Integrated Management of Transport and Commutation Resources over the Network Layer

Marius-Ioan CANDREA-BOZGA and Petrică CIOTÎRNAE

***Abstract*—In the information and automation era, the activities of today's society are based on communications media with fast and secure transmission of information. Because of the information dependency, today's networking solutions must transport more data traffic and deliver it in a fast and efficient way. It is useless to assign an appropriate epithet to the degree of importance that efficient and accurate design and the use of a management mechanism appropriate to the necessity, have in the context of the current society. The usage of programming via Python for increasing management capabilities is required in order to do high performance management of a network. One of the most important aspects of managing a large scale network that uses various platforms bought from different vendors is how to manage all these systems using a unique management platform. The current paper is concerned with Python capabilities put to good use, to develop an Integrated Management platform that performs the management of the devices that form or are part of the telecommunications network. The topologies and the script programming were done in GNS3 using Ubuntu Docker Containers with Python installed, to remotely configure CISCO IOS MultyLayer Switches and Routers.**

***Index Terms*—Integrated management; Network management using Python; Python in networking.**

I. INTRODUCTION

The present-day society we live in is beginning to show its dependency on information. Everything that exists consists of information and it is necessary for many to accomplish their daily working tasks. This need for information is increasing the data traffic which is getting bigger and harder to manage every day.

Higher amounts of information being transferred from here to there are the main reason for evolving and creating better equipment with specs that enhance switching and routing capabilities, and protocol developments that allow a better data flow control over the network [1].

All these pieces of equipment which form the communications network need to be managed and it is not possible for a network administrator team that is located in a town at an office to physically travel the country just to deploy a configuration on a device, so it was developed a remote management type [4]. The remote management means having logical access to the device as it was in front of you. Instead of transferring a device located 1000 km away; one can just configure it by using a remote connection management tool. Two protocols used for this kind of management are Telnet and SSH - Secure Shell [6].

The action of managing a communications network represents the actions through which network supervision, control and management are achieved. The control, supervision and maintenance functions underpin these objectives.

Telnet protocol offers support for connecting to a device which has the support for it and managing it from a remote location. There are some requirements that are expected to be configured before starting a Telnet session with a device. These requirements are: configuring a hostname, a logical address, an enable password (for data network devices), a user and a password and at least one VTY line that permits telnet traffic. The main problem of the Telnet protocol is the lack of security within its session. All the data sent across the network are being sent as plain text, neither one of these sent packets aren't being encrypted.

SSH protocol offers a secure alternative for remote management. The SSH creates encrypted sessions using a public and private key and authenticates the user that tries to login by comparing the credentials configured to those entered and then granting access to the legitimate users.

The main focus of a network administrator is to centralize all the devices within a monitorization platform and perform integrated management [11]. A communications network consists of different devices that have different uses and capabilities and need different configuration, because a switch has a different configuration from a router or a firewall. Because of the high amount of data consumption and the exhaustion of analogical technology development, there is a powerful migration of all devices from analogical to digital switching.

Integrated management of all these resources is being done by some producers within proprietary platforms like PRIME or FABRIC or others like that. The main problem of these platforms is the interoperability with different OS - Operating Systems and different syntax forms they use, to configure the equipment and implement their solutions [10].

The goal of the article is to realize that you can create an integrated management of both commutation and transport resources over the network layer, free of cost, without buying licenses. In order to achieve this product we will use Python and libraries that implement Telnet and SSH protocols.

II. PYTHON AND PYTHON MODULES

Python is a programming language derived directly from ABC scripted language. This language is kind-of new because of its relatively young age of use. The founder of the language is the well known Guigo van Rossum who started developing Python in the early year of 1989 from which he developed a strong and powerful tool for programmers to use worldwide for free. A very important characteristic of Python is that it can function as a pegboard language that connects multiple software components that

M.-I. CANDREA-BOZGA is with the Military Technical Academy "Ferdinand I", Communication Department, Bucharest, Romania (e-mail: candrea.mariusioan@gmail.com).
P. CIOTÎRNAE is with the Military Technical Academy "Ferdinand I", Communication Department, Bucharest, Romania.

Digital Object Identifier 10.32754/JMT.2019.1.05

are independent in a flexible environment using a simple syntax. The ease with which this language is programmed dictates that it can serve as a programming environment for both students and application development experts.

The functionality and flexibility of the programming language is provided by its ability to work with certain libraries, also called modules. These libraries contain functions and protocol implementations that allow developers to create relatively small size but very powerful scripts that can be reused as functions in other scripts. Most modules are integrated into the Python architecture and are part of the standard library. In order to increase the effectiveness of the programming language, its developers have allowed the addition of non-standard libraries, with added features being part of the extended Python library [1].

