



RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

1 Introduction

The objective of this Lab assignment is to gain a better understanding of the Internet Protocol (IP). You will learn to implement a **Tracerouter** application using ICMP request and reply messages.

For that purpose you will use a Network Emulator with a Graphical User Interface (GUI), named **C.O.R.E.**¹ that allows to create realistic virtual networks, running real kernel, switch and application code, on a single machine (in this case, a Virtual Machine), also equipped with the necessary network tools (e.g., traceroute, Network MAP-per (Nmap), whois, etc.) and analyser tools (e.g., tcpdump, Wireshark², etc.).

In this Lab you will implement a simplified version of a **Tracerouter** tool that will use ICMP messages. The functionality provided by the programs is similar to the functionality provided by the standard **traceroute/tracert** tools available in modern operating systems.

Before beginning this lab, you'll probably want to review section 3.4 of RFC 2151 <https://tools.ietf.org/html/rfc2151> to update yourself on the operation of the traceroute program. You'll also want to have RFC 791 <https://tools.ietf.org/html/rfc791> on hand as well, for a discussion of the IP protocol.

Preliminary Notes

The instructions in this document are applicable to Computers at the IST Labs.

Nevertheless, one nice feature of the software stack we are going to use is that it is portable to many platforms including **YOUR OWN** personal computers, running the following Operating Systems:

- Microsoft Windows from version 10 up
- Apple macOS from versions 10.13 'High Sierra' up
- Debian-based Linux, such as Ubuntu (recommended) from versions 12.04 'Precise' up.

It is not recommended to apply this setup to a virtual machine (nested virtualization), although possible, as the configuration requires access to a hypervisor environment (recommended Virtualbox) in the host system.

Before proceeding you should verify if you have a “clean” environment, i.e., no Virtual Machine “instances” running (using precious resources in your system), or inconsistent instances in Vagrant and Virtualbox. For that purpose run the `vagrant global-status` command and observe the results (as in the following example):

¹A Quick Tutorial of C.O.R.E. is available to download from the course website in Fenix)

²A web Manual of wireshark: https://www.wireshark.org/docs/wsug_html_chunked

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

```

:~$ vagrant global-status
id            name      provider  state    directory
-----
28fb48a      mininet   virtualbox poweroff  /Users/x/Projects/mininet
f0ccec2      web1      virtualbox running   /Users/x/Projects/multinode
f09c279      web2      virtualbox running   /Users/x/Projects/multinode

```

In the above example, you can observe that there are three Virtual Machine instances, being the first “mininet”, which is powered off, but two “web” servers still running. It is **advisable to halt VMs** if they are running, and then **clean and destroy VMs from previous Lab experiments** that are not related with this Lab, or that are not anymore needed.

Note: Avoid copying text strings from the command line examples or configurations in this document, as pasting them into your system or files may introduce/modify some characters, leading to errors or inadequate results.

2 Setting up the Experimental Environment

For this Lab experiment you should use the same project directory, named **core**, from previous Lab.

Download from the course website in Fenix the file `Tracerouter.py`, and place it in the **core/data/apps** folder as exemplified. Please ensure that you have a folder structure like the following:

```

.
|-- Vagrantfile
|-- bootstrap-core.sh
|-- data
|   |-- apps
|   |   |-- ICMPPing.py
|   |   |-- Tracerouter.py
|   |   |-- UDPPingerClient.py
|   |   |-- UDPPingerServer.py
|   |-- core
|   |   |-- configs
|   |   |-- myservices
|   |   |-- nodes.conf
|   |   |-- prefs.conf
|   |-- icons
|   |   |-- normal
|   |   |-- tiny
|   |-- output

```

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

Files named ICMPPinger.py, UDPPingerClient.py, UDPPingerServer.py might be already present in the “data/apps” folder from previous Lab. It will be in that folder that you will completing the missing code for the “Tracerouter” program.

