Network Automation Using Python Threading Library

Network Topology Specifications

I have created topology using GNS3 and VMware. I am using five cisco routers, a switch and network automation docker to perform tasks using python scripting. Docker is connecting to routers via SSH connection, which is done by using paramiko library in python. Figure 1.1 is the topology diagram, used to demonstrate this project.

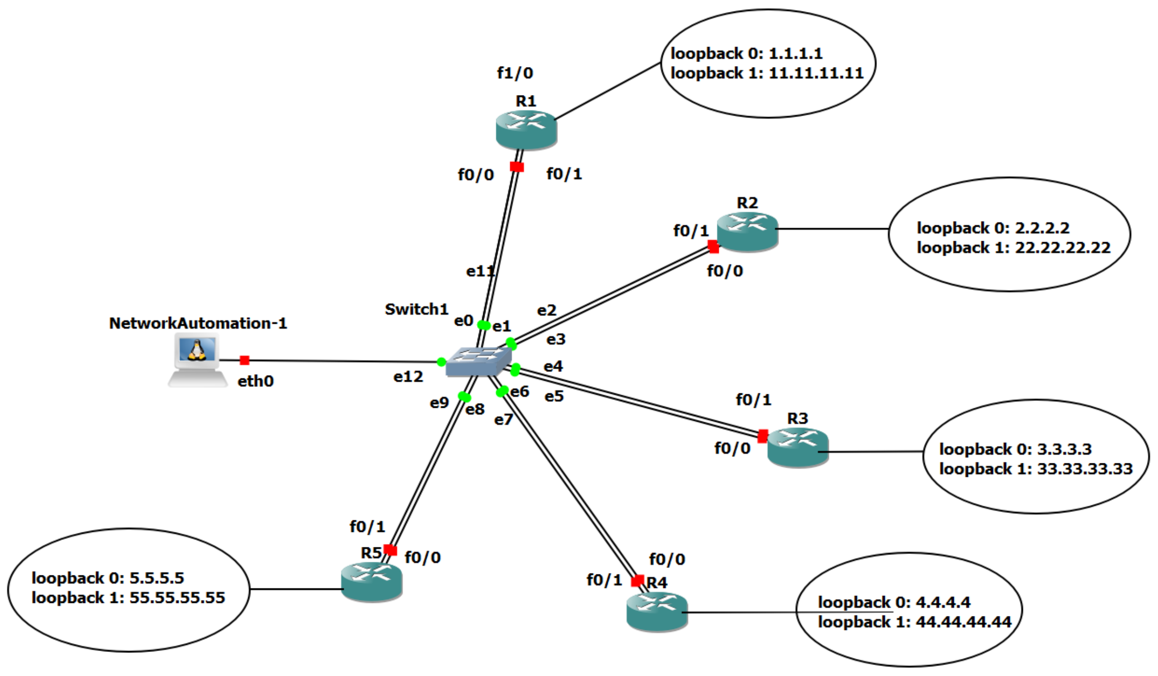


Figure 1.1 Network Topology

Each router has two loopbacks with '/24' mask and we can see IP addresses in the figure. Docker is connected to switch via e12 port of switch. We can see the network addresses and corresponding Vlan-ids in the table 1.1.

Docker is connected to e12 port of switch in Vlan 1, its default gateway is f1/0 interface of R1.

Table 1.1 IP Address Assignment

Router – interface	Switch interface	Vlan	Ip address
R1 – f0/0	E0	Vlan 15	10.0.15.1/24
R1 – f0/1	E1	Vlan 12	10.0.12.1/24
R1 – f1/0	E11	Vlan 1	10.0.100.1/24
R2 – f0/0	E2	Vlan 12	10.0.12.2/24
R2 – f0/1	E3	Vlan 23	10.0.23.2/24
R3 – f0/0	E4	Vlan 23	10.0.23.3/24
R3 – f0/1	E5	Vlan 34	10.0.34.3/24
R4 – f0/0	E6	Vlan 34	10.0.34.4/24
R4 – f0/1	E7	Vlan 45	10.0.45.4/24
R5 – f0/0	E8	Vlan 45	10.0.45.5/24
R5 – f0/1	E9	Vlan 15	10.0.15.5/24
Docker – eth0	E12	Vlan 1	10.0.100.3/24

Requirements

Firstly, we require GNS3 software which is open source. It allows us to run small topology with only a few devices on our laptop, also to those who have many devices hosted on multiple servers or even hosted in the cloud [4]. After that we have to install VMware software in order to support GNS3 VM. VMware is used for virtualization and cloud computing. We will require docker to run the python script and for centralized access. I have used 'network automation' docker, which we can download from the GNS3 – marketplace.

