

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

## 1 Introduction

The objective of this Lab assignment is to study and to gain a better understanding of the Transport Layer Protocols, i.e., User Datagram Protocol (UDP), Transmission Control Protocol (TCP) and Internet Control Message Protocol (ICMP). You will learn how to send and receive datagram packets using UDP sockets, how to set a proper socket timeout, and using ICMP request and reply messages. Throughout the lab, you will gain familiarity with **Ping applications** and their usefulness in computing network statistics such as **packet loss rate**, by learning how to implement them using UDP and ICMP.

For that purpose you will use a Network Emulator with a Graphical User Interface (GUI), named **C.O.R.E.**<sup>1</sup> 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 Mapper (Nmap), whois, etc.) and analyser tools (e.g., tcpdump, Wireshark<sup>2</sup>, etc.).

As you already have experienced in previous Labs, **Ping** is a computer network application used to test whether a particular host is reachable across an IP network. It is also used to self-test the network interface card of the computer or as a network latency test tool.

The standard **Ping** tool works by sending ICMP “**echo request**” packets to the target host and listening for ICMP “**echo reply**” replies. The “**echo reply**” is usually called a **Pong**. Ping measures the Round-trip Time (RTT), records packet loss, and prints a statistical summary of the “**echo reply**” packets received (the minimum, maximum, and the mean of the RTTs, and in some versions the standard deviation of the mean).

In this Lab you will start by studying a simple Internet **Ping server** written in Python, and then you will implement a corresponding **Ping client**. The functionality provided by these programs is similar to the functionality provided by the standard Ping tool available in modern operating systems. These Python server and client programs that you will implement, use the simpler protocol UDP (rather than ICMP) to communicate with each other, allowing a client machine to send a packet of data to a remote machine, and have the remote machine return the data back to the client unchanged (an action referred to as “echoing”).

After that first approach, you will then be able to implement an **ICMP Pinger** application written in Python, that will use ICMP messages, but, in order to keep it simple, the application will not exactly follow the official specification in RFC 1739<sup>3</sup>.

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

<sup>2</sup>A web Manual of wireshark: [https://www.wireshark.org/docs/wsug\\_html\\_chunked](https://www.wireshark.org/docs/wsug_html_chunked)

<sup>3</sup>This network tool usage was first described in RFC 1739 (<https://tools.ietf.org/html/rfc1739>), later replaced by RFC 2151 (<https://tools.ietf.org/html/rfc2151>)

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

## 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):

```

:~$ 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 any-more 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.

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

## 2 Setting up the Experimental Environment

To start, you need to create a project directory for this Lab (that will be used also in future Labs), named **core**.

Download from the course website in Fenix the file RC\_19\_20\_LAB\_02\_support\_files.zip, and uncompress its content to the **core** project folder. Please ensure that you end up with a folder structure like the following (be careful about folder names and Paths containing spaces or accented characters):

```
MySystem:~/Documents/Projects/core
$ ls -la
total xx
-rwxr-xr-x@ 1 user  staff   2.0K Oct 13 22:05 Vagrantfile
-rw-r--r--@ 1 user  staff   1.4K Oct 13 22:02 bootstrap-core.sh
drwxr-xr-x  7 user  staff  224B Aug 12 16:10 data
```

After that, confir also that the **core** project folder has a structure, and content similar to the following:

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

Files named ICMPPinger.py, UDPPingerClient.py, UDPPingerServer.py will be present in the “data/apps” folder. It will be in that folder that you will create (completing the missing code) the “ICMPPing” and the “UDPPingerClient” programs.

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

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|

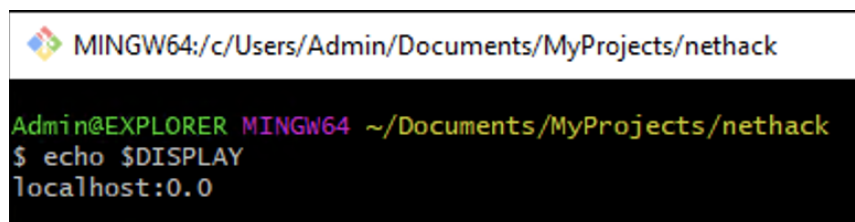
  config.ssh.insert_key = false
  config.ssh.forward_agent = true
  config.ssh.forward_x11 = true
  config.vbguest.auto_update = true
  config.vm.box_check_update = false

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

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

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 **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",