In that **core** folder verify that the Vagrantfile contains a structure similar to the following (adapt addresses, if needed, to suit the environment in your host system):

```
## -*- mode: ruby -*-
# vi: set ft=ruby :

# CORE emulator VM
Vagrant.configure("2") do |config|

  config.ssh.insert_key = false
  config.ssh.forward_agent = true
  config.ssh.forward_x11 = true
  config.vbguest.auto_update = true
  config.vm.box_check_update = false

  # create CORE node
  config.vm.define "core" do |core_config|
    core_config.vm.box = "ubuntu/trusty64"
    core_config.vm.hostname = "core"
    core_config.vm.network "private_network", ip: "192.168.56.200"
    #core_config.vm.network "public_network", type: "dhcp"
    if Vagrant::Util::Platform.windows? then
      core_config.vm.synced_folder "data/output", "/home/vagrant/output",
        id: "vagrant-root", owner: "vagrant", group: "vagrant",
        mount_options: ["dmode=775,fmode=664"]
      .....
    else
      core_config.vm.synced_folder "data/output", "/home/vagrant/output"
      .....
    end
    core_config.vm.provider "virtualbox" do |vb|
      vb.name = "core"
      opts = ["modifyvm", :id, "--natdnshostresolver1", "on"]
      vb.customize opts
      vb.memory = "2048"
    end
    core_config.vm.provision "shell", path: "bootstrap-core.sh"
  end
end

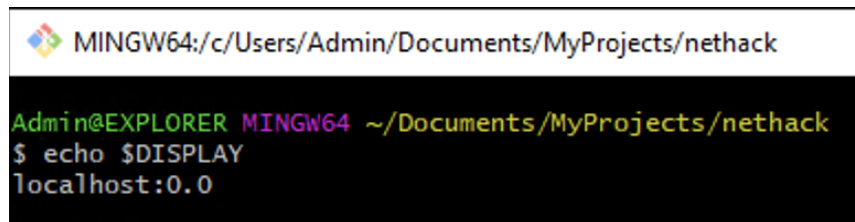
# acede-se directamente ao GUI com: vagrant ssh -c core-gui -- -X
end
```

For Microsoft Windows hosts, you need to start “**VcXsrv**” before spin off the lab environment. The “**VcXsrv**” application will start a **X-Windows** server that will forward application GUIs running inside the VM to the host system.

Verify also that you have an environment Variable with name **DISPLAY** and with

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

Variable value of localhost:0.0, as illustrated in Figure 1.



```

MINGW64:/c/Users/Admin/Documents/MyProjects/nethack
Admin@EXPLORER MINGW64 ~/Documents/MyProjects/nethack
$ echo $DISPLAY
localhost:0.0

```

Figure 1: Verify DISPLAY Variable

For the experiments in this lab you will use **Wireshark** (A network Sniffer and Protocol dissector tool, provisioned and configured in the VM) to capture the packets in the network interface in order for you to analyse them. For that purpose, you will need several Terminal windows opened. On the first window start the C.O.R.E. emulator GUI with the following command:

```
$ vagrant ssh -c core-gui -- -X
```

This command will open the C.O.R.E. GUI in your host.

In the second Terminal window start the ssh connection with the C.O.R.E. machine:

```
$ vagrant ssh
```

2.1 Using Shared Folders in Vagrant

One simple way to share information between one virtual machine, designated as **guest**, and the machine that hosts it, known as the **host**, is by mapping one folder or directory of the **host**'s file system to one folder on the **guest**'s file system.

For the C.O.R.E. emulator you have several shared folders, as you can observe in the Vagrantfile in the lines that define them, where the first path is related to the **host** and the second path refers to the **guest**.

```

core_config.vm.synced_folder "data/output",
                             "/home/vagrant/output"
core_config.vm.synced_folder "data/apps",
                             "/home/vagrant/apps"
core_config.vm.synced_folder "data/core/configs",
                             "/home/vagrant/.core/configs"
core_config.vm.synced_folder "data/core/myservices",
                             "/home/vagrant/.core/myservices"

```

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

Having this type of share, you ensure that whatever you put on the “**data/apps**” folder in your **host** will appear and be available inside the VM in the “apps” **home** folder. Similarly, whatever is produced as result of commands or programs running inside the VM can be made available on the “**data/output**” folder in your **host**.