What is Docker?

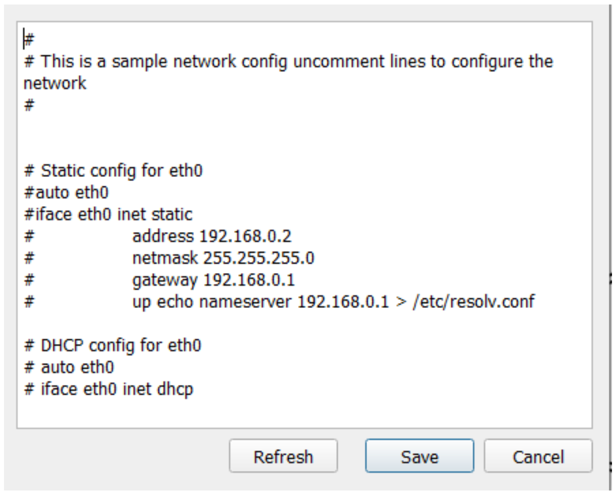
Docker simplifies and accelerates workflow, while giving developers the freedom to innovate with their choice of tools, application stacks, and deployment environments for each project [5].

Network Connectivity

We have already discussed our network topology in the introduction section. First, we will create our topology and then we will assign each port to Vlans according to table 1.1. After that, we will assign IP addresses on each router and in docker. We will use RIP as the routing protocol in order to get full connectivity. After we get full connectivity, we have to enable SSH on all routers in order to get secure remote access via docker. We will use paramiko library of python to get access. The configuration commands are provided in Appendix – A.

Docker IP Address Assignment

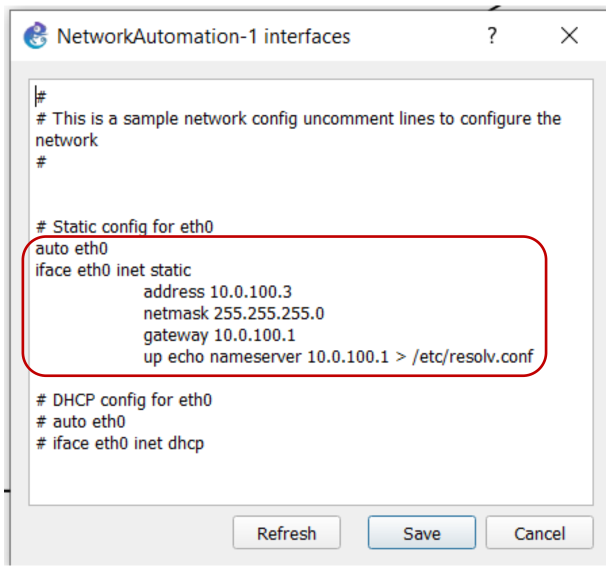
To assign IP address in docker, do right-click on the docker and then open edit-config. It will look like the figure 3.1. We can assign IP address statically or via DHCP, which are highlighted with red boxes in the figure 3.1. Here, we will assign it via static method. To do that, we have to uncomment the static config portion (remove '#' sign) and then edit the IP address, network mask and gateway address.



```
#
# This is a sample network config uncomment lines to configure the network
#


# Static config for eth0
#auto eth0
#iface eth0 inet static
#               address 192.168.0.2
#               netmask 255.255.255.0
#               gateway 192.168.0.1
#               up echo nameserver 192.168.0.1 > /etc/resolv.conf

# DHCP config for eth0
# auto eth0
# iface eth0 inet dhcp
```

