesa_GYE_PE (config-router-af)# neighbor 2.2.2.2 send-
community extended
Urdesa_GYE_PE (config-router-af)# exit-address-
family
```

Disminución de carga de procesamiento utilizando el protocolo CEF

Nombre de dispositivo: Florida_GYE_PE

Descripción: Se configura el protocolo CEF, LDP y MPLS de forma global y por interfaces.



```
Florida_GYE_PE (config)# ip cef
Florida_GYE_PE (config)# mpls label protocol ldp
Florida_GYE_PE (config)# mpls ldp router-id Loopback0
Florida_GYE_PE (config)# mpls ip
```

Nombre de dispositivo: Urdesa_GYE_PE

Descripción: Se configura el protocolo CEF, LDP y MPLS de forma global y por interfaces.



```
Urdesa_GYE_PE (config)# ip cef
Urdesa_GYE_PE (config)# mpls label protocol ldp
Urdesa_GYE_PE (config)# mpls ldp router-id Loopback0
Urdesa_GYE_PE (config)# mpls ip
```

Creación de VRFs y asociación de sub-interfaces en los PE

Nombre de dispositivo: Florida_GYE_PE

Descripción: Se crea la VRF CE_CRF con su respectivo router distinguisher, además se establece que un cliente de esta VRF se conectará por la subinterfaz GigabitEthernet0/0.100.



```
Florida_GYE_PE (config)# ip vrf CE_CRF
Florida_GYE_PE (config-vrf)# rd 65535:100
Florida_GYE_PE (config-vrf)# route-target export 100:100
Florida_GYE_PE (config-vrf)# route-target import 100:100
Florida_GYE_PE (config-vrf)# route-target import 200:200
Florida_GYE_PE (config-if)# interface GigabitEthernet0/0
Florida_GYE_PE (config-if)# no shutdown
Florida_GYE_PE (config-if)# interface GigabitEthernet0/0.100
Florida_GYE_PE (config-if)# description HACIA-SW1-CE
Florida_GYE_PE (config-if)# encapsulation dot1Q 100
Florida_GYE_PE (config-if)# ip vrf forwarding CE_CRF
Florida_GYE_PE (config-if)# ip address 20.20.20.1
255.255.255.252
Florida_GYE_PE (config-if)# exit
```

Nombre de dispositivo: Urdesa_GYE_PE

Descripción: Se crea la VRF CE_AlmacenesJr con su respectivo router distinguisher, además se establece que un cliente de esta VRF se conectará por la subinterfaz GigabitEthernet1/0.200.



```
Urdesa_GYE_PE (config)# ip vrf CE_AlmacenesJr
Urdesa_GYE_PE (config-vrf)# rd 65535:100
Urdesa_GYE_PE (config-vrf)# route-target export 200:200
Urdesa_GYE_PE (config-vrf)# route-target import 200:200
Urdesa_GYE_PE (config-vrf)# route-target import 100:100
Urdesa_GYE_PE (config-if)# interface GigabitEthernet1/0
Urdesa_GYE_PE (config-if)# no shutdown
Urdesa_GYE_PE (config-if)# interface GigabitEthernet1/0.200
Urdesa_GYE_PE (config-if)# description HACIA-SW2-CE
Urdesa_GYE_PE (config-if)# encapsulation dot1Q 100
Urdesa_GYE_PE (config-if)# ip vrf forwarding CE_AlmacenesJr
Urdesa_GYE_PE (config-if)# ip address 20.30.20.1
255.255.255.252
Urdesa_GYE_PE (config-if)# exit
```

Configuración de la comunicación entre PE y CE

Nombre de dispositivo: Florida_GYE_PE

Descripción: Se establece una ruta para permitir la salida del tráfico de la red LAN del cliente con la VRF CE_CRF.



```
Florida_GYE_PE (config)# ip route vrf CE_CRF 192.168.2.1
255.255.255.0 20.20.20.2
```

Nombre de dispositivo: Urdesa_GYE_PE

Descripción: Se establece una ruta para permitir la salida del tráfico de la red LAN del cliente con la VRF CE_AlmacenesJr.



```
Urdesa_GYE_PE (config)# ip route vrf CE_AlmacenesJr
192.168.1.1 255.255.255.0 20.30.20.1
```

Redistribución de rutas del aprendidas por el CE

Nombre de dispositivo: Florida_GYE_PE

Descripción: Redistribución de rutas estáticas por la VRF CE_CRF en el protocolo BGP.



```
Florida_GYE_PE (config)# router bgp 1
Florida_GYE_PE (config-router)# address-family ipv4 vrf
CE_CRF
Florida_GYE_PE (config-router-af)# redistribute static
```

Nombre de dispositivo: Urdesa_GYE_PE

Descripción: Redistribución de rutas estáticas por la VRF CE_AlmacenesJr en el protocolo BGP.



```
Urdesa_GYE_PE (config)# router bgp 1
Urdesa_GYE_PE (config-router-af)# address-family ipv4 vrf
CE_AlmacenesJr
Urdesa_GYE_PE (config-router-af)# redistribute static
Urdesa_GYE_PE (config-router-af)# exit-address-family
```

Configuración de direccionamiento en Provider

Nombre de dispositivo: Matriz_P

Descripción: Se configura el direccionamiento en las interfaces Loopback0, GigabitEthernet0/0 y GigabitEthernet0/1. En estas dos últimas interfaces se habilita MPLS ya que están conectadas a dispositivos tipo PE:



```
Matriz_P(config)# interface Loopback0
Matriz_P(config-if)# ip address 1.1.1.1 255.255.255.255
Matriz_P(config-if)# interface GigabitEthernet0/0
Matriz_P(config-if)# description HACIA-Florida_GYE_PE
Matriz_P(config-if)# ip address 20.255.255.5
255.255.255.252
Matriz_P(config-if)# mpls ip
Matriz_P(config-if)# interface GigabitEthernet0/1
Matriz_P(config-if)# description HACIA-Urdesa_GYE_PE
Matriz_P(config-if)# ip address 20.255.255.1
255.255.255.252
Matriz_P(config-if)# mpls ip
```

Configuración de un IGP (Interior Gateway Protocol) para la comunicación entre PE1 y PE2

Nombre de dispositivo: Matriz_P

Descripción: Se anuncian las redes 1.1.1.1/32 y 20.255.255.0/29 por el protocolo de enrutamiento OSPF con AS 1 y en el área de backbone 0.



```
Matriz_P(config)# router ospf 1
Matriz_P(config-router)# network 1.1.1.1 0.0.0.0 area 0
Matriz_P(config-router)# network 20.255.255.0 0.0.0.7
area 0
Matriz_P(config-router)# exit
```

Disminución de carga de procesamiento utilizando el protocolo CEF

Nombre de dispositivo: Matriz_P

Descripción: Se configura el protocolo CEF, MPLS y LDP de forma global y por interfaces.



```
Matriz_P(config)# ip cef
Matriz_P(config)# mpls label protocol ldp
Matriz_P(config)# mpls ldp router-id Loopback0
Matriz_P(config)# mpls ip
```

Configuración de los Customer Edge (Gye_CFR y Cue_AlmacenesJR)

Nombre de dispositivo: Gye_CFR**Descripción:** Se configura el direccionamiento en la interfaz GigabitEthernet0/0 y GigabitEthernet0/1, además de la ruta estática que le permite comunicarse con los dispositivos del ISP.

```
Gye_CFR (config)# interface GigabitEthernet0/0
Gye_CFR (config-if)# description HACIA-SW1-
Florida_Gye_PE
Gye_CFR (config-if)# ip address 20.20.20.2
255.255.255.252
Gye_CFR (config-if)# no shutdown
Gye_CFR (config-if)# interface GigabitEthernet0/1
Gye_CFR (config-if)# description LAN
Gye_CFR (config-if)# ip address 192.168.2.2
255.255.255.0
Gye_CFR (config-if)# no shutdown
Gye_CFR (config-if)# exit
Gye_CFR (config)# ip route 0.0.0.0 0.0.0.0 20.20.20.2
```

Nombre de dispositivo: Cue_AlmacenesJR**Descripción:** Se configura el direccionamiento en la interfaz GigabitEthernet0/0 y GigabitEthernet0/1, además de la ruta estática que le permite comunicarse con los dispositivos del ISP.