3 The Tracerouter in Python

Recall from previous Lab, that Traceroute is a computer networking diagnostic tool which allows a user to trace the route from a host running the Traceroute program to any other host in the world.

As you have observed in previous Lab, Traceroute operates by first sending a series of datagrams with the Time-To-Live (TTL) field in the IP header set to **1** and an unlikely port number; it then sends a series of one or more datagrams towards the same destination with a TTL value of **2**; it then sends a series of datagrams towards the same destination with a TTL value of **3**; and so on. Recall that a router must decrement the TTL in each received datagram by **1** (actually, RFC 791 says that the router must decrement the TTL by at least one).

If the TTL reaches **0**, the router returns an ICMP message (of **type 11**, meaning TTL-exceeded) to the sending host.

As a result of this behavior, a datagram with a TTL of **1** (sent by the host executing traceroute) will cause the router which is **one hop away** from the sender, to send an ICMP TTL-exceeded message back to the sender; the datagram sent with a TTL of **2** will cause the router which is **two hops away** to send an ICMP message back to the sender; the datagram sent with a TTL of 3 will cause the router which is **three hops away** to send an ICMP message back to the sender; and so on. . .

In this manner, the host executing Traceroute can learn the identities of the routers between itself and destination **X** by looking at the source IP addresses in the datagrams containing the ICMP TTL-exceeded messages.

One of the datagrams will eventually make it all the way to the destination host. Because this datagram contains a UDP segment with an unlikely port number, the destination host sends a “port unreachable” ICMP message (**type 3, code 3**) back to the source host. When the source host receives this particular ICMP message, the traceroute tool knows it does not need to send additional probe packets.

Your task is to develop your own Tracerouter application in Python but using only using ICMP. Your application will use ICMP but, in order to keep it simple, it will not exactly follow the official specification in RFC 1739.

The checksum calculation and the header making are not covered in this Lab, similarly to what was done for the ICMP Pinger, with the naming of most of the variables and socket being also the same.

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

3.1 The Tracerouter Skeleton Code

The places where you need to fill in code are marked with **Fill in start** and **Fill in end**. Each place may require one or more lines of code.

Please note that the code for the `checksum(string)` function is almost identical to the one you already have in the ICMP Pinger program.

```

1  # Tracerouter.py
2
3  from socket import *
4  import os
5  import sys
6  import struct
7  import time
8  import select
9  import binascii
10
11 if sys.version_info[0] < 3:
12     raise Exception("Python 3 or a more recent version is
13         required.")
14
15 ICMP_ECHO_REQUEST = 8
16 MAX_HOPS = 30
17 TIMEOUT = 2.0
18 TRIES = 2
19 ID = os.getpid() & 0xffff
20 # Get arguments (destination hostname or IP address)
21 hostname = sys.argv[1]
22
23 # The packet that we shall send to each router along the path is
24 # the ICMP echo
25 # request packet, which is exactly what we had used in the ICMP
26 # Pinger
27 # We shall use the same packet that we built in the ICMP Pinger.
28
29 def checksum(string):
30     # get the code for this function from the ICMP Pinger Lab
31     #Fill in start
32
33     #Fill in end
34
35
36 # In this function we make the checksum of our packet
37 def build_packet():
38     # In the sendOnePing() method of the ICMP Pinger,
39     # firstly the header of the packet to be sent was made,
40     # secondly the checksum was appended to the header and then

```

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

```

36     # finally the complete packet was sent to the destination.
37     # For the Tracerouter, make the header in a similar way to
        the ICMP Pinger.
38     # Append checksum to the header.
39     # Header is type (8), code (8), checksum (16), sequence (16)
40     # Make a dummy header with a 0 checksum
41     # struct -- Interpret strings as packed binary data
42
43     #Fill in start
44
45     #Fill in end
46
47     # Do not send the packet yet, just return the final packet
        in this function.
48     # So the function ending should look like this:
49     packet = header + data
50     return packet
51
52 def get_route(hostname):
53     timeLeft = TIMEOUT
54     # Resolve the hostname via DNS (or just use the IP Address)
        and display it.
55     destAddr = gethostbyname(hostname)
56     print("Route to " + hostname + " (" + destAddr + "): " )
57
58     for ttl in range(1,MAX_HOPS):
59         for tries in range(TRIES):
60             destAddr = gethostbyname(hostname)
61             # SOCK_RAW is a powerful socket type.
62             # For more details:  http://sock-raw.org/papers/
                sock_raw
63             # Make here a raw socket named mySocket, and bind it
64             #Fill in start
65
66             #Fill in end
67
68             # setsockopt method is used to set the time-to-live
                field.
69             mySocket.setsockopt(IPPROTO_IP, IP_TTL, struct.pack(
                'I', ttl))
70             mySocket.settimeout(TIMEOUT)
71             try:
72                 d = build_packet()
73                 mySocket.sendto(d, (hostname, 0))
74                 t= time.time()