Figure 3.1 Docker IP Assignment – 1

Following figure 3.2 shows the docker configuration after we have assigned IP address, which is highlighted with red box.



```
#
# This is a sample network config uncomment lines to configure the
network
#


# Static config for eth0
auto eth0
iface eth0 inet static
		address 10.0.100.3
		netmask 255.255.255.0
		gateway 10.0.100.1
		up echo nameserver 10.0.100.1 > /etc/resolv.conf

# DHCP config for eth0
# auto eth0
# iface eth0 inet dhcp
```

Figure 3.2 Docker IP Assignment – 2

Python Paramiko Library

Paramiko is python library which provides SSHv2 connectivity and provides both client and server functionality [6]. First. we have to create SSHClient object [7]. In order to get direct control, we have to pass a socket and use it to negotiate with the remote host [7].

As a client, we are responsible for authenticating using a password or private key, and checking the server's host key [7]. As a server, we are responsible for deciding which users, passwords, and keys to allow, and what kind of channels to allow [7].

Once we are finished, either side may request flow-controlled channels to the other side, which are python objects that act like sockets, but send and receive data over the encrypted session [7].

Sample Commands to get Connectivity

```
ssh_client = paramiko.SSHClient()

ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(hostname=ip_address,username=username,password=password)
```

Python Threading Library

We use threading in order to get parallelism in code execution. Threading is usually provided by operating system [8]. Threads are lighter then process and share same memory space [8]. Threads are more useful in cases which includes input/output operations. Here, threads uses shared memory space which makes it easier for one thread to get state information.

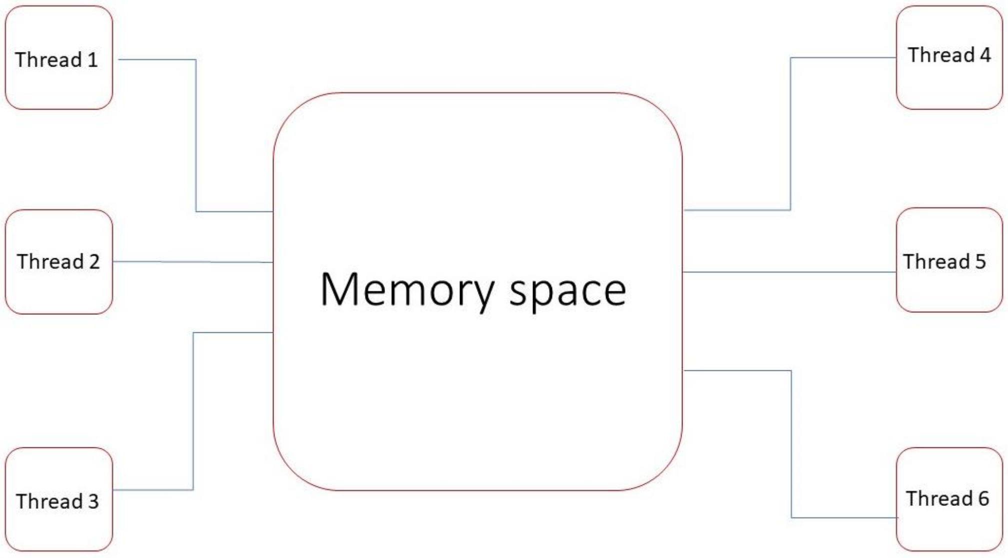


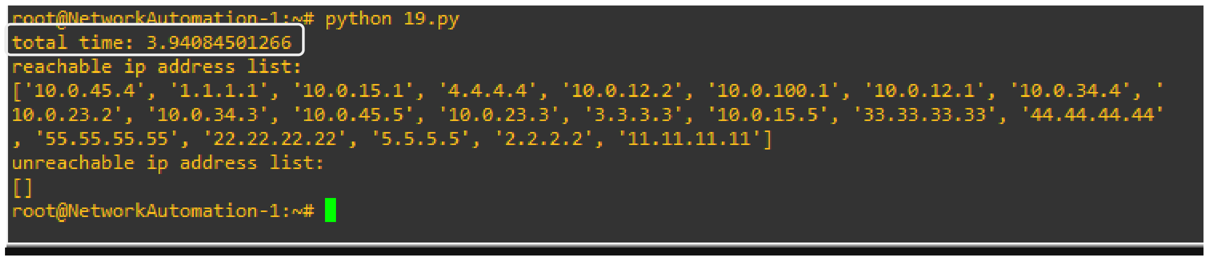
Figure 3.3 Threading Memory Space

Sample Code for Threading

```
Task_list = [ task 1, task 2, task 3]

threads = []

  for task in Task_list:

    th = threading.Thread(target = function_without_arguments, args = (function_arguments))

    th.start()
    threads.append(th)
  for th in threads:
    th.join()
```