One of the libraries that will be used to meet the goal of developing integrated network resource management is TelnetLib. TelnetLib is a standard Python Library and is the implementation of the Telnet protocol which is being used to make connections to network devices in order to manage them.

The second module of interest for the purpose of the work is Paramiko. This module is the implementation of SSHv2 protocol which is used to establish encrypted sessions between source and destination. The Paramiko library is an extended Python library so it is not Python proprietary. More information about the library can be found at the official Paramiko website which is stated in the References section.

The most important module for this article is Netmiko. This is a Python module in which the SSHv2 protocol was implemented. The development of the Netmiko module was carried out from the Paramiko module itself and aimed at eliminating the complexity of establishing a secure connection and increasing the compatibility level to achieve SSHv2 connections with different network equipment [3].

In the near future, it will be necessary to address another management concept, one in which programming is a part of this management. Most vendors have already done this, and have built platforms that allow, or even recommend, the use of programming environments to streamline management actions, and one of the most used is Python with its API - Application Programming Interfaces.

The field of communications is a dynamic environment, subject to continuous changes and moving towards a virtualization of the equipment. This virtualization means creating a single, highly-suited hardware component inside which virtual machines are created, which in turn are transformed into virtual equipment. These devices work exactly like physical ones, with the same operating system, the necessary resources being allocated by an administrator or through an automation process that transfers resources by work needs.

This transfer from physical to virtual allows the introduction of new concepts of communications resource management.

III. PYTHON IMPLEMENTATIONS

In the first script that will be presented here, you will see a part of the configuration that will be sent to a range of switches, which will be configured as shown. The range of switches is contained inside a file named *ipswitch*.

```
!/usr/bin/env python

import getpass
import sys
import telnetlib

# Credentials required to login on the devices
utilizator = raw_input("Introduceti credentialele telnet: ")
parola = getpass.getpass()
# Open the file that contains the Switches IP's
s = open ("ipswitch")
for line in s:
        print "Se configureaza switch-ul " + (line)
        HOST = line.strip()
        tn = telnetlib.Telnet(HOST)
        # It is necesary to use the exact lines that appear at login in the equipment
        tn.read_until("Username: ")
        tn.write(utilizator + "\n")
        if parola:
                tn.read_until("Password: ")
                tn.write(parola + "\n")
        # CONFIGURATION that will be written on the equipment
        tn.write("configure terminal\n")
        for n in range (2,36):
        tn.write("vlan " + str(n) + "\n")
        tn.write("name VLanPython" + str(n) + "\n")
        tn.write("exit\n")
        for n in range (0,4):
                tn.write("interface GigabitEthernet3/" + str(n) + "\n")
                tn.write("switchport mode access\n")
                tn.write("switchport access vlan 1" + str(n) + "\n")
                tn.write("exit\n")
        tn.write("end\n")
        # Write down the configuration in the memory of the Device
        tn.write("write\n")
        tn.write("exit\n")
        # Every command or set entered will be returned
        print tn.read_all()
```

Figure 1. Script using TelnetLib module to configure switches

In the second image it can be seen how the script conducts Telnet traffic and configures both Router and Switch devices. As for the Switch, it can be seen in the information returned to the Programmer station, on the left window inside the picture [2].

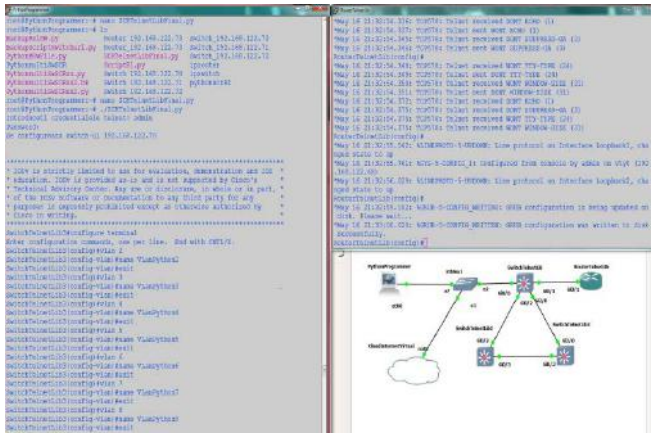

Figure 2. The topology used for simulation in GNS3 and Script writing config on Router and Switch using TelnetLib module

As can be seen from previous images and explanations, the TelnetLib library meets the requirements of an integrated management but does not meet the minimum security requirements. The following deployed module, through which a more complex management solution is presented using another Python library, but implementing the SSH protocol, provides the necessary security requirements in a communications network.