```

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

```

75         startedSelect = time.time()
76         whatReady = select.select([mySocket], [], [],
                                   timeLeft)
77         howLongInSelect = (time.time() - startedSelect)
78         if whatReady[0] == []: # Timeout
79             print(" * * * Request timed
80                   out.")
81             recvPacket, addr = mySocket.recvfrom(1024)
82             timeReceived = time.time()
83             timeLeft = timeLeft - howLongInSelect
84             if timeLeft <= 0:
85                 print(" * * * Request
86                       timed out.")
87
88     except timeout:
89         continue
90
91     else:
92         # Fetch the ICMP type and code from the received
93         packet
94         #Fill in start
95
96         #Fill in end
97
98         if types == 11:
99             bytes = struct.calcsize("d")
100             timeSent = struct.unpack("d", recvPacket[28:
101                                     28 + bytes])[0]
102             print(" %d rtt = %.0f ms %s" %(ttl, (
103                 timeReceived - t)*1000, addr[0]))
104
105         elif types == 3:
106             bytes = struct.calcsize("d")
107             timeSent = struct.unpack("d", recvPacket[28:
108                                     28 + bytes])[0]
109             print(" %d rtt = %.0f ms %s" %(ttl, (
110                 timeReceived - t)*1000, addr[0]))
111
112         elif types == 0:
113             bytes = struct.calcsize("d")
114             timeSent = struct.unpack("d", recvPacket[28:
115                                     28 + bytes])[0]
116             print(" %d rtt = %.0f ms %s" %(ttl, (
117                 timeReceived - timeSent)*1000, addr[0]))
118
119         return

```


RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

```

110
111         else:
112             print("error")
113             break
114
115         finally:
116             mySocket.close()
117
118 get_route(hostname)

```

Please note The following:

1. This program will not work for websites that block ICMP traffic;
2. You do not need to be much concerned about the code for the `checksum` function and the `build_packet` function, as it was already given in the code of the ICMP Pinger program. Just copy it to your Tracerouter program.
3. This Lab requires the use of **raw sockets**. If your program is used elsewhere, in some operating systems you may need administrator/root privileges to be able to run the program.
4. To run correctly, this program requires **Python 3**.

3.2 About the ICMP Header

The ICMP header starts after bit 160 of the IP header (unless IP options are used) as illustrated in Table 1.

Table 1: The ICMP Header Format

Bits	160-167	168-175	176-183	184-191
160	Type	Code	Checksum	
192	ID		Sequence	

- **Type** - ICMP type.
- **Code** - Subtype to the given ICMP type.
- **Checksum** - Error checking data calculated from the ICMP header + data, with value 0 for this field.
- **ID** - An ID value, should be returned in the case of echo reply.
- **Sequence** - A sequence value, should be returned in the case of echo reply.

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

3.3 About the Echo Request

The “echo request” (**ping**) is an ICMP message whose data is expected to be received back in an “echo reply” (**pong**). The host must respond to all echo requests with an “echo reply” containing the exact data received in the request message.

- **Type** must be set to **8**.
- **Code** must be set to **0**.
- The **ID** and **Sequence** Number can be used by the client to match the reply with the request that caused the reply. In practice, most Linux systems use a unique identifier for every **ping** process, and sequence number is an increasing number within that process. Microsoft Windows uses a fixed identifier, which varies between Windows versions, and a sequence number that is only reset at boot time.
- The data received by the echo request must be entirely included in the echo reply.