Python Script Using Threading

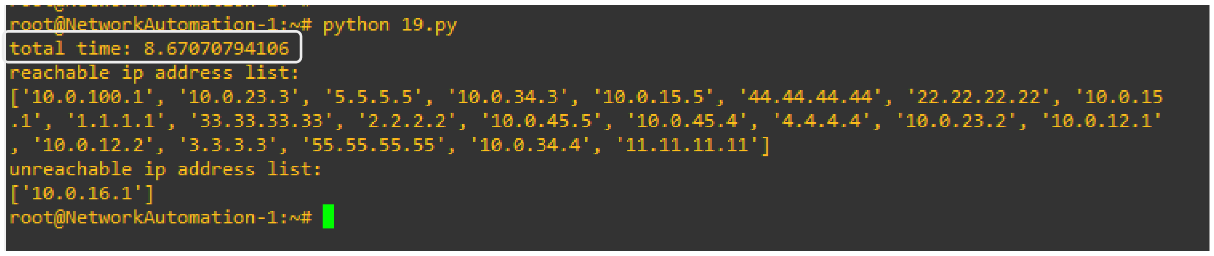
Here, I will demonstrate this concept for 0 unreachable IP, 1 unreachable IP, 3 unreachable IPs and finally for 5 unreachable IPs. We can see the script in appendix – B.



```
root@NetworkAutomation-1:~# python 19.py
total time: 3.94084501266
reachable ip address list:
['10.0.45.4', '1.1.1.1', '10.0.15.1', '4.4.4.4', '10.0.12.2', '10.0.100.1', '10.0.12.1', '10.0.34.4', '
10.0.23.2', '10.0.34.3', '10.0.45.5', '10.0.23.3', '3.3.3.3', '10.0.15.5', '33.33.33.33', '44.44.44.44'
, '55.55.55.55', '22.22.22.22', '5.5.5.5', '2.2.2.2', '11.11.11.11']
unreachable ip address list:
[]
root@NetworkAutomation-1:~#
```

Figure 4.1 Threading – All Reachable IP

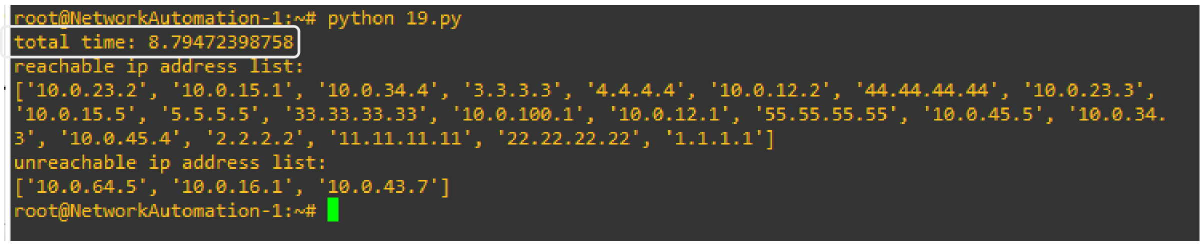
From the figure 4.1, we can see that script only takes less than 4 seconds to check the connectivity to all nodes. Here, I am demonstrating for all reachable IP addresses. In the following scenarios I will demonstrate this script for multiple unreachable nodes.



```
root@NetworkAutomation-1:~# python 19.py
total time: 8.67070794106
reachable ip address list:
['10.0.100.1', '10.0.23.3', '5.5.5.5', '10.0.34.3', '10.0.15.5', '44.44.44.44', '22.22.22.22', '10.0.15
.1', '1.1.1.1', '33.33.33.33', '2.2.2.2', '10.0.45.5', '10.0.45.4', '4.4.4.4', '10.0.23.2', '10.0.12.1'
, '10.0.12.2', '3.3.3.3', '55.55.55.55', '10.0.34.4', '11.11.11.11']
unreachable ip address list:
['10.0.16.1']
root@NetworkAutomation-1:~#
```