```
Cue_AlmacenesJR (config)# interface GigabitEthernet0/0
Cue_AlmacenesJR (config-if)# description HACIA-SW2-
Urdesa_GYE_PE
Cue_AlmacenesJR (config-if)# ip address 20.30.20.2
255.255.255.252
Cue_AlmacenesJR (config-if)# no shutdown
Cue_AlmacenesJR (config-if)# interface
GigabitEthernet0/1
Cue_AlmacenesJR (config-if)# description LAN
Cue_AlmacenesJR (config-if)# ip address 192.168.1.2
255.255.255.0
Cue_AlmacenesJR (config-if)# no shutdown
Cue_AlmacenesJR (config-if)# exit
Cue_AlmacenesJR (config)# ip route 0.0.0.0 0.0.0.0
20.30.20.2
```

Código Fuente

En esta sección se describirán las distintas funciones y su implementación dentro del proyecto. Se excluyen de esta explicación los métodos creados por el framework PyQt Designer.

Login.py

Nombre de archivo: login.py

Nombre de función: showRemoteWindow()

Descripción: Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.

Método de llamado: ingresar()



```
def showRemoteWindow(self):
    self.remoteWindow = QtWidgets.QDialog()
    self.ui = Ui_Remote()
    self.ui.setupUi(self.remoteWindow)
    self.remoteWindow.show()
```

Nombre de archivo: login.py

Nombre de función: showRemoteWindow()

Descripción: Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla. Actualmente este código no se ejecuta ya que la conexión por puerto COM esta deshabilitado en esta versión de MPLStudio.

Método de llamado: ingresar()



```
def showlocalWindow(self):
    self.localWindow = QtWidgets.QDialog()
    self.ui = Ui_local()
    self.ui.setupUi(self.localWindow)
    self.localWindow.show()
```

Nombre de archivo: login.py


Nombre de función: ingresar()

Descripción: Esta función verifica los datos ingresados en el cuadro de texto de login para la aplicación. Realiza conexión con la base de datos bajo SQL y una vez que los datos estén validados muestra la pantalla correspondiente al tipo de conexión. Por defecto en esta versión del aplicativo automáticamente entra a showRemoteWindow()


Método de llamado: self.btn_ingresar.clicked()



```
def ingresar(self):
    username = self.txt_usuario.text()
    password = self.txt_contrasena.text()
    connection = sqlite3.connect("login.db")
    result = connection.execute("SELECT * FROM USERS WHERE
    USERNAME = ? AND CONTRASENA = ?", (username, password))
    if (len(result.fetchall()) > 0):
```

	<pre> print("Usuario encontrado ! ") loginWindow.close() if self.chk_remote.checkState(): self.showRemoteWindow() else: self.showLocalWindow() else: print("Usuario no encontrado !") ctypes.windll.user32.MessageBoxW(0, "Credenciales invalidas", "Error", 0) connection.close() </pre>
<p>Nombre de archivo: login.py</p> <p>Nombre de función: MAIN</p> <p>Descripción: Este es el llamado a la acción MAIN que se ejecuta al momento de abrir la aplicación. Carga la ventana de login</p> <p>Método de llamado: Automático.</p>	
	<pre> if __name__ == "__main__": import sys app = QtWidgets.QApplication(sys.argv) loginWindow = QtWidgets.QDialog() ui = Ui_Dialog() ui.setupUi(loginWindow) loginWindow.show() sys.exit(app.exec_()) </pre>

Remote.py

<p>Nombre de archivo: remote.py</p> <p>Nombre de función: showSelectorLocal()</p> <p>Descripción: Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.</p> <p>Método de llamado: verificarIP()</p>	
	<pre> def showSelectorLocal(self, remote_conn): self.selectorLocal = QtWidgets.QDialog() self.ui = selectorLocal.Ui_SelectorLocal() self.ui.setupUi(self.selectorLocal, remote_conn) self.selectorLocal.show() </pre>
<p>Nombre de archivo: remote.py</p> <p>Nombre de función: verificarIP()</p> <p>Descripción: Obtiene los datos de los campos de texto de ip ingresados por el cliente, verifica que son campos con datos numéricos y que los mismos sean campos coherentes con el formato de dirección ipv4. Si todos los campos están correctos intenta realizar la conexión por medio de ssh. Si ocurre algún problema muestra mensaje popup informando en que campo en específico existe una inconsistencia.</p> <p>Método de llamado: self.btn_conectar.clicked()</p>	



```

def verificarIP(self, Form):
    if self.txt_1.text().isnumeric() and (int(self.txt_1.text())
< 256):
        if self.txt_2.text().isnumeric() and
(int(self.txt_2.text()) < 256):
            if self.txt_3.text().isnumeric() and
(int(self.txt_3.text()) < 256):
                if self.txt_4.text().isnumeric() and
(int(self.txt_4.text()) < 256):

                    print("estableciendo conexion...")
                    ip = self.txt_1.text() + "." +
self.txt_2.text() + "." + self.txt_3.text() + "." +
self.txt_4.text()
                    print(
                        self.txt_1.text() + "." +
self.txt_2.text() + "." + self.txt_3.text() + "." +
self.txt_4.text() + " " + self.txt_usuario.text() + "
" + self.txt_contrasena.text())
                    try:
                        remote_conn_pre, remote_conn =
login_ssh(ip, self.txt_usuario.text(),

self.txt_contrasena.text()) #
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

                    print("1")
                    disable_paging(remote_conn)
                    print("2")
                    back_home(remote_conn)
                    print("3")
                    self.showSelectorLocal(remote_conn)
                    print("4")
                    Form.close()
                    except Exception :
                        ctypes.windll.user32.MessageBoxW(0, "No
se pudo establecer la conexion, verificar ip o credenciales.",
"Error", 0)

                    else:
                        ctypes.windll.user32.MessageBoxW(0, "Error en
el cuarto octeto", "Error", 1)
                    else:
                        ctypes.windll.user32.MessageBoxW(0, "Error en el
tercer octeto", "Error", 1)
                    else:
                        ctypes.windll.user32.MessageBoxW(0, "Error en el
segundo octeto", "Error", 1)
                    else:
                        ctypes.windll.user32.MessageBoxW(0, "Error en el primer
octeto", "Error", 1)

```

SelectorLocal.py

Nombre de archivo: selectorLocal.py**Nombre de función:** showConfiguracionBasicaWindow()**Descripción:** Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.**Método de llamado:** btn_configuracionBasica.clicked ()

```
def
showConfiguracionBasicaWindow(self, SelectorLocal, remote_conn):
    self.configuracionBasicaWindow = QtWidgets.QDialog()
    self.ui = configuracionBasica.Ui_ConfiguracionBasica()
    self.ui.setupUi(self.configuracionBasicaWindow, remote_conn)
    self.configuracionBasicaWindow.show()
    SelectorLocal.close()
```

Nombre de archivo: remote.py**Nombre de función:** showSelectorMPLS()**Descripción:** Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.**Método de llamado:** btn_configuracionMPLS.clicked ()

```
def showSelectorMPLS(self, SelectorLocal, remote_conn):
    self.selectorConfiguracionMPLS = QtWidgets.QDialog()
    self.ui =
    selectorConfiguracionMPLS.Ui_SelectorConfiguracionMPLS()
    self.ui.setupUi(self.selectorConfiguracionMPLS, remote_conn)
    self.selectorConfiguracionMPLS.show()
    SelectorLocal.close()
```

ConfiguracionBasica.py


Nombre de archivo: configuracionBasica.py**Nombre de función:** showPlantilla ()**Descripción:** Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.**Método de llamado:** btn_plantilla.clicked()