```

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

```

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

```

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 Ping Server in Python

The UDPPingerServer.py code fully implements a Ping server using UDP. You need to **run this program before** running your **Ping Client** program.

**You do not need to modify this** UDPPingerServer.py code.

In this Ping server code, 30% of the client’s packets are simulated to be lost. You should study this code carefully, as it will help you write your Ping client.

```

1  # UDPPingerServer.py
2  # We will need the following module to generate randomized lost
   # packets
3  import random
4  from socket import *
5  # Create a UDP socket
6  # Notice the use of SOCK_DGRAM for UDP packets
7  serverSocket = socket(AF_INET, SOCK_DGRAM)
8  # Assign IP address and port number to socket
9  serverSocket.bind(('', 12000))
10
11 while True:
12     # Generate random number in the range of 0 to 10
13     rand = random.randint(0, 10)
14     # Receive the client packet along with the address it is
       # coming from
15     message, address = serverSocket.recvfrom(1024)
16     # Capitalize the message from the client
17     message = message.upper()
18     # If rand is less is than 4, we consider the packet lost and
       # do not respond
19     if rand < 4:
20         continue
21     # Otherwise, the server responds
22     serverSocket.sendto(message, address)

```

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

The “UDPPingerServer” sits in an infinite loop listening for incoming UDP packets. When a packet comes in and if a randomized integer is greater than or equal to 4, the server simply capitalizes the encapsulated data and sends it back to the client.

As UDP provides applications with an unreliable transport service, messages may get lost in the network due to router queue overflows, faulty hardware or some other reasons. Because packet loss is rare or even non-existent in typical campus networks, the “UDPPingerServer” in this LAB injects artificial loss to simulate the effects of network packet loss. The server creates a variable randomized integer which determines whether a particular incoming packet is lost or not.

## 4 Developing the Ping Client in Python

Inspired by the code for the UDPPing Server, develop the code (by completing the missing lines) for the UDPPing Client.

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.

You already have the skeleton file “UDPPingClient.py” inside the “**data/apps**” folder (the file will appear inside the C.O.R.E. machine in the “apps” folder).

The Ping client **should send 10 pings** to the Ping server. Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network, or vice versa. For this reason, **the client cannot wait indefinitely for a reply** to a ping message. You should get the client to wait, up to one second, for a reply; if no reply is received within one second, the Ping client program should assume that the packet was lost during transmission across the network. It is recommended to look up the Python documentation to find out how to set the timeout value on a datagram socket. Specifically, your Ping client program should:

1. send the ping message using UDP (Note: Unlike TCP, you do not need to establish a connection first, since UDP is a connectionless protocol.);
2. print the response message from server, if any;
3. calculate and print the round trip time (RTT), in seconds, of each packet, if the server responds;
4. otherwise, print “Request timed out”.

The ping messages in this Lab are formatted in a simple way. The Ping client message is one line, consisting of ASCII characters in the following format:

```
Ping sequence_number time
```

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

where `sequence_number` starts at 1 and progresses to 10 for each successive ping message sent by the Ping client, and `time` is the time when the client sends the message.

## 5 The ICMP Pinger in Python

The `ICMPPinger` in Python application will use ICMP but, in order to keep it simple, will not exactly follow the official specification. Note that you will only need to write—i.e., complete the code of—the client side of the program, as the functionality needed on the server side is built into almost all operating systems.

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.