Figure 4.2 Threading – 1 Unreachable IP

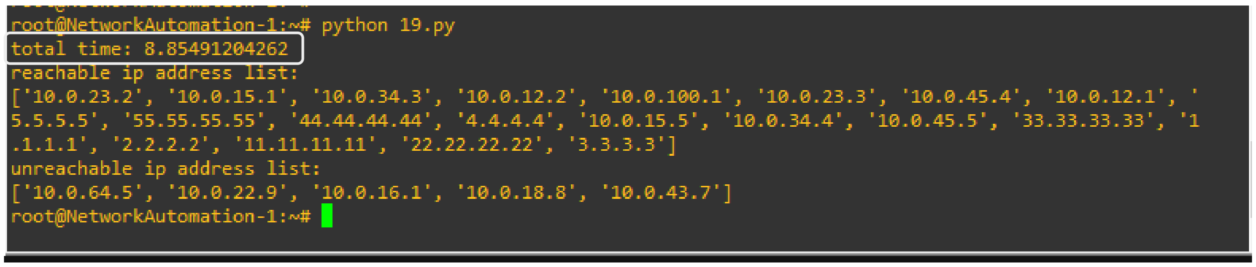
From the figure 4.2, we can see that it takes about 8.67 seconds for 1 unreachable IP address.



```
root@NetworkAutomation-1:~# python 19.py
total time: 8.79472398758
reachable ip address list:
['10.0.23.2', '10.0.15.1', '10.0.34.4', '3.3.3.3', '4.4.4.4', '10.0.12.2', '44.44.44.44', '10.0.23.3',
'10.0.15.5', '5.5.5.5', '33.33.33.33', '10.0.100.1', '10.0.12.1', '55.55.55.55', '10.0.45.5', '10.0.34.
3', '10.0.45.4', '2.2.2.2', '11.11.11.11', '22.22.22.22', '1.1.1.1']
unreachable ip address list:
['10.0.64.5', '10.0.16.1', '10.0.43.7']
root@NetworkAutomation-1:~#
```

Figure 4.3 Threading – 3 Unreachable IP

From the figure 4.3 and 4.4, we can see that it took almost same time, as scenario with 1 unreachable IP, to check connectivity.



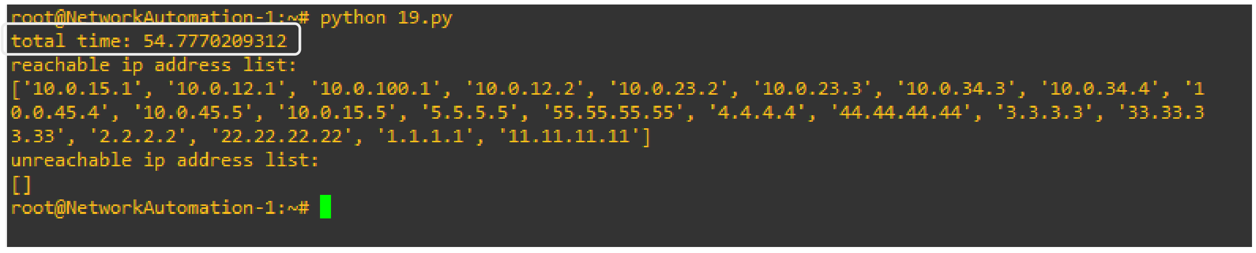
```
root@NetworkAutomation-1:~# python 19.py
total time: 8.85491204262
reachable ip address list:
['10.0.23.2', '10.0.15.1', '10.0.34.3', '10.0.12.2', '10.0.100.1', '10.0.23.3', '10.0.45.4', '10.0.12.1', '
5.5.5.5', '55.55.55.55', '44.44.44.44', '4.4.4.4', '10.0.15.5', '10.0.34.4', '10.0.45.5', '33.33.33.33', '1
.1.1.1', '2.2.2.2', '11.11.11.11', '22.22.22.22', '3.3.3.3']
unreachable ip address list:
['10.0.64.5', '10.0.22.9', '10.0.16.1', '10.0.18.8', '10.0.43.7']
root@NetworkAutomation-1:~#
```

Figure 4.4 Threading – 5 Unreachable IP

From the above scenarios, we can say that if we use threading the script will take maximum 9 seconds to check connectivity to all nodes.