```
def showPlantilla(self, Form, remote_conn):
    self.plantilla = QtWidgets.QDialog()
    self.ui = Ui_Plantilla()
    self.ui.setupUi(self.plantilla, remote_conn)
    self.plantilla.show()
    Form.close()
```

Nombre de archivo: configuracionBasica.py**Nombre de función:** showSelectorCredenciales ()**Descripción:** Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.**Método de llamado:** btn_credenciales.clicked()

	<pre>def showSelectorCredenciales(self, Form, remote_conn): self.selectorCredenciales = QtWidgets.QDialog() self.ui = selectorCredenciales.Ui_SelectorCredenciales() self.ui.setupUi(self.selectorCredenciales, remote_conn) self.selectorCredenciales.show() Form.close()</pre>
	<p>Nombre de archivo: configuracionBasica.py</p> <p>Nombre de función: showDireccionamientoLocal ()</p> <p>Descripción: Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.</p> <p>Método de llamado: btn_direccionamiento.clicked()</p>
	<pre>def showDireccionamientoLocal(self, Form, remote_conn): self.direccionamientoLocal = QtWidgets.QDialog() self.ui = direccionamientoLocal.Ui_DireccionamientoLocal() self.ui.setupUi(self.direccionamientoLocal, remote_conn) self.direccionamientoLocal.show() Form.close()</pre>
	<p>Nombre de archivo: configuracionBasica.py</p> <p>Nombre de función: showSelectorLocal ()</p> <p>Descripción: Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.</p> <p>Método de llamado: btn_atras.clicked()</p>
	<pre>def showSelectorLocal(self, Form, remote_conn): self.selectorLocal = QtWidgets.QDialog() self.ui = selectorLocal.Ui_SelectorLocal() self.ui.setupUi(self.selectorLocal, remote_conn) self.selectorLocal.show() Form.close()</pre>

DireccionamientoLocal.py

	<p>Nombre de archivo: DireccionamientoLocal.py</p> <p>Nombre de función: habilitadorInt# () *Reemplazar # por numero de interfaz</p> <p>Descripción: Cuando esta función es llamada habilita y pone visibles los campos de texto para ingresar direcciones ipv4 por el usuario.</p> <p>Método de llamado: habilitadorCampos()</p>
	<pre>####Ejemplo con Interfaz 1 def habilitadorInt1(self): self.txt_ip1_1.setEnabled(True) self.txt_ip1_2.setEnabled(True) self.txt_ip1_3.setEnabled(True) self.txt_ip1_4.setEnabled(True) self.txt_mask1_1.setEnabled(True) self.txt_mask1_2.setEnabled(True) self.txt_mask1_3.setEnabled(True) self.txt_mask1_4.setEnabled(True)</pre>

Nombre de archivo: DireccionamientoLocal.py

Nombre de función: habilitadorCampos(interfaces)

Descripción: Esta función recibe como parámetro una lista de interfaces disponibles del router para configurar, de esta forma obtiene el numero de interfaces (máximo 5), les asigna un ID y las habilita por medio de la función habilitadorInt#()

Método de llamado: automatico




```
def habilitadorCampos(self, interfaces):
    n = len(interfaces)
    if n == 0:
        ctypes.windll.user32.MessageBoxW(0, "No existe ninguna
interfaz disponible para configurar", "Error", 1)
    elif n == 1:
        self.lbl_int1.setText(interfaces[0][0])
        self.lbl_int1.setEnabled(True)
        self.habilitadorInt1()
    elif n == 2:
        self.lbl_int1.setText(interfaces[0][0])
        self.lbl_int1.setEnabled(True)
        self.lbl_int2.setText(interfaces[1][0])
        self.lbl_int2.setEnabled(True)
        self.habilitadorInt2()
    elif n == 3:
        self.lbl_int1.setText(interfaces[0][0])
        self.lbl_int1.setEnabled(True)
        self.lbl_int2.setText(interfaces[1][0])
        self.lbl_int2.setEnabled(True)
        self.lbl_int3.setText(interfaces[2][0])
        self.lbl_int3.setEnabled(True)
        self.habilitadorInt3()
    elif n == 4:
        self.lbl_int1.setText(interfaces[0][0])
        self.lbl_int1.setEnabled(True)
        self.lbl_int2.setText(interfaces[1][0])
        self.lbl_int2.setEnabled(True)
        self.lbl_int3.setText(interfaces[2][0])
        self.lbl_int3.setEnabled(True)
        self.lbl_int4.setText(interfaces[3][0])
        self.lbl_int4.setEnabled(True)
        self.habilitadorInt4()
    elif n >= 5:
        self.lbl_int1.setText(interfaces[0][0])
        self.lbl_int1.setEnabled(True)
        self.lbl_int2.setText(interfaces[1][0])
        self.lbl_int2.setEnabled(True)
        self.lbl_int3.setText(interfaces[2][0])
        self.lbl_int3.setEnabled(True)
        self.lbl_int4.setText(interfaces[3][0])
        self.lbl_int4.setEnabled(True)
        self.lbl_int5.setText(interfaces[4][0])
        self.lbl_int5.setEnabled(True)
        self.habilitadorInt5()
```

Nombre de archivo: DireccionamientoLocal.py



Nombre de función: set#IP () *Reemplazar # por numero ordinal ej: Primera,Segunda,Tercera,etc.*

Descripción: Valida si el usuario ha ingresado datos para determinado campo de ipv4, de ser correcta la validación procede a la configuración en el enrutador, caso contrario muestra mensaje de error.

Método de llamado: setDireccionamiento()

	<pre>####Ejemplo con PrimeraIP def setPrimeraIp(self, int, remote_conn): m1 = self.txt_mask1_1.text() m2 = self.txt_mask1_2.text() m3 = self.txt_mask1_3.text() m4 = self.txt_mask1_4.text() ip1 = self.txt_ip1_1.text() ip2 = self.txt_ip1_2.text() ip3 = self.txt_ip1_3.text() ip4 = self.txt_ip1_4.text() valIp = ip1 + ip2 + ip3 + ip4 valMask = m1 + m2 + m3 + m4 if (valIp != "" and valMask != ""): ip = ip1 + "." + ip2 + "." + ip3 + "." + ip4 mask = m1 + "." + m2 + "." + m3 + "." + m4 if (type(remote_conn) is paramiko.channel.Channel): funciones2.config_dir(remote_conn, int, ip, mask) ctypes.windll.user32.MessageBoxW(0, "Se configuro la interfaz" + int, "Exito", 0) else: funciones_com.config_dir(remote_conn, int, ip, mask)</pre>
---	--

SelectorCredenciales.py

	<p>Nombre de archivo: selectorCredenciales.py</p> <p>Nombre de función: verUsuarios()</p> <p>Descripción: Hace el llamado a un método que devuelve los usuarios existentes en el router y los muestra en pantalla</p> <p>Método de llamado: btn_verUsuarios.clicked()</p> <pre>def verUsuarios(self, remote_conn): funciones2.back_home(remote_conn) lista=funciones2.sh_usernames(remote_conn) time.sleep(2) print(lista) ctypes.windll.user32.MessageBoxW(0, lista , "Información usuarios", 0)</pre>
	<p>Nombre de archivo: selectorCredenciales.py</p> <p>Nombre de función: showAnadirUsuarios()</p> <p>Descripción: Muestra la ventana para poder añadir usuarios al enrutador</p> <p>Método de llamado: btn_anadirUsuario.clicked()</p> <pre>def showAnadirUsuario(self, SelectorCredenciales, remote_conn): print("here 1") self.AnadirUsuario = QtWidgets.QDialog() self.ui = anadirUsuario.Ui_AnadirUsuario() self.ui.setupUi(self.AnadirUsuario, remote_conn) self.AnadirUsuario.show() SelectorCredenciales.close()</pre>

AnadirUsuario.py

Nombre de archivo: AnadirUsuario.py**Nombre de función:** addUsuario()**Descripción:** Valida el tipo de dato de los campos de texto y si todo es correcto añade un usuario al enrutador mediante otra función.**Método de llamado:** btn_anadir.clicked()