3.4 About the Echo Reply

The “echo reply” is an ICMP message generated in response to an “echo request”, and is mandatory for all hosts and routers.

- **Type** and **Code** must be set to **0**.
- The identifier and sequence number can be used by the client to determine which echo requests are associated with the echo replies.
- The data received in the echo request must be entirely included in the echo reply.

4 Running the Experiments – Analyzing IP

Once the C.O.R.E. machine is ready, access to the GUI (core-gui) with the command:

```
:~$ vagrant ssh -c core-gui -- -X
```

A window of the C.O.R.E. emulator GUI will appear. In the top menu, select to open a file named `wan-bgp-base.imn` which will display the network scenario as in Figure 2.

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

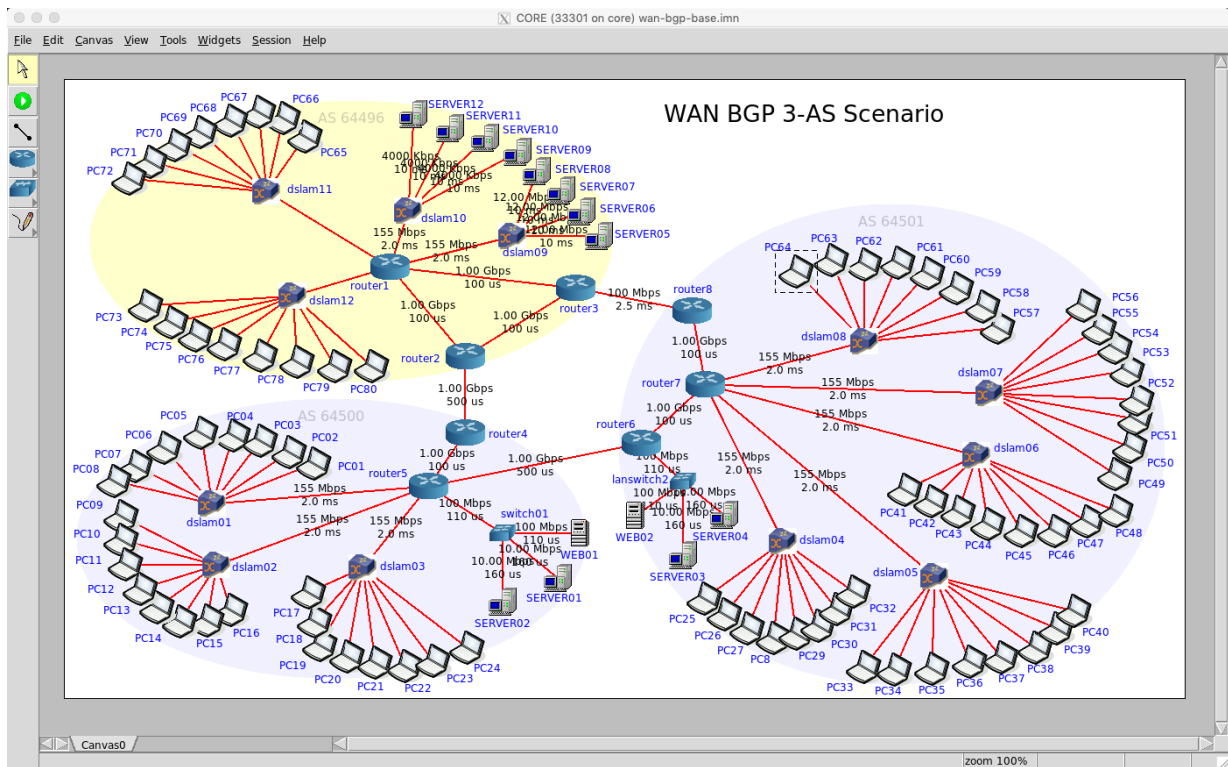


Figure 2: A complex network created in C.O.R.E.

That network simulates three interconnected “Autonomous Systems (AS)”, i.e., three Service Providers of the Internet, with their clients (PCs, Servers and Web servers) connected through links similar to “Asymmetric Digital Subscriber Lines (ADSL) or higher (Ethernet or equivalent).