You should complete the `ICMPPinger` application, which skeleton code is given below, so that it sends **ping** requests to a specified host separated by approximately one second. Each message contains a payload of data that includes a timestamp. After sending each packet, the application waits, up to one second, to receive a reply. If one second goes by without a reply from the server, then the client assumes that either the **ping** packet or the **pong** packet was lost in the network (or that the server is down).

```

1  # ICMPPinger.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 ICMP_ECHO_REQUEST = 8
12
13 def checksum(string):
14     csum = 0
15     countTo = (len(string) // 2) * 2
16     count = 0
17     while count < countTo:
18         thisVal = ord(string[count+1]) * 256 + ord(string[count
19             ])
20         csum = csum + thisVal
21         csum = csum & 0xffffffff
22         count = count + 2
23     if countTo < len(string):
24         csum = csum + ord(string[len(string) - 1])

```



<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

```

24         csum = csum & 0xffffffff
25
26         csum = (csum >> 16) + (csum & 0xffff)
27         csum = csum + (csum >> 16)
28         answer = ~csum
29         answer = answer & 0xffff
30         answer = answer >> 8 | (answer << 8 & 0xff00)
31         return answer
32
33 def receiveOnePing(mySocket, ID, timeout, destAddr):
34     timeLeft = timeout
35     while 1:
36         startedSelect = time.time()
37         whatReady = select.select([mySocket], [], [], timeLeft)
38         howLongInSelect = (time.time() - startedSelect)
39         if whatReady[0] == []: # Timeout
40             return "Request timed out."
41         timeReceived = time.time()
42         recPacket, addr = mySocket.recvfrom(1024)
43
44         #Fill in start
45
46         #Fetch the ICMP header from the IP packet
47
48         #Fill in end
49
50         timeLeft = timeLeft - howLongInSelect
51         if timeLeft <= 0:
52             return "Request timed out."
53
54 def sendOnePing(mySocket, destAddr, ID):
55     # Header is type (8), code (8), checksum (16), id (16),
56     # sequence (16)
57     myChecksum = 0
58     # Make a dummy header with a 0 checksum
59     # struct -- Interpret strings as packed binary data
60     header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0,
61                          myChecksum, ID, 1)
62     data = struct.pack("d", time.time())
63     # Calculate the checksum on the data and the dummy header.
64     myChecksum = checksum(str(header + data))
65     # Get the right checksum, and put in the header
66     if sys.platform == 'darwin':
67         # Convert 16-bit integers from host to network byte
68         order

```

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

```

66         myChecksum = htons(myChecksum) & 0xffff
67     else:
68         myChecksum = htons(myChecksum)
69
70     header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0,
71                           myChecksum, ID, 1)
72     packet = header + data
73     mySocket.sendto(packet, (destAddr, 1)) # AF_INET address
74     must be tuple, not str
75     # Both LISTS and TUPLES consist of a number of objects
76     # which can be referenced by their position number within
77     the object.
78
79 def doOnePing(destAddr, timeout):
80     icmp = getprotobyname("icmp")
81     # SOCK_RAW is a powerful socket type. For more details:
82     http://sock-raw.org/papers/sock_raw
83     mySocket = socket(AF_INET, SOCK_RAW, icmp)
84     myID = os.getpid() & 0xFFFF # Return the current process i
85     sendOnePing(mySocket, destAddr, myID)
86     delay = receiveOnePing(mySocket, myID, timeout, destAddr)
87     mySocket.close()
88     return delay
89
90 def ping(host, timeout=1):
91     # timeout=1 means: If one second goes by without a reply
92     from the server,
93     # the client assumes that either the client's ping or the
94     server's pong is lost
95     dest = gethostbyname(host)
96     print("Pinging " + dest + " using Python: ")
97     print("")
98     # Send ping requests to a server separated by approximately
99     one second
100    while 1 :
101        delay = doOnePing(dest, timeout)
102        print(delay)
103        time.sleep(1) # one second
104    return delay
105
106 ping("google.com")

```

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

Please note The following:

1. In the ICMPing, the “receiveOnePing” method needs to receive the structure ICMP\_ECHO\_REPLY and fetch the information you need, such as checksum, sequence number, time to live (TTL), etc. Study the “sendOnePing” method before trying to complete the “receiveOnePing” method.
2. You do not need to be concerned about the checksum, as it is already given in the code.
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.