Python Script Without Threading

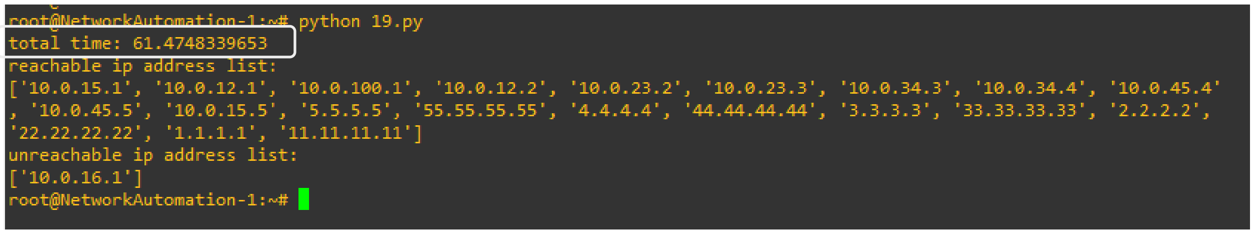
In this section, I will demonstrate the same scenarios, with 0 unreachable IP, 1 unreachable IP, 3 unreachable Ips and 5 unreachable Ips to check the time it takes to check connectivity for all nodes. We can see the script in appendix – C.



```
root@NetworkAutomation-1:~# python 19.py
total time: 54.7770209312
reachable ip address list:
['10.0.15.1', '10.0.12.1', '10.0.100.1', '10.0.12.2', '10.0.23.2', '10.0.23.3', '10.0.34.3', '10.0.34.4', '1
0.0.45.4', '10.0.45.5', '10.0.15.5', '5.5.5.5', '55.55.55.55', '4.4.4.4', '44.44.44.44', '3.3.3.3', '33.33.3
3.33', '2.2.2.2', '22.22.22.22', '1.1.1.1', '11.11.11.11']
unreachable ip address list:
[]
root@NetworkAutomation-1:~#
```

Figure 4.5 Normal Script – All Reachable IP

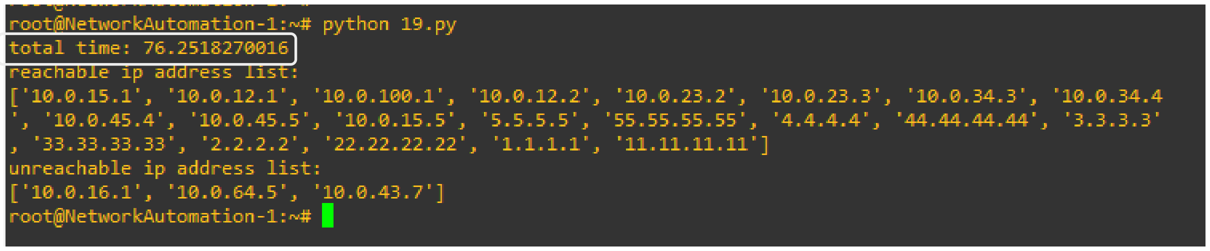
In the figure 4.5 we can see that it takes almost 55 seconds to run the script sequentially. In the following scenarios we will check time for 1 and more unreachable IP addresses.



```
root@NetworkAutomation-1:~# python 19.py
total time: 61.4748339653
reachable ip address list:
['10.0.15.1', '10.0.12.1', '10.0.100.1', '10.0.12.2', '10.0.23.2', '10.0.23.3', '10.0.34.3', '10.0.34.4', '10.0.45.4'
, '10.0.45.5', '10.0.15.5', '5.5.5.5', '55.55.55.55', '4.4.4.4', '44.44.44.44', '3.3.3.3', '33.33.33.33', '2.2.2.2',
'22.22.22.22', '1.1.1.1', '11.11.11.11']
unreachable ip address list:
['10.0.16.1']
root@NetworkAutomation-1:~#
```