```
def addUsuario(self, AnadirUsuario, remote_conn):
    print("#####CODIGO ANADIR USUARIOS")
    print(type(remote_conn))
    if (self.txt_privilegio.text().isnumeric() and
        ((int(self.txt_privilegio.text())) <= 15) and
        self.txt_usuario.text() != "" and self.txt_contrasena.text() != ""):
        funciones2.conf_credencial(remote_conn,
            self.txt_usuario.text(), self.txt_contrasena.text(),
            self.txt_privilegio.text())
        ctypes.windll.user32.MessageBoxW(0, "Configuracion
realizada con éxito", "éxito!", 0)
        self.atras(AnadirUsuario, remote_conn)
    else:
        ctypes.windll.user32.MessageBoxW(0, "Verifica los campos,
asegúrese del valor de privilegio", "Error!!", 0)
```

Plantilla.py

Nombre de archivo: plantilla.py**Nombre de función:** showConfiguracionBasica()**Descripción:** Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.**Método de llamado:** atras()

```
def showConfiguracionBasica(self, Form, remote_conn):
    self.configuracionBasica = QtWidgets.QDialog()
    self.ui = configuracionBasica.Ui_ConfiguracionBasica()
    self.ui.setupUi(self.configuracionBasica, remote_conn)
    self.configuracionBasica.show()
    Form.close()
```


Nombre de archivo: plantilla.py**Nombre de función:** configurar()**Descripción:** Primero valido todos los campos de ingreso de texto. Si los campos necesarios (hostname y domain name) no están completos o no validados la función lanza un mensaje de alerta. Los campos opcionales (dns1, dns2) se validan y se incluyen en la configuración siempre y cuando se detecte contenido en su campo de texto, caso contrario se omiten de la configuración.**Método de llamado:** btn_configurar.clicked()

```
def configurar(self, Form, remote_conn):

    hostname = self.txt_hostname.text()
    domainName = self.txt_domainName.text()
    if
    (self.verificarIP(self.txt_dns1_1, self.txt_dns1_2, self.txt_dns1_3
, self.txt_dns1_4)):
        dns1 =
self.txt_dns1_1.text() + "." + self.txt_dns1_2.text() + "." + self.txt_dn
s1_3.text() + "." + self.txt_dns1_4.text()
    else:
```

	<pre> dns1="" ctypes.windll.user32.MessageBoxW(0, "No se configurará DNS1", "Alerta", 0) if (self.verificarIP(self.txt_dns2_1,self.txt_dns2_2,self.txt_dns2_3 ,self.txt_dns2_4)): dns2 = self.txt_dns2_1.text()+"."+self.txt_dns2_2.text()+"."+self.txt_dns2_3 self.txt_dns2_4.text()+"."+self.txt_dns2_4.text() else: dns2="" ctypes.windll.user32.MessageBoxW(0, "No se configurará DNS2", "Alerta", 0) tipo_ssh = "paramiko" #tipo_ssh = "<paramiko.Channel 0 (open) window=1004 -> <paramiko.Transport at 0x7546c70 (cipher aes128-cbc, 128 bits) (active; 1 open channel(s))>>" print(type(remote_conn) is paramiko.channel.Channel) if type(remote_conn) is paramiko.channel.Channel: if len(hostname)>0 and len(domainName)>0: funciones2.conf_plantilla(remote_conn, hostname, domainName, dns1, dns2) ctypes.windll.user32.MessageBoxW(0, "Configuración realizada con éxito", "Done", 0) else: ctypes.windll.user32.MessageBoxW(0, "Llene los campos necesarios para la configuración (*)", "Error", 0) print("Entro por REMOTE") else: print("Entro por LOCAL") funciones_com.conf_plantilla(remote_conn, hostname, domainName, dns1, dns2) </pre>
--	---



SelectorConfiguracionMPLS.py

Nombre de archivo: selectorConfiguracionMPLS.py Nombre de función: showConfigurarPE() Descripción: Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla. Método de llamado: btn_configurarPE.clicked()	
	<pre> def showConfigurarPE(self,Form, remote_conn): self.configurarPE2 = QtWidgets.QDialog() self.ui = Ui_ConfigurarPE2() self.ui.setupUi(self.configurarPE2, remote_conn) self.configurarPE2.show() Form.close() </pre>
Nombre de archivo: selectorConfiguracionMPLS.py Nombre de función: showConfigurarCE() Descripción: Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.	


Método de llamado: btn_configurarCE.clicked()	
	<pre>def showConfigurarCE(self, Form, remote_conn): self.configurarCE = QtWidgets.QDialog() self.ui = Ui_ConfigurarCE() self.ui.setupUi(self.configurarCE, remote_conn) self.configurarCE.show() Form.close()</pre>
Nombre de archivo: selectorConfiguracionMPLS.py Nombre de función: showSelectorLocal() Descripción: Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla. Método de llamado: btn_atras.clicked()	
	<pre>def showSelectorLocal(self, Form, remote_conn): self.selectorLocal = QtWidgets.QDialog() self.ui = selectorLocal.Ui_SelectorLocal() self.ui.setupUi(self.selectorLocal, remote_conn) self.selectorLocal.show() Form.close()</pre>
Nombre de archivo: selectorConfiguracionMPLS.py Nombre de función: configurarP() Descripción: Envía la configuración al enrutador obteniendo los datos de las funciones del archivo <i>funciones2.py</i> (ver funciones2.py) Método de llamado: btn_configurarP.clicked()	
	<pre>def configurarP(self, remote_conn): if (type(remote_conn) is paramiko.channel.Channel): funciones2.config OSPF(remote_conn) funciones2.save_ID(remote_conn) else: funciones_com.config OSPF(remote_conn) print("1") funciones2.config_cef_mpls_ldp(remote_conn) print("2") ctypes.windll.user32.MessageBoxW(0, "Configuración realizada con éxito", "Done", 0)</pre>



ConfigurarCE.py

Nombre de archivo: configurarCE.py
Nombre de función: showSelectorMPLS()
Descripción: Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.
Método de llamado: btn_atras.clicked()

	<pre>def showSelectorMPLS(self, Form, remote_conn): self.selectorMpls = QtWidgets.QDialog() self.ui = selectorConfiguracionMPLS.Ui_SelectorConfiguracionMPLS() self.ui.setupUi(self.selectorMpls, remote_conn) self.selectorMpls.show() Form.close()</pre>
<p>Nombre de archivo: configurarCE.py</p> <p>Nombre de función: configurar()</p> <p>Descripción: Envía la configuración al enrutador obteniendo los datos de las funciones del archivo <i>funciones2.py</i> (ver funciones2.py)</p> <p>Método de llamado: btn_configurar.clicked()</p>	
	<pre>def configurar(self, Form, remote_conn): if self.verificarIP(): print("ENTROOOOOOOOOOOOOOOOO") funciones2.conf_route_CE(remote_conn, self.txt_1.text() + "." + self.txt_2.text() + "." + self.txt_3.text() + "." + self.txt_4.text()) ctypes.windll.user32.MessageBoxW(0, "Configuración realizada con éxito", "Done", 0)</pre>

ConfigurarPE2.py

<p>Nombre de archivo: configurarPE2.py</p> <p>Nombre de función: showSelectorMPLS()</p> <p>Descripción: Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.</p> <p>Método de llamado: btn_atras.clicked()</p>	
	<pre>def showSelectorMPLS(self, Form, remote_conn): self.selectorMpls = QtWidgets.QDialog() self.ui = selectorConfiguracionMPLS.Ui_SelectorConfiguracionMPLS() self.ui.setupUi(self.selectorMpls, remote_conn) self.selectorMpls.show() Form.close()</pre>
<p>Nombre de archivo: configurarPE2.py</p> <p>Nombre de función: showagregarCliente()</p> <p>Descripción: Esta función permite la navegación entre ventanas, creando la instancia de la ventana y mostrándola en pantalla.</p> <p>Método de llamado: btn_agregarCliente.clicked()</p>	

	<pre>def showAgregarCliente(self, Form, remote_conn): self.agregarCliente = QtWidgets.QDialog() self.ui = agregarCliente.Ui_AgregarCliente() self.ui.setupUi(self.agregarCliente, remote_conn) self.agregarCliente.show() Form.close()</pre>
	<p>Nombre de archivo: configurarPE2.py</p> <p>Nombre de función: confPE()</p> <p>Descripción: Permite la configuración de un dispositivo PE, validando que los campos ingresados para direcciones IP sean correctos. Luego, se procede a configurar los protocolos iBGP, MP-BGP, CEF, LDP, MPLS, creación de VRF y enrutamiento entre dispositivos PE y CE.</p> <p>Método de llamado: btn_anadir.clicked()</p>
	<pre>def ConfPE(self, remote_conn): if self.verificarIP(self.txt_ip1_1, self.txt_ip1_2, self.txt_ip1_3, self.txt_ip1_4): if self.verificarIP(self.txt_ip2_1, self.txt_ip2_2, self.txt_ip2_3, self.txt_ip2_4): if self.verificarIP(self.txt_mask1_1, self.txt_mask1_2, self.txt_mask1_3, self.txt_mask1_4): if self.verificarIP(self.txt_mask2_1, self.txt_mask2_2, self.txt_mask2_3, self.txt_mask2_4): if self.verificarIP(self.txt_mask3_1, self.txt_mask3_2, self.txt_mask3_3, self.txt_mask3_4): if self.txt_vlan.text().isnumeric() and ((int(self.txt_AS.text())) <= 65535) and self.txt_AS.text().isnumeric() and self.txt_vrf.text() != "" and self.txt_interfaz.text() != "" : if (type(remote_conn) is paramiko.channel.Channel): funciones2.back_home(remote_conn) print("1") funciones2.config_iBGP(remote_conn) print("2") time.sleep(1) funciones2.config_MP_BGP(remote_conn) print("3") time.sleep(1) funciones2.config_cef_mpls_ldp(remote_conn) print("4") time.sleep(1) funciones2.config_vrf(remote_conn, self.txt_vrf.text(), self.txt_AS.text(), self.txt_vlan.text()) print("5") time.sleep(1) redCliente = self.txt_ip2_1.text() + "." + self.txt_ip2_2.text() + "." + self.txt_ip2_3.text() + "." + self.txt_ip2_4.text() maskCliente = self.txt_mask2_1.text() + "." + self.txt_mask2_2.text() + "." + self.txt_mask2_3.text() + "." + self.txt_mask2_4.text() dirCliente = self.txt_ip1_1.text() + "." + self.txt_ip1_2.text() + "." + self.txt_ip1_3.text() + "." + self.txt_ip1_4.text() maskDirCliente = self.txt_mask1_1.text() + "." + self.txt_mask1_2.text() + "." + self.txt_mask1_3.text() + "." + self.txt_mask1_4.text() gatewayCliente = self.txt_mask3_1.text() + "." + self.txt_mask3_2.text() + "." + self.txt_mask3_3</pre>