## 5.1 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
<b>160</b>	Type	Code	Checksum	
<b>192</b>	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.

## 5.2 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**.

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

- 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.

### 5.3 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.

## 6 Running the Experiments

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.

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

The routers interconnecting the Autonomous Systems (AS) run Border Gateway Protocol (BGP). 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”.

While not yet running the emulation, you can verify the configuration of the nodes and the links of the network by clicking with the right mouse button over one of those elements and selection the option “configure” as in Figure 3.

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

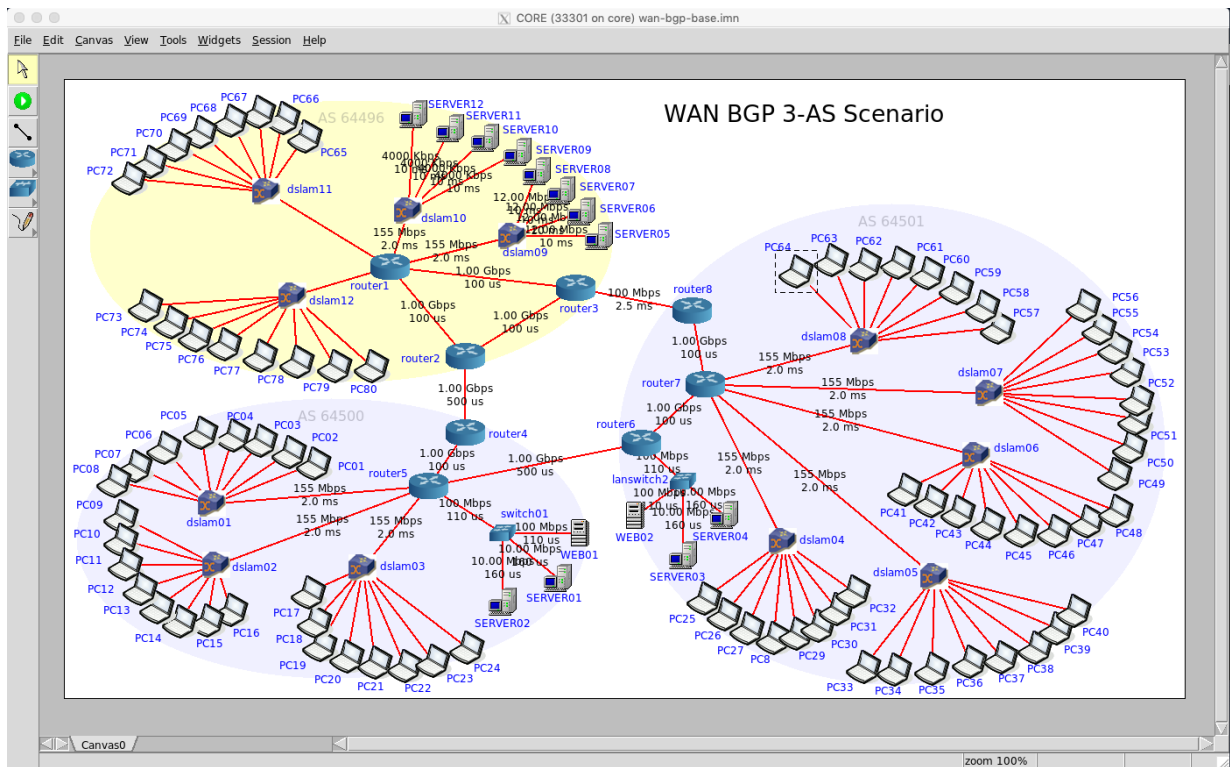


Figure 2: A complex network created in C.O.R.E.