The routers interconnecting the “Autonomous Systems (AS)” run BGP (Border Gateway Protocol). When the emulation starts, the node **PC01**, initiates a Ping for a few seconds with the output redirected to a file that can be analyzed in folder “data/output”.

4.1 Starting the Emulation

To start the Emulation, you just need to press the “green” Start button located on the left Toolbar. While the nodes boot up you will observe some red square brackets surrounding each node icon. As soon as a node finishes booting the square brackets will turn green for a while and then disappear (see Figure 3).

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

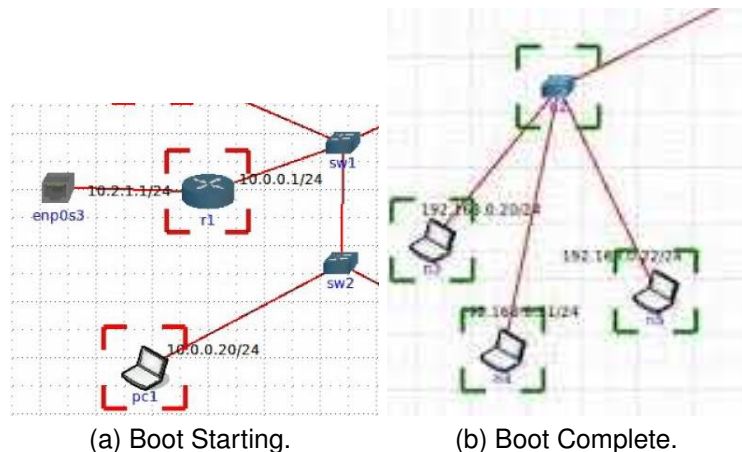


Figure 3: Starting the emulation.

In your host, verify that in folder “data/output” some new files have appeared. Those files correspond to the redirected outputs of the programs running in the respective nodes in the network.

4.2 Running the Tracerouter program

Open a “bash” interface in **PC16**. Also in **PC16** select Wireshark and start a capture in the interface **eth0**.

In the console of **PC16**, go to the “apps” folder and run the Tracerouter targeting any remote Host in the network (using its IP address), for example SERVER12, PC56, the interfaces of the line interconnecting router8 and router3:

```
root@PC16 :~$ cd /home/vagrant/apps
root@PC16:~/home/vagrant/apps$ sudo python3 Tracerouter.py IP_addr
```

Analyse the Wireshark capture and try to identify the messages exchanged between the Tracerouter program and the target Hosts.

You may save the wireshark capture to a file in the “output” folder, so that you can examine it later. Do not forget to take screenshots of the Tracerouter output while sending ICMP probes through the network to the target Hosts.

4.3 A look at the captured trace

In your trace, you should be able to see the series of ICMP messages sent by **PC16** and the ICMP TTL-exceeded messages returned to **PC16** by the intermediate routers.

Zoom in on the first ICMP packet (sent by the Tracerouter), and expand the Internet Protocol part of the packet in the packet details window.

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

You can see that the IP datagram within that packet has protocol number 01, which is the protocol number for ICMP. This means that the payload of the IP datagram is an ICMP packet.

Expanded the ICMP protocol information in the packet contents window, you can observe that this ICMP packet is of **Type 8** and **Code 0** – a so-called ICMP “echo request” packet. Also note that this ICMP packet contains a checksum, an identifier, and a sequence number.

When answering a question below, you should hand in a dissection of the packet(s) within the trace that you used to answer the question asked. To export a packet summary/data, use **File → Export Packet Dissections → As Plain text**, choose **Selected packet only**, choose **Packet summary line**, and select the minimum amount of packet detail that you need to answer the question.

1. What is the IP address of the PC host where the Tracerouter program was run? What are the IP addresses of the destination hosts (e.g., SERVER12, PC56, router8 and router3)?
2. Within the IP packet header, what is the value in the upper layer protocol field?
3. How many bytes are in the IP header? How many bytes are in the payload of the IP datagram? Explain how you determined the number of payload bytes.
4. Has this IP datagram been fragmented? Explain how you determined whether or not the datagram has been fragmented.