Unlike TelnetLib, Paramiko is not a standard Python library and requires the installation of packages in the Linux operating system for use in Python. As an SSH protocol implementation, Paramiko provides a secure (encrypted) connection from source to destination. Paramiko's command-line writing capabilities are not so different from those of the TelnetLib library, with the difference being functions, syntax, commands, and interoperability.

 Digital Object Identifier 10.32754/JMT.2019.1.05

```
#!/usr/bin/dev python
import paramiko
import time
# Credentials used for login
utilizator = "admin"
parola = "paramiko"
s = open ("iprswitch")
for line in s:
        adresaip = line.strip()
        # Establishing conexion
        ssh_client = paramiko.SSHClient()
        print "Se efectueaza conectarea la RouterSwitch-ul" + line
        # Aquiring a security policy automatically
        ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh_client.connect(hostname=adresaip,username=utilizator,password=parola)
        # Confirming establishing SSH conexion
        print "Conectare la RSwitch ", adresaip," realizata cu succes!"
        # Within SSH conexion transmit Instruction set ##2 and ##3
        remote_connection = ssh_client.invoke_shell()
        ##2  2nd Instruction Set for RSw(RouterSwitch)
        remote_connection.send("configure terminal\n")
        remote_connection.send("no int loop 0\n")
        remote_connection.send("ip address 1.1.1.1 255.255.255.255\n")
        remote_connection.send("no int loop 1\n")
        remote_connection.send("ip address 2.2.2.2 255.255.255.255\n")
        remote_connection.send("router ospf 1\n")
        remote_connection.send("network 0.0.0.0 255.255.255.255 area 0\n")
        ##3 3rd Instruction Set for RSw(RouterSwitch)
        for n in range (2,21):
           print "Se creeaza vlan-ul " + str(n)
           remote_connection.send(" vlan " + str(n) +  "\n")
           remote_connection.send("name VlanPython " + str(n) +  "\n")
           time.sleep(0.5)
        for n in range (0,4):
                remote_connection.send("interface Gigabit2/" + str(n) +"\n")
                remote_connection.send("switchport trunk encapsulation dot1q\n")
                remote_connection.send("switchport mode trunk\n")
                remote_connection.send("description ===WAN " + str(n) + "Trunk
Port===\n")
                remote_connection.send("switchport trunk allowed vlan 20-100\n")
                time.sleep(1)
        remote_connection.send("end\n")
        remote_connection.send("write\n")
        time.sleep(1)
        # dateiesire will contain all infos above written on equipments
        dateiesire = remote_connection.recv(65535)
        # Print variable dateiesire
        print dateiesire
```

Figure 3. Script using Paramiko module to configure RSW

The above presented script fulfils management for both layer 2 and layer 3 equipment, as it can be seen in Fig. 4 where information about the configurations sent via script appear above the topology [5].

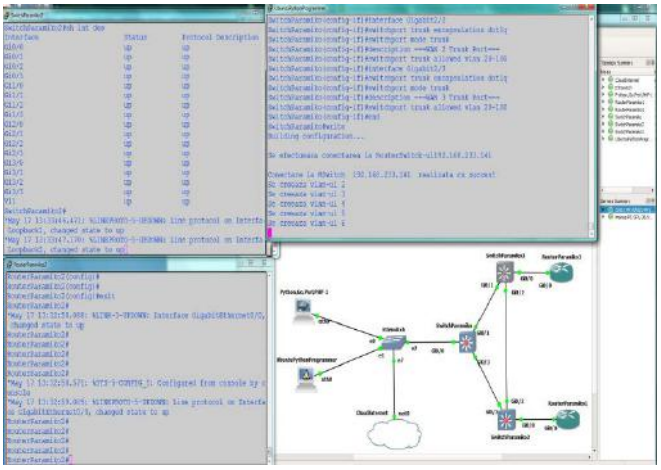

Figure 4. The topology used for simulation in GNS3 and Script writing config on Router and Switch using Paramiko module

The Paramiko module is characterized as having a hard and complex implementation and therefore another SSHv2 implementation module has been created in order to enhance Paramiko and focus its uses on establishing sessions with network equipment.

Netmiko is presented as a “multi-vendor” library [7]. The explanation of this concept could be described as being compatible with as many platforms as possible from as many equipment manufacturers as possible. In addition to this increased level of compatibility, Netmiko simplifies the use of the library for transmitting instruction sets across the network to the device with which the SSHv2-encrypted connection has been made.