```

_3.text()+"."+self.txt_mask3_4.text()
    print("6")
    time.sleep(1)

funciones2.config_add_interfaz_vrf(remote_conn, self.txt_vrf.text(),
self.txt_vlan.text(),

self.txt_interfaz.text(), dirCliente, maskDirCliente)
    print("7")
    time.sleep(1)

funciones2.config_route_PE_CE(remote_conn, self.txt_vrf.text(),
redCliente, maskCliente,

gatewayCliente)

    print("8")
    time.sleep(1)

funciones2.redistribute_vrf(remote_conn, self.txt_vrf.text())
    print("9")
    time.sleep(4)
    remote_conn.recv(MAX_BUFFER)

self.txt_Clientes.setText(funciones2.show_res(remote_conn))
    ctypes.windll.user32.MessageBoxW(0,
"Configuración realizada con éxito",
"Done", 0)

    else:
        funciones_com.config OSPF(remote_conn)
        print("1")
        funciones_com.config_iBGP(remote_conn)
        print("2")

funciones_com.config_MP_BGP(remote_conn)
    print("3")

funciones_com.config_cef_mpls_ldp(remote_conn)
    print("4")
    funciones_com.config_vrf(remote_conn,
self.txt_vrf.text(), self.txt_AS.text(), self.txt_vlan.text())
    print("5")
    redCliente = self.txt_ip2_1.text() +
"." + self.txt_ip2_2.text() + "." + self.txt_ip2_3.text() + "." +
self.txt_ip2_4.text()

    maskCliente = self.txt_mask2_1.text()
+ "." + self.txt_mask2_2.text() + "." + self.txt_mask2_3.text() + "."
+ self.txt_mask2_4.text()

    dirCliente = self.txt_ip1_1.text() +
"." + self.txt_ip1_2.text() + "." + self.txt_ip1_3.text() + "." +
self.txt_ip1_4.text()

    maskDirCliente =
self.txt_mask1_1.text() + "." + self.txt_mask1_2.text() + "." +
self.txt_mask1_3.text() + "." + self.txt_mask1_4.text()
    print("6")

funciones_com.config_add_interfaz_vrf(remote_conn,
self.txt_vrf.text(), self.txt_vlan.text(),

self.txt_interfaz.text(), dirCliente, maskDirCliente)
    print("7")

```

```

funciones_com.config_route_PE_CE(remote_conn, self.txt_vrf.text(),
redCliente, maskCliente,

self.txt_interfaz.text())

print("8")

funciones_com.redistribute_vrf(remote_conn, self.txt_vrf.text())
print("9")

self.txt_Clientes.setText(funciones_com.show_res(remote_conn))

else:
self.txt_Clientes.setText("Error al
configurar el dispositivo")
else:
self.txt_Clientes.setText("Error al configurar
el dispositivo")
else:
self.txt_Clientes.setText("Error al configurar el
dispositivo")
else:
self.txt_Clientes.setText("Error al configurar el
dispositivo")
else:
self.txt_Clientes.setText("Error al configurar el
dispositivo")
else:
self.txt_Clientes.setText("Error al configurar el
dispositivo")

```

Funciones2.py

Nombre de archivo: funciones2.py

Nombre de función: obtener_FechayHora()

Descripción: Permite obtener la hora y fecha del sistema en cadena de texto.




Método de llamado: varios.