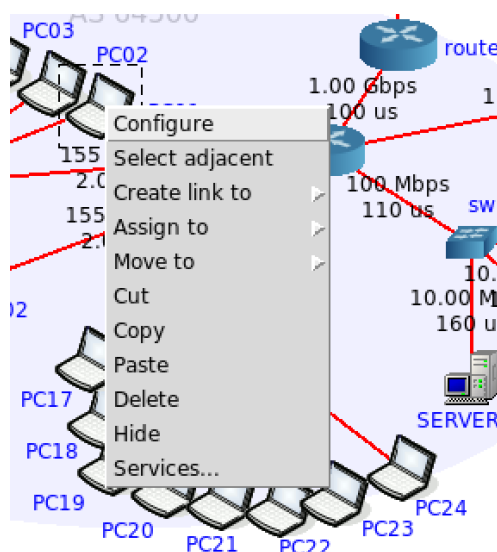


Figure 3: Configuring a node of the network

You may observe with that option, the configuration of the interfaces of a node, as illustrated in Figure 4

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

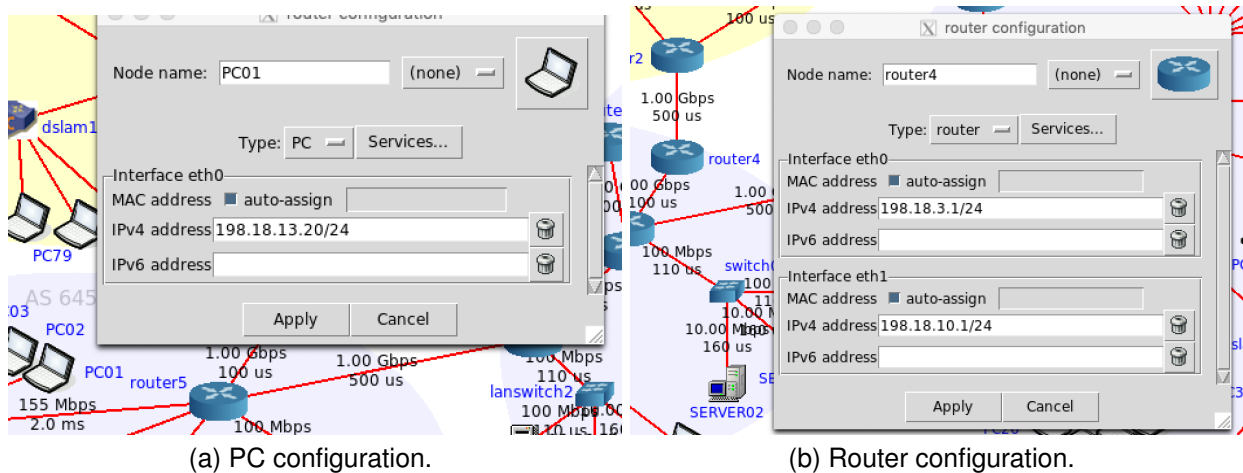


Figure 4: Configuration of the interfaces of Network Nodes

You can also define which pre-defined “services” will be available (and running) in the nodes, as well as select new ones, modify them, etc, as illustrated in Figures 5 and 6.