In the image below, you can see the network topology simulated in GNS3, a network where Python code was inserted through the script hereby presented. The Python running code has configured topology switches and routers, all actions being done according to the set of instructions assigned to each file. Each set of devices has been assigned a text file from which the configuration commands have been extracted.

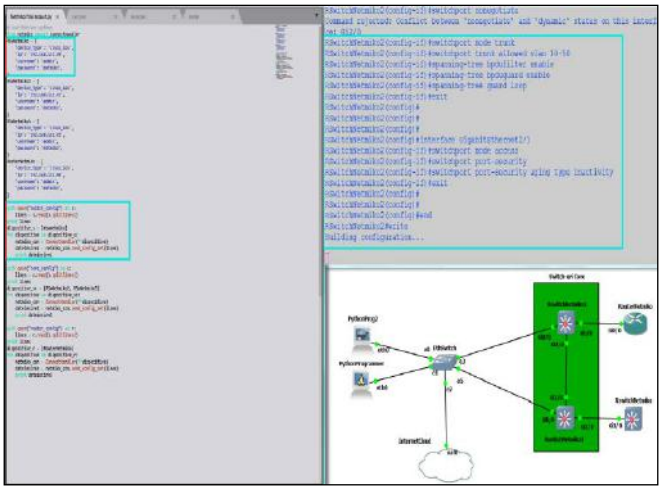

Figure 5. The topology used for simulation in GNS3 and the Script used for writing config on Router and Switch using Netmiko module

The most important script obtained using Netmiko was creating an app that finds a MAC Address on switch equipment. The complexity of the script requires knowledge from both CISCO technology configuring and Python programming functionality to be implemented.

At the time of running the macsearch.py script, it was given the initial data needed to find the MAC address, namely, an IP address on which to start the search for the host, this being the core switch in the topology used, then passing through each switch to the destination device where this MAC address was found to belong to its own interface. Successive passes through the switch devices are marked to know the path that the MAC address has on the network [8].

The reason for returning mentioned information lies at the root of the remedy of the case, namely that the information may be used to reconfigure the equipment (or the interface) or even to close the port on which the network connection was made.