```

def obtener_FechaYHora():
    info=datetime.datetime.now()
    fecha=str(info.day)+"-"+str(info.month)+"-"+str(info.year)
    if info.minute<10:
        minutos="0"+str(info.minute)
    else:
        minutos=str(info.minute)
    if info.second<10:
        segundos="0"+str(info.second)
    else:
        segundos=str(info.second)
    if info.hour<10:
        horas="0"+str(info.hour)

```

	<pre> else: horas=str(info.hour) hora=horas+"H"+minutos+"M"+segundos+"S" return fecha,hora </pre>
	<p>Nombre de archivo: funciones2.py</p> <p>Nombre de función: login_ssh()</p> <p>Descripción: Permite inicializar una conexión por SSH utilizando la librería paramiko, dado por parámetro una dirección IP de alguna de las interfaces del dispositivo, un usuario y contraseña correspondiente a una credencial de acceso para este tipo de conexión del dispositivo objetivo. Se retorna dos objetos llamados remote_conn_pre y remote_conn, este último es el que permite la interacción con la CLI de los dispositivos y es el que se envía por parámetro en todas las funciones que interactúen con la CLI del dispositivo que se este configurando.</p> <p>Método de llamado: varios.</p>
	<pre> def login_ssh(IP, username, password): # Crea instancia de un cliente SSH. remote_conn_pre = paramiko.SSHClient() # Agrega automáticamente hosts no confiables (asegúrese de que esté # bien para la política de seguridad en su entorno). remote_conn_pre.set_missing_host_key_policy(paramiko.AutoAddPolicy()) # Inicia conexión SSH. remote_conn_pre.connect(IP, username=username, password=password, look_for_keys=False, allow_agent=False) # Permite una conexión interactiva con el host. remote_conn = remote_conn_pre.invoke_shell() # Leer 1000 bytes de la salida actual del host. output = remote_conn.recv(MAX_BUFFER) return remote_conn_pre, remote_conn </pre>
	<p>Nombre de archivo: funciones2.py</p> <p>Nombre de función: disable_paging()</p> <p>Descripción: Permite eliminar cualquier texto adicional que se encuentre en la CLI del dispositivo en el que se encuentre haciendo conexión SSH, de forma que se pueda leer la última parte del texto de la CLI con la que se interactúe.</p> <p>Método de llamado: varios.</p>
	<pre> def disable_paging(remote_conn): '''Disable paging on a Cisco router''' remote_conn.send("terminal length 0\r\n") while(not remote_conn.recv_ready()): time.sleep(0.5) output = (remote_conn.recv(MAX_BUFFER)).decode() </pre>
	<p>Nombre de archivo: funciones2.py</p> <p>Nombre de función: back_home()</p> <p>Descripción: Permite volver al modo de configuración privilegiado, con lo que se evita posibles malas configuraciones por encontrarse en el modo de configuración incorrecto.</p> <p>Método de llamado: varios.</p>
	<pre> def back_home(remote_conn): remote_conn.send("\n") while(not remote_conn.recv_ready()): time.sleep(0.5) output = (remote_conn.recv(MAX_BUFFER)).decode() print(output) if('config' in output): remote_conn.send("end\n") </pre>

	<pre> remote_conn.send("\n") while (not remote_conn.recv_ready()): time.sleep(0.5) output = (remote_conn.recv(MAX_BUFFER)).decode() </pre>
	<p>Nombre de archivo: funciones2.py</p> <p>Nombre de función: sh_usernames()</p> <p>Descripción: Permite leer todas las credenciales de acceso para el protocolo SSH del router objetivo, devolviendo una cadena de texto con la información del nombre de usuario de dichas credenciales.</p> <p>Método de llamado: varios.</p>
	<pre> def sh_usernames(remote_conn): list_users = "" remote_conn.send("show start include username\n") while (not remote_conn.recv_ready()): time.sleep(0.5) output = (remote_conn.recv(MAX_BUFFER)).decode() output = output.replace('\r\n', '\n').split('\n') print(output) for i in range(1, len(output) - 1): line = output[i].split(" ") if ("username" in line[0]): list_users += (line[1] + "\n") print("\n" + list_users) return list_users </pre>
	<p>Nombre de archivo: funciones2.py</p> <p>Nombre de función: add_import()</p> <p>Descripción: Permite agregar la importación de rutas de una nueva VRF, de forma que se permita la comunicación entre dispositivos de distintas VRF.</p> <p>Método de llamado: varios.</p>
	<pre> def add_import(remote_conn, name_vrf, vlan): if(vlan.isnumeric()): remote_conn.send("conf t\n") remote_conn.send("ip vrf " + name_vrf + "\n") remote_conn.send("route-target import " + vlan + ":" + vlan + "\n") while (not remote_conn.recv_ready()): time.sleep(0.5) output = (remote_conn.recv(MAX_BUFFER)).decode() output = output.replace('\r\n', '\n') print("/////") print(output) if "please configure" in output: ctypes.windll.user32.MessageBoxW(0, "La VRF ingresada no tiene un route distinguisher asignado", "Error", 0) else: ctypes.windll.user32.MessageBoxW(0, "Configuración realizada con éxito", "Done", 0) back_home(remote_conn) else: ctypes.windll.user32.MessageBoxW(0, "VLAN incorrecta", "Error", 0) </pre>
	<p>Nombre de archivo: funciones2.py</p> <p>Nombre de función: config_dir()</p> <p>Descripción: Permite configurar el direccionamiento en una interfaz, dado la interfaz, la dirección IP y la</p>

máscara de subred por parámetro para realizar esta configuración.

Método de llamado: varios.



```
def config_dir(remote_conn, interfaces, IPs, masks):
    remote_conn.send("conf t\n")
    remote_conn.send("int "+interfaces+ "\n")
    remote_conn.send("ip add "+IPs+" "+masks+ "\n")
    remote_conn.send("no sh\n")
    remote_conn.send("do wr\n")
    while (not remote_conn.recv_ready()):
        time.sleep(0.5)
    back_home(remote_conn)
```

Nombre de archivo: funciones2.py

Nombre de función: conf_plantilla()

Descripción: Permite configurar la plantilla básica de un enrutador, así como su nombre de dominio, servidores dns, hostname y mensaje de bienvenida al conectarse al dispositivo.

Método de llamado: varios.



```
def conf_plantilla(remote_conn, hostname, domain_name, dns1, dns2):
    remote_conn.send("conf t\n")
    if (len(dns1)>0):
        remote_conn.send("ip name-server "+dns1+"\n")
    if (len(dns2)>0):
        remote_conn.send("ip name-server " + dns2+"\n")
    remote_conn.send("host "+hostname+"\n"+"ip domain-name "+domain_name+"\n"+
        "banner motd # ACCESO SOLO A PERSONAL AUTORIZADO#\n"+
        "line vty 0 4\n"+
        "transport input all\n"+
        "login local\n"+
        "exec-timeout 3 3\n"+
        "logging synchronous\n"+
        "line console 0\n"+
        "transport output all\n"+
        "login local\n"+
        "exec-timeout 3 3\n"+
        "logging synchronous\n"+
        "do wr\n")
    while (not remote_conn.recv_ready()):
        time.sleep(0.5)
    back_home(remote_conn)
```

Nombre de archivo: funciones2.py

Nombre de función: conf_credencial()

Descripción: Permite agregar una nueva credencial de acceso SSH en un enrutador, dado un nombre de usuario, contraseña y privilegio de dicha credencial.

Método de llamado: varios.



```
def conf_credencial(remote_conn, user, password, privilegio):
    try:
        remote_conn.send("conf t" + "\n"+
            "username "+user+" privilege "+privilegio+"
secret "+password+"\n"+
            "do wr" + "\n")
        while (not remote_conn.recv_ready()):
            time.sleep(0.5)
        back_home(remote_conn)
    except Exception:
```

```
ctypes.windll.user32.MessageBoxW(0, "Ocurrió un error",
                                   "Error", 1)
```

Nombre de archivo: funciones2.py

Nombre de función: get_dirs_red()

Descripción: Permite obtener todas las direcciones de red conectadas directamente a un enrutador, de forma que estas redes puedan ser anunciadas por OSPF.

Método de llamado: varios.



```
def get_dirs_red(remote_conn):
    list_dirs_red = []
    remote_conn.send("sh ip route connected | exclude /32\n")
    while (not remote_conn.recv_ready()):
        time.sleep(0.5)
    time.sleep(3)
    output = (remote_conn.recv(MAX_BUFFER)).decode()
    output = output.replace('\r\n', '\n').split('\n')
    print(output)
    for i in range(8, len(output) - 1):
        if ("connected" in output[i] and "/" in output[i] and not("ip"
in output[i])):
            line = output[i].split(" ")
            print(" OSPF")
            for j in range(0, len(line)):
                if ("/" in line[j]):
                    list_dirs_red.append(line[j])
                    break
    back_home(remote_conn)
    return list_dirs_red
```

Nombre de archivo: funciones2.py

Nombre de función: get_wildcard()

Descripción: Permite obtener las wildcards de una lista de redes dadas por parámetro.

Método de llamado: varios.