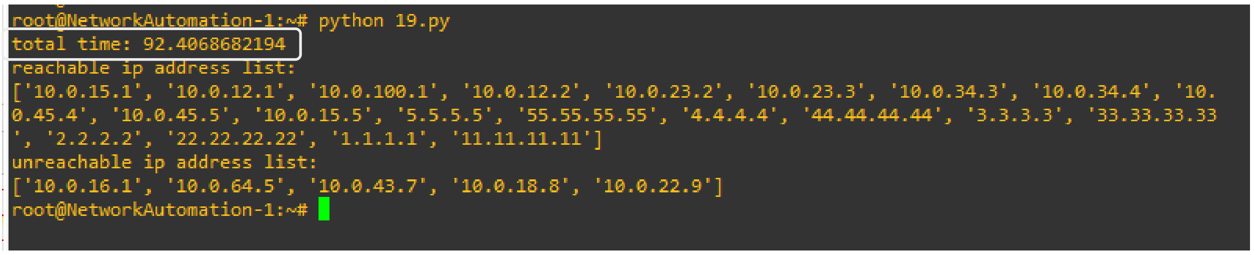
Figure 4.6 Normal Script – 1 Unreachable IP

In the figure 4.6, 4.7 and 4.8 we can see that time increases as number of unreachable IP addresses are increases.



```
root@NetworkAutomation-1:~# python 19.py
total time: 76.2518270016
reachable ip address list:
['10.0.15.1', '10.0.12.1', '10.0.100.1', '10.0.12.2', '10.0.23.2', '10.0.23.3', '10.0.34.3', '10.0.34.4
', '10.0.45.4', '10.0.45.5', '10.0.15.5', '5.5.5.5', '55.55.55.55', '4.4.4.4', '44.44.44.44', '3.3.3.3'
, '33.33.33.33', '2.2.2.2', '22.22.22.22', '1.1.1.1', '11.11.11.11']
unreachable ip address list:
['10.0.16.1', '10.0.64.5', '10.0.43.7']
root@NetworkAutomation-1:~#
```

Figure 4.7 Normal Script – 3 Unreachable IP



```
root@NetworkAutomation-1:~# python 19.py
total time: 92.4068682194
reachable ip address list:
['10.0.15.1', '10.0.12.1', '10.0.100.1', '10.0.12.2', '10.0.23.2', '10.0.23.3', '10.0.34.3', '10.0.34.4', '10.
0.45.4', '10.0.45.5', '10.0.15.5', '5.5.5.5', '55.55.55.55', '4.4.4.4', '44.44.44.44', '3.3.3.3', '33.33.33.33
', '2.2.2.2', '22.22.22.22', '1.1.1.1', '11.11.11.11']
unreachable ip address list:
['10.0.16.1', '10.0.64.5', '10.0.43.7', '10.0.18.8', '10.0.22.9']
root@NetworkAutomation-1:~#
```

Figure 4.8 Normal Script – 5 Unreachable IP

Time Comparison

The output of the above scenarios are arranged in tabular format, so that we can compare the resulting time.

Table 4.1 Output Comparison

IP address	Threading script run-time (in seconds)	Normal script run-time (in seconds)
All reachable	3.94	54.77
1 unreachable	8.67	61.47
3 unreachable	8.79	76.25
5 unreachable	8.85	92.40

REFERENCES

[1] Juniper Networks. 2020. *What Is Network Automation? - Juniper Networks*. [online] Available at: <https://www.juniper.net/us/en/products-services/what-is/network-automation/> [Accessed 17 April 2020].

[2] Edelman, J., Lowe, S. and Oswalt, M., 2018. *Network Programmability And Automation*. Sebastopol, CA: O'Reilly Media.

[3] Cisco. 2020. *What Is Network Automation?*. [online] Available at: <https://www.cisco.com/c/en/us/solutions/automation/network-automation.html> [Accessed 17 April 2020].