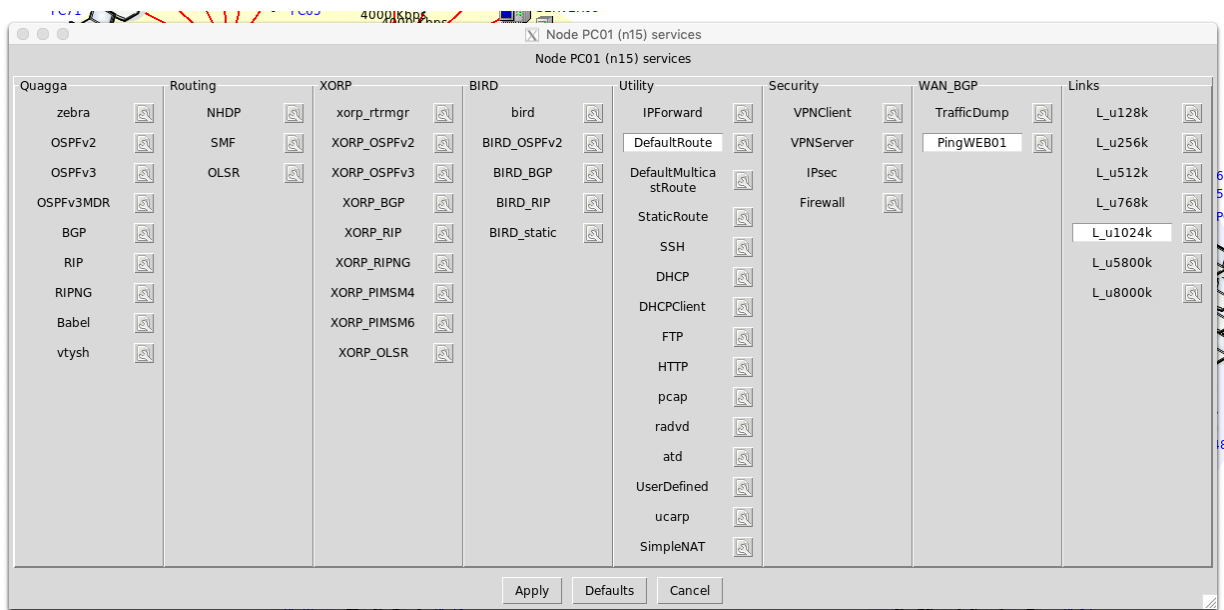


Figure 5: Selecting pre-defined “services” for runtime.

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

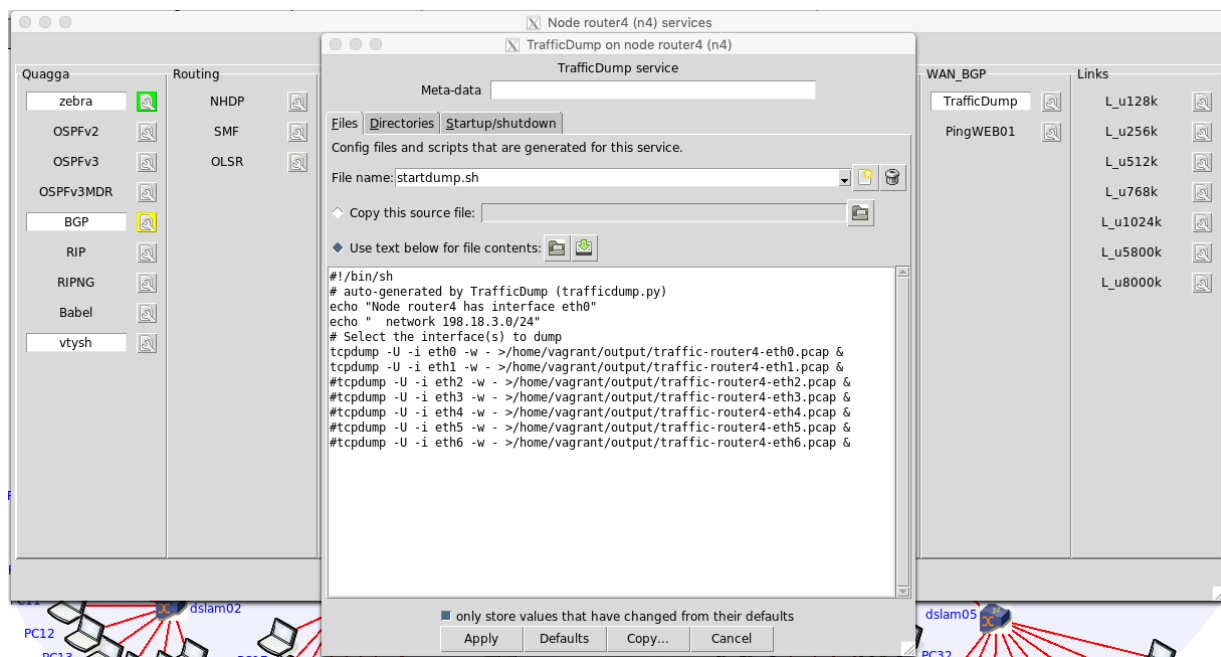


Figure 6: Modifying a pre-defined “service” (in this case, tcpdump) for runtime.  
In the figure, tcpdump commands were uncommented for interfaces “eth0” and “eth1”.