```
def get_wildcard(list_dirs_red):
    list_wildcards = []
    for k in range(0, len(list_dirs_red)):
        line = list_dirs_red[k].split('/')
        mask = 32 - int(line[1])
        wildcard = [0, 0, 0, 0]
        octeto = 3
        while (mask > 8):
            wildcard[octeto] = 2 ** 8 - 1
            octeto -= 1
            mask -= 8
        wildcard[octeto] = 2 ** mask - 1
        list_wildcards.append(
            str(wildcard[0]) + '.' + str(wildcard[1]) + '.' +
            str(wildcard[2]) + '.' + str(wildcard[3]))
    return list_wildcards
```

Nombre de archivo: funciones2.py

Nombre de función: get_ip_interfaz()

Descripción: Permite obtener una lista con todas las interfaces que tiene un enrutador, esta información comprende el nombre de las interfaces, su estado (up/down) y la dirección IP asignada a la misma.

Método de llamado: varios.



```
def get_ip_interfaz(remote_conn):
    print("int")
    back_home(remote_conn)
    info_interfaz=[]
    remote_conn.send("sh ip int b\n")
    while (not remote_conn.recv_ready()):
        time.sleep(0.5)
    output = (remote_conn.recv(MAX_BUFFER)).decode()
    output = output.replace('\r\n', '\n').split('\n')
    for i in range(1, len(output)-1):
        line = output[i].split()
        print(line)
        if not("#" in line[0]) and "/" in line[0]:
            if(len(line[0].split("."))==1):
                interfaz = [line[0], line[4], line[1]]
                if line[4]!="up":
                    interfaz[1]="down"
                info_interfaz.append(interfaz)
    back_home(remote_conn)
    return info_interfaz
```

Nombre de archivo: funciones2.py



Nombre de función: get_ospf_neig()





Descripción: Permite obtener los vecinos de OSPF, lo cual permite configurar el protocolo iBGP y MP-BGP entre los dispositivos PE.



Método de llamado: varios.



```
def get_ospf_neig(remote_conn):
    loopback_rd=""
    back_home(remote_conn)
    remote_conn.send("show ip interface brief | include Loopback\n")
    time.sleep(3)
    while (not remote_conn.recv_ready()):
        time.sleep(0.5)
    output = (remote_conn.recv(MAX_BUFFER)).decode()
    output = output.replace('\r\n', '\n').split('\n')
    for i in range(0, len(output)):
        line = output[i].split()
        if "Loopback" in line[0]:
            print(line[0])
            loopback_rd = (line[1])
    neighbors=[]
    list_neig=[]
    print("----" + loopback_rd)
    remote_conn.send("sh ip ospf database network | include Attached Router\n")
    while (not remote_conn.recv_ready()):
        time.sleep(0.5)
    output = (remote_conn.recv(MAX_BUFFER)).decode()
    output = output.replace('\r\n', '\n').split('\n')
    for i in range(1, len(output)):
        print(output[i])
        if('Attached Router' in output[i]):
            list_neig.append((output[i].split(":"))[1])
    for j in range(0, len(list_neig)):
        dir = list_neig[j][1:]
        if not(dir in neighbors) and dir!=loopback_rd:
            neighbors.append(dir)
    print(neighbors)
    back_home(remote_conn)
    return neighbors
```

	<p>Nombre de archivo: funciones2.py</p> <p>Nombre de función: config OSPF()</p> <p>Descripción: Permite configurar el protocolo OSPF en el AS 1 y en el área de backbone 0. Para esto se obtienen todas las direcciones de redes conectadas directamente de un enrutador, posteriormente la wildcard de cada una de estas redes y son anunciadas por el protocolo ya mencionado.</p> <p>Método de llamado: varios.</p>
	<pre>def config_OSPF(remote_conn): list_dirs_red = get_dirs_red(remote_conn) list_wildcards = get_wildcard(list_dirs_red) print(list_dirs_red) print(list_wildcards) remote_conn.send("conf t\n") remote_conn.send("router ospf 1\n") for i in range(0, len(list_dirs_red)): remote_conn.send("network "+list_dirs_red[i].split('/')[0]+" "+list_wildcards[i]+" area 0"+"\n") while (not remote_conn.recv_ready()): time.sleep(0.5) back_home(remote_conn)</pre>
	<p>Nombre de archivo: funciones2.py</p> <p>Nombre de función: conf_mpls_interfaces()</p> <p>Descripción: Permite habilitar mpls en todas las interfaces habilitadas (estado up) y que no sean las interfaces Loopback. Para esto se obtiene primero todas las interfaces disponibles del enrutador con la función get_ip_interfaz.</p> <p>Método de llamado: varios.</p>
	<pre>def conf_mpls_interfaces(remote_conn): back_home(remote_conn) interfaces = get_ip_interfaz(remote_conn) print(interfaces) remote_conn.send("conf t\n") for i in range(0, len(interfaces)): if (interfaces[i][1]=="up" and not("Loopback" in interfaces[i][0])): print(i) remote_conn.send("int "+interfaces[i][0]+" \n") remote_conn.send("mpls ip\n") remote_conn.send("exit\n") while (not remote_conn.recv_ready()): time.sleep(0.5) time.sleep(5)</pre>
	<p>Nombre de archivo: funciones2.py</p> <p>Nombre de función: config_cef_mpls_ldp()</p> <p>Descripción: Se configura el protocolo CEF, LDP y MPLS.</p> <p>Método de llamado: varios.</p>

	<pre>def config_cef_mpls_ldp(remote_conn): remote_conn.send("conf t\n") remote_conn.send("ip cef\n") remote_conn.send("mpls label protocol ldp\n") remote_conn.send("mpls ldp router-id loopback0\n") remote_conn.send("mpls ip\n") print("2.0") conf_mpls_interfaces(remote_conn) back_home(remote_conn)</pre>
<p>Nombre de archivo: funciones2.py</p> <p>Nombre de función: config_vrf()</p> <p>Descripción: Se crea una VRF con su respectiva router distinguisher, exportación e importación de sus rutas.</p> <p>Método de llamado: varios.</p>	
	<pre>def config_vrf(remote_conn, name_vrf, AS, vlan): remote_conn.send("conf t\n") remote_conn.send("ip vrf "+name_vrf+"\n") remote_conn.send("rd "+AS+": "+vlan+"\n") remote_conn.send("route-target export "+vlan+": "+vlan+"\n") remote_conn.send("route-target import " + vlan + ":" + vlan + "\n") while (not remote_conn.recv_ready()): time.sleep(0.5)</pre>
<p>Nombre de archivo: funciones2.py</p> <p>Nombre de función: config_add_interfaz_vrf()</p> <p>Descripción: Se agrega una interfaz a una VRF, para lo cual se requiere el nombre de la VRF, la VLAN del cliente, la interfaz por la que se conectará el cliente y el direccionamiento de la interfaz por donde se conectará el cliente. Esto se configura en un dispositivo tipo PE.</p> <p>Método de llamado: varios.</p>	
	<pre>def config_add_interfaz_vrf(remote_conn, name_vrf, vlan, interfaz, ip, mascara): remote_conn.send("interface "+interfaz+"\n") remote_conn.send("no sh\n") remote_conn.send("no mpls ip\n") remote_conn.send("interface "+interfaz+"."+vlan+"\n") remote_conn.send("encapsulation dot1Q "+vlan+"\n") while (not remote_conn.recv_ready()): time.sleep(0.5) remote_conn.send("ip vrf forwarding " + name_vrf + "\n") remote_conn.send("ip add "+ip+" "+mascara+" "\n") while (not remote_conn.recv_ready()): time.sleep(0.5) back_home(remote_conn)</pre>
<p>Nombre de archivo: funciones2.py</p> <p>Nombre de función: config_route_PE_CE()</p> <p>Descripción: Permite configurar una ruta que le permite comunicarse por su respectiva cliente a un cliente, recibiendo como parámetro el nombre de la VRF y la red de la LAN del cliente.</p> <p>Método de llamado: varios.</p>	
	<pre>def config_route_PE_CE(remote_conn, name_vrf, red_CE, mascara_CE, int_salida): remote_conn.send("conf t\n") remote_conn.send("ip route vrf "+name_vrf+" "+red_CE+" "+mascara_CE+" "+int_salida+"\n") while (not remote_conn.recv_ready()): time.sleep(0.5)</pre>

	<code>back_home(remote_conn)</code>
Nombre de archivo: funciones2.py Nombre de función: redistribute_vrf() Descripción: Permite redistribuir las rutas aprendidas de un dispositivo CE por las rutas estáticas, al proceso MP-BGP usando el address-family de la respectiva VRF dada por parámetro. Método de llamado: varios.	
	<pre>def redistribute_vrf(remote_conn, name_vrf): remote_conn.send("conf t\n") remote_conn.send("router bgp 1\n") remote_conn.send("address-family ipv4 vrf "+name_vrf+"\n") remote_conn.send("redistribute static\n") remote_conn.send("do wr\n") while (not remote_conn.recv_ready()): time.sleep(0.5) back_home(remote_conn)</pre>
Nombre de archivo: funciones2.py Nombre de función: conf_route_CE() Descripción: Se configura una ruta estática que le permite comunicarse al router del cliente con su respectivo router PE. Se desactiva el protocolo OSPF ya que se configura previamente al establecer el direccionamiento en el enrutador. Método de llamado: varios.	
	<pre>def conf_route_CE(remote_conn, ip_salida): remote_conn.send("conf t\n") remote_conn.send("no router ospf 1\n") remote_conn.send("ip route 0.0.0.0 0.0.0.0 "+ip_salida+" \n") remote_conn.send("do wr\n") while (not remote_conn.recv_ready()): time.sleep(0.5) back_home(remote_conn)</pre>
Nombre de archivo: funciones2.py Nombre de función: config_iBGP() Descripción: Permite configurar el protocolo iBGP entre los dispositivos PE. Para esto, se lee las ip de loopback de los dispositivos P con la función read_file de un archivo de texto, ya que se agregará como vecinos iBGP a los vecinos OSPF, siendo los dispositivos P vecinos OSPF pero no pueden ser vecinos iBGP. Por lo que de esta manera se evita establecer a un dispositivo P como vecino iBGP. Método de llamado: varios.	