Next, sort the traced packets according to IP source addresses by clicking on the Source column header; a small downward pointing arrow should appear next to the word “Source”. If the arrow points up, click on the Source column header again. Select the first ICMP Echo Request message sent by **PC16**, and expand the Internet Protocol portion in the “details of selected packet header” window. In the “listing of captured packets” window, you should see all of the subsequent ICMP messages (perhaps with additional interspersed packets sent by other protocols running on the PC) below this first ICMP. Use the down arrow to move through the ICMP messages sent by that computer.

5. Which fields in the IP datagram *always* change from one datagram to the next within this series of ICMP messages sent by **PC16**?
6. Which fields stay constant? Which of the fields *must* stay constant? Which fields must change? Explain Why?
7. Describe the pattern you see in the values in the Identification field of the IP datagram.

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

Next (with the packets still sorted by source address) find the series of ICMP TTL-exceeded replies sent to **PC16** by the nearest (first hop) router.

8. What is the value in the Identification field and the TTL field?
9. Do these values remain unchanged for all of the ICMP TTL-exceeded replies sent to your computer by the nearest (first hop) router? Explain Why?

5 Finishing your Experiments

To finish the experiments, you should start by closing the opened console windows and the C.O.R.E. GUI.

In order to stop the C.O.R.E. Virtual Machine and to verify the global state of all active Vagrant environments on the system, we can issue the following commands:

```
:~$ vagrant halt
==> core: Attempting graceful shutdown of VM...
:~$ vagrant global-status
```

Confirm that the statuses of the VMs is 'powered off'. In order to prevent those instantiated machines to use resources, you can destroy them, as they can now be recreated with that simple command `vagrant up`. Confirm that there are no VMs listed.

```
:~$ vagrant destroy
core: Are you sure you want to destroy the 'core' VM? [y/N]
==> core: Destroying VM and associated drives...
:~$ vagrant global-status
```

6 Submitting the Results of the Experiments

The experiments that you execute in this LAB will either produce results that you need to report or from which you will be asked questions about the execution. In order to report the results you achieved, proceed as follows:

6.1 General Procedure

The procedure for submission is quite simple:

1. In your Google Classroom for this course, you will find an Assignment for reporting this specific Lab experiment (with a Due Date);

RC 19/20	LAB ASSIGNMENT	Guide.Part #:	3.0
Network Layer		Issue Date:	4 Nov 2019
Dissecting IP - Traceroute in Python		Due Date:	
Author: Prof. Rui Cruz		Revision:	4.1

2. The Assignment provides a Link to the TSUGI CLOUD Web Form with Exercise Questions;
3. When you are prepared with the requested materials (screen-shots, command line outputs, developed code, etc.) you will submit the items into the respective Exercise Form questions of the Assignment;
4. In the same Exercise Form of the Assignment you may be asked to comment your submissions;
5. Please note that in some types of Exercises you may also be asked to evaluate (anonymously) and provide feedback to the answers from some of your classmates (this peer-grading feedback counts for your grading).
6. When finished answering the Exercise Form, click the button **Done** in the top left of the Form; You will be returned to the Google Classroom Assignment;
7. In the Assignment, do not forget to confirm (click the button) **MARK AS DONE** or **TURN IN** when the assignment is completed;

6.2 Specific Procedure

For this LAB Assignment you will provide the Python code you have developed for the Tracerouter program. Additionally, you will also provide results from using the Wireshark Tool, as well as from using the Tracerouter program that you developed. Additionally some screenshots are also needed, as follows:

1. Submit the Python code for the Tracerouter Python program in the TSUGI Form where asked;
2. Capture a screenshot of the output of the Tracerouter running in the **PC16** of the emulated network, to submit in the TSUGI Form where asked;
3. Report (content of a small txt file) your **analysis of the IP** protocol when using the Tracerouter program with the answers to Questions 1 to 9, to submit in the TSUGI Form where asked;

WARNING. Submissions MUST BE MADE in the TSUGI CLOUD Web Form through Classroom. No other type of submission will be considered for evaluation.