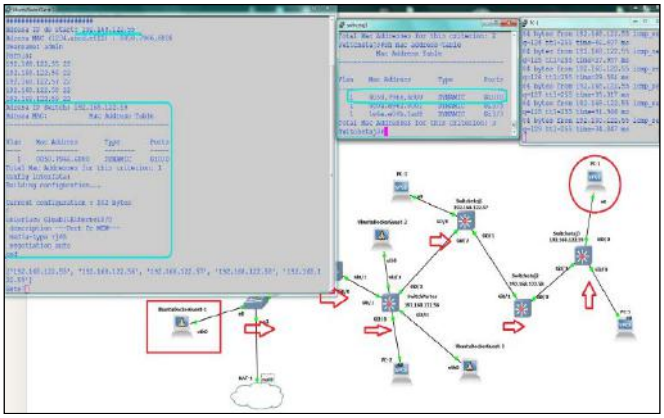

Figure 6. The topology used for MAC Search in GNS3

As can be seen, the search script of a MAC address successfully performed the search and display of the data presented above.

```
from netmiko import ConnectHandler, SCPConn
from netmiko import Netmiko
import time
import getpass
import socket

print("#"*23)
print("#"+" MAC Layer 2 Network "+"#")
print("#"*23)

# Datas needed to begin search
ip_start = input("Adresa IP de start: ")
mac_address = input("Adresa MAC (1234.abcd.ef12) : ")

# Credentials used for login
username = input("Username: ")
parola = getpass.getpass("Parola: ")

# A list that is writed with the IP addresses of the switches in which the MAC is
switch_list = []

# Begin connection using the socket presented and surpass error if Telnet port is met
def connect_to_switch(ip_add):
  '''
  :param ip_add: Represents the IP add to which we connect using SSH
  :return: False - If MAC is in loop
       True - If everything is ok
  '''
  client_socket = socket.socket()
  try:
    client_socket.connect((ip_add, 22))
    port = 22
  except socket.error:
    client_socket.connect((ip_add, 23))
    port = 23
  finally:
    time.sleep(5)
    print(ip_add+" "+str(port))
    client_socket.close()

    # Establishing SSH conexion using credentials
    if port == 22:  # SSH
      device = {'device_type': 'cisco_ios',
            'ip': ip_add,
            'username': username,
            'password': parola}
      ssh_conn = Netmiko(**device)
      comanda = "sh mac address-table | i " + mac_address
      output = ssh_conn.send_command(comanda)

      # Print Host down! for no response
      if len(output) == 0:
        print("Host down!")
        return False

      # From the last line select only the targeted info
      interfata = output.splitlines()[0].split(" ")[-1]
      # Within the class presented, search for neighbor and send command written below
      comanda = 'sh cdp nei ' + interfata + ' detail | i 192.168.122.'
      output = ssh_conn.send_command(comanda)
      # Present the IP address of the switch
      if len(output) == 0:
        print("Adresa IP Switch: " + ip_add)
        # Search for the MAC address
        comanda = 'sh mac address-table int ' + interfata
        output = ssh_conn.send_command(comanda)
        print("Adresa MAC: " + output)
        # Print informations asked for, about the location of the MAC add connection
        comanda = 'sh run int ' + interfata
        output = ssh_conn.send_command(comanda)
        print("Config interfata: \n" + output)
        return True
      # Add to file ip add if mac was not found on sw
      ip_switch = output.splitlines()[0].split(" ")[-1]

      # Do not enter loop and to not search in switches you have searched already for MAC
      if ip_switch in switch_list:
        return False
      else:
        switch_list.append(ip_switch)
        connect_to_switch(ip_switch)
```

Figure 6. Part of the MAC Search Script that connects SSHv2

IV. Conclusions

What has been accomplished in this article represents a small part of the capabilities that the usage of programming offers to network administrators.

As seen and explained in the previous studies, there were 3 libraries used for completing remote management[12]. The reasons for implementing all 3 modules within the Python scripts are to show the weak points in everyone of them, what advantages and disadvantages each of them offers. Hence, a clear conclusion can be drawn regarding which of them is the best to use for implementing integrated remote management and automation of the networks tasks that need to be done periodically.

Within the script that uses the Python TelnetLib module, it can be seen clearly the weak point of the module is clearly the lack of security. A network hacker performing Man in the middle attack, or packet sniffing may find the packets containing information about credentials used to login, IP addressing or the configuration done on the network devices when these are sent in clear text messages using the non-encrypted module TelnetLib. The second module implemented in the Python scripts was Paramiko. This module offers secure connections to the devices via encrypted SSHv2 protocol but lacks interoperability enhancements. The 3rd module implemented is a multi-vendor library developed from Paramiko and it is named Netmiko. Netmiko offer secure connections via encrypted links, by implementing SSHv2 protocol. The reasons for why Netmiko is better than Paramiko are: interoperability enhancements and a much easier way to implement a connection to the managed device [9].

As can be seen in the MAC Search script, various actions can be done using the programming in the networking domain via Python.

Concluding on the issues presented in the article, it can be appreciated that the advantage of using the TelnetLib library is given by the ease of implementation, with the disadvantages of platform compatibility and the unsafe connection to the destination [11]. The advantage of the Paramiko module to the TelnetLib library is given by the encryption of the connection, the disadvantages being the difficult implementation and incompatibility with many of the communications platforms [13]. In order to eliminate the disadvantages of the Paramiko implementation, Netmiko offers an easy implementation with the possibility of selecting the operating system of the platform to be connected, thus eliminating the compatibility issue.

Acknowledgment

This work was supported by a grant of the Ministry of Innovation and Research, UEFISCDI, project number 9SOL/12.04.2019 within PNCDI III.

References

[1] www.python.org/
[2] docs.python.org/2/library/telnetlib.html
[3] 160592857366.free.fr/joe/ebooks/tech/Wiley%20Making%20Use%20of%20Python.pdf
[4] Craig Hunt, “TCP/IP Network Administration”, O'REILLY, 2002
[5] docs.paramiko.org/en/2.4/
[6] Jay Liebowitz and David S. Prerau, “Worldwide intelligent systems: approaches to telecommunications and network management”, 1995.
[7] netmiko.readthedocs.io/en/latest
[8] pynet.twb-tech.com/blog/automation/netmiko.html
[9] www.cisco.com/c/en/us/solutions/internet-of-things/iot-management-automation.html?dtid=osscdc000283
[10] www.fortinet.com/products/management.html
[11] A. S. Tanenaum, D. J. Wetherhall, “Computer Networks 5th Edition,” 2010.
[12] M. O. Faruque Sarker, S. Washington, “Learning Python Network Programming”, 2015.
[13] G. C. Hillar, *Internet of Things with Python*, Packt Publishing, 2016.