```
def config_iBGP(remote_conn):
    bgp_neigh = get_ospf_neig(remote_conn)
    rd_p = read_file()
    remote_conn.send("conf t\n")
    remote_conn.send("router bgp 1\n")
    for i in range (0, len(bgp_neigh)):
        if not (bgp_neigh[i] in rd_p):
            remote_conn.send("neighbor "+bgp_neigh[i]+" remote-as
1\n")
            remote_conn.send("neighbor " + bgp_neigh[i] + " update-
source loopback0\n")
            remote_conn.send("neighbor " + bgp_neigh[i] + " next-hop-
self\n")
            remote_conn.send("no auto-summary\n")
    while (not remote_conn.recv_ready()):
        time.sleep(0.5)
    back_home(remote_conn)
```

Nombre de archivo: funciones2.py

Nombre de función: config_MP_BGP()

Descripción: Permite configurar el protocolo MP-BGP entre los dispositivos PE. Para esto, se lee las ip de loopback de los dispositivos P con la función read_file de un archivo de texto, ya que se agregará como vecinos MP-BGP a los vecinos OSPF, siendo los dispositivos P vecinos OSPF, pero no pueden ser vecinos MP-BGP. Por lo que de esta manera se evita establecer a un dispositivo P como vecino MP-BGP.

Método de llamado: varios.



```
def config_MP_BGP(remote_conn):
    bgp_neigh = get_ospf_neig(remote_conn)
    rd_p = read_file()
    remote_conn.send("conf t\n")
    remote_conn.send("router bgp 1\n")
    remote_conn.send("address-family vpnv4\n")
    print("en BGP ")
    print(bgp_neigh)
    for i in range(0, len(bgp_neigh)):
        if not (bgp_neigh[i] in rd_p):
            remote_conn.send("neighbor " + bgp_neigh[i] + "
activate\n")
            remote_conn.send("neighbor " + bgp_neigh[i] + " send-
community extended\n")
            while (not remote_conn.recv_ready()):
                time.sleep(0.5)
            remote_conn.send("exit-address-family\n")
            remote_conn.send("end\n")
    while (not remote_conn.recv_ready()):
        time.sleep(0.5)
    back_home(remote_conn)
```

Nombre de archivo: funciones2.py

Nombre de función: show_res()

Descripción: Permite obtener la salida del comando show vrf, el cual es mostrado si se realiza con éxito toda la configuración correspondiente a un enrutador tipo PE.

Método de llamado: varios.



```
def show_res(remote_conn):
    copy = False
    salida = ""
    remote_conn.recv(1000)
    remote_conn.send("show vrf\n")
    while (not remote_conn.recv_ready()):
        time.sleep(0.5)
    time.sleep(1)
    output = (remote_conn.recv(MAX_BUFFER)).decode()
    output = output.split('\r\n')
    for i in range(0, len(output)):
        if("show vrf" in output[i]):
            copy = True
            if (copy == True):
                salida+=(output[i]+"\r\n")
    print(output)
    return salida
```

Nombre de archivo: funciones2.py

Nombre de función: save_ID()

Descripción: Permite guardar en un archivo de texto la dirección ip de la interfaz loopback de un enrutador, esto se hace en los P y es considerado en la configuración de los protocolos iBGP y MP-BGP como ya se explicó en sus respectivas funciones de configuración.

Método de llamado: varios.



```
def save_ID(remote_conn):
    back_home(remote_conn)
    remote_conn.send("show ip interface brief | include Loopback\n")
    while (not remote_conn.recv_ready()):
        time.sleep(0.5)
    output = (remote_conn.recv(MAX_BUFFER)).decode()
    output = output.replace('\r\n', '\n').split('\n')
    for i in range(0, len(output)):
        line = output[i].split()
        if "Loopback" in line[0]:
            write_file(line[1])
    back_home(remote_conn)
```

Nombre de archivo: funciones2.py

Nombre de función: write_file()

Descripción: Permite escribir en el archivo de texto rd_p.txt la información dada por parámetro, la cual corresponde a la dirección IP de loopback de un enrutador tipo P.

Método de llamado: varios.



```
def write_file(dir_id):
    archivo = open('rd_P.txt', 'a')
    archivo.write(dir_id+'\n')
    archivo.close()
```

Nombre de archivo: funciones2.py

Nombre de función: read_file()

Descripción: Permite leer el archivo rd_p.txt, obteniendo así la dirección IP de loopback de todos los enrutadores tipo P de la red.

Método de llamado: varios.



```
def read_file():
    rd_p = []
    archivo = open('rd_P.txt', 'r')
    lines = archivo.readlines()
    for i in range (0, len(lines)):
        rd_p.append(lines[i][:-1])
    print(rd_p)
    return rd_p
```

Database.py

Nombre de archivo: database.py

Descripción: Script individual, al ser corrido se actualiza la tabla de usuarios y contraseñas del aplicativo. Reemplazar en los campos 'USUARIO' 'PRIVILEGIO' 'CONTRASENA' por las credenciales de la persona que desea agregar a la base de datos.



```
import sqlite3

def createTable():
    connection = sqlite3.connect('login.db')

    connection.execute("INSERT INTO USERS
VALUES (?, ?, ?)", ('USUARIO', 'PRIVILEGIO', 'CONTRASENA'))

    connection.commit()

    result = connection.execute("SELECT * FROM USERS")

    for data in result:
        print("Username : ", data[0])
        print("Privilegio : ", data[1])
        print("Password : ", data[2])

    connection.close()

createTable()
```

Soporte

Para más información o algún problema comunicarse con el equipo de soporte del software.

- jpcadena@espol.edu.ec
- ilcedeno@espol.edu.ec
- caencede@espol.edu.ec
- yofiguer@espol.edu.ec
- aaorella@espol.edu.ec
- lvrivas@espol.edu.ec