[4] Docs.gns3.com. 2020. *Getting Started With GNS3 - GNS3*. [online] Available at: <https://docs.gns3.com/1PvtRW5eAb8RJZ11maEYD9_aLY8kkdhgaMB0wPCz8a38/index.html> [Accessed 19 April 2020].

[5] Docker. 2020. *Why Docker? | Docker*. [online] Available at: <https://www.docker.com/why-docker> [Accessed 19 April 2020].

[6] Paramiko.org. 2020. *Welcome To Paramiko! — Paramiko Documentation*. [online] Available at: <http://www.paramiko.org/> [Accessed 19 April 2020].

[7] Docs.paramiko.org. 2020. *Welcome To Paramiko'S Documentation! — Paramiko Documentation*. [online] Available at: <http://docs.paramiko.org/en/stable/> [Accessed 19 April 2020].

[8] Toptal Engineering Blog. 2020. *Python Multithreading And Multiprocessing Tutorial*. [online] Available at: <https://www.toptal.com/python/beginners-guide-to-concurrency-and-parallelism-in-python> [Accessed 19 April 2020].

APPENDIX – A

```
Router (config) # interface <interface name>

Router (config) # hostname <name>

          (config-if) # ip address a.b.c.d <mask>

          (config-if) # no shutdown

          (config-if) # exit

          (config) # router rip

          (config-router) # version 2

          (config-router) # no auto-summary

            (config-router) # net a.b.c.d

            (config-router) # exit

          (config) # ip domain-name <name>

          (config) # crypto key generate rsa

          (use 1024 bit key)

          (config) # username <name> privilege <number> password <password>

          (config) # line vty 0 100

          (config-line) # login local

          (config-line) # transport input ssh

          (config-line) # exit
```

APPENDIX – B

```
import paramiko

import time

import threading


reachable_ip = []

unreachable_ip = []


def ping_device(host,password):


    ssh_client = paramiko.SSHClient()

    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    ssh_client.connect(hostname='10.0.12.1', username='advait', password=password)

    remote_connection = ssh_client.invoke_shell()

    cmd = 'ping ' + host + ' timeout 1 \n'

    remote_connection.send(cmd)

    time.sleep(2)

    output = remote_connection.recv(65536)

    output = output.decode()

    temp_lst = output.split(':')

    temp_lst2 = temp_lst[1].split('\n')

    if temp_lst2[1].strip() == '.':

        time.sleep(5)

        unreachable_ip.append(host)

    elif temp_lst2[1].strip() == '!!!!!':

        reachable_ip.append(host)
```

```
def create_threads(host_list, function, password):
    threads = []
    for host in host_list:
        th = threading.Thread(target = function, args = (host,password))
        th.start()
        threads.append(th)

    for th in threads:
        th.join()

f = open('ip.txt', 'r')
ip_list = []
for i in f:
    ip_list.append(i.strip())

start = time.time()

create_threads(ip_list, ping_device, 'cisco')

end = time.time()

print 'total time: ' + str(end-start)
print 'reachable ip address list: '
print reachable_ip
print 'unreachable ip address list: '
print unreachable_ip
```

APPENDIX – C

```
import paramiko

import time


reachable_ip = []

unreachable_ip = []


def ping_device(host,password):


    ssh_client = paramiko.SSHClient()

    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    ssh_client.connect(hostname='10.0.12.1', username='advait', password=password)

    remote_connection = ssh_client.invoke_shell()

    cmd = 'ping ' + host + ' timeout 1 \n'

    remote_connection.send(cmd)

    time.sleep(2)

    output = remote_connection.recv(65536)

    output = output.decode()

    temp_lst = output.split(':')

    temp_lst2 = temp_lst[1].split('\n')

    if temp_lst2[1].strip() == '.':

        time.sleep(5)
        unreachable_ip.append(host)

    elif temp_lst2[1].strip() == '!!!!!':

        reachable_ip.append(host)


f = open('ip.txt', 'r')

ip_list = []

for i in f:

    ip_list.append(i.strip())


start = time.time()


for ip in ip_list:

    ping_device(ip, ‘cisco’)


end = time.time()


print 'total time: ' + str(end-start)

print 'reachable ip address list: '

print reachable_ip

print 'unreachable ip address list: '

print unreachable_ip
```