While the emulation is running, you can open console “Terminal” windows with a “bash” interface on any PC/server node, giving direct access to the Operating System (in this case, Linux), as illustrated in Figure 7. This option allows to run Linux programs and command-line tools, as for example the Ping server and Ping client that you will develop in Python.

In Figure 7 you can also observe that some links are “panited” in red with a thicker line. That effect is provided by the Throughput Widget that you can configure to show where traffic that goes over a certain bandwidth flows.

You can configure that Throughput Widget in the top menu of the C.O.R.E. GUI, as illustrated in Figure 8.

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

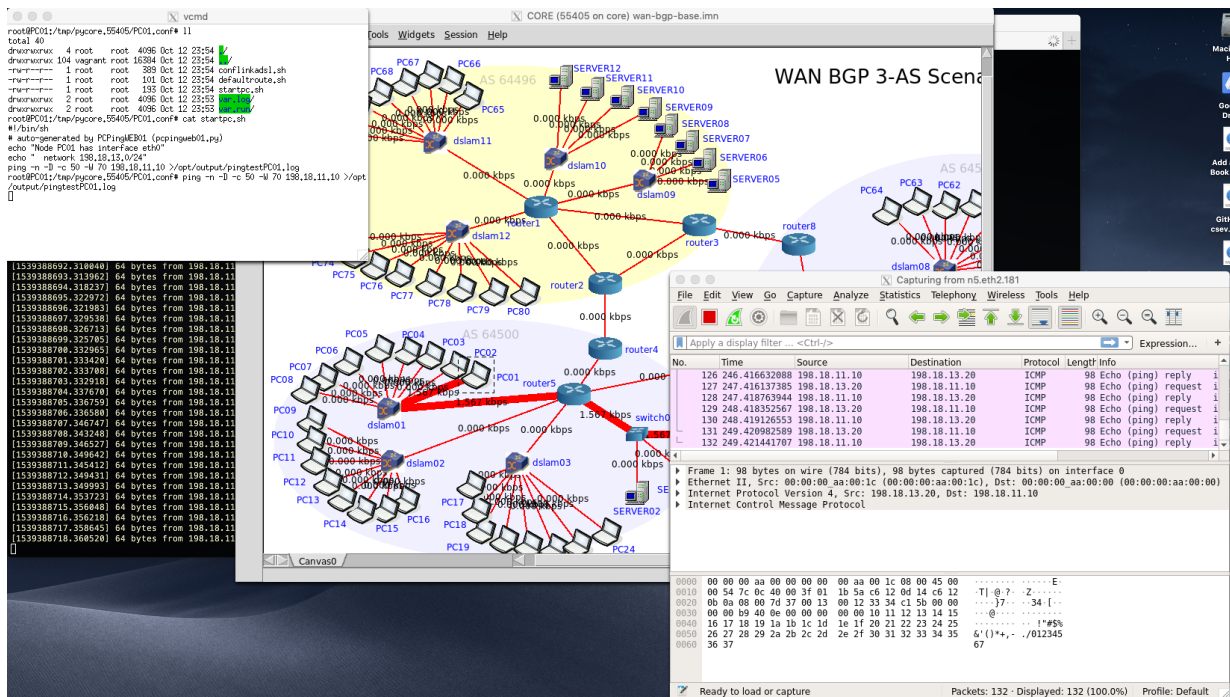


Figure 7: An emulation running with an opened “bash” console of a node, and a Wireshark window capturing packets at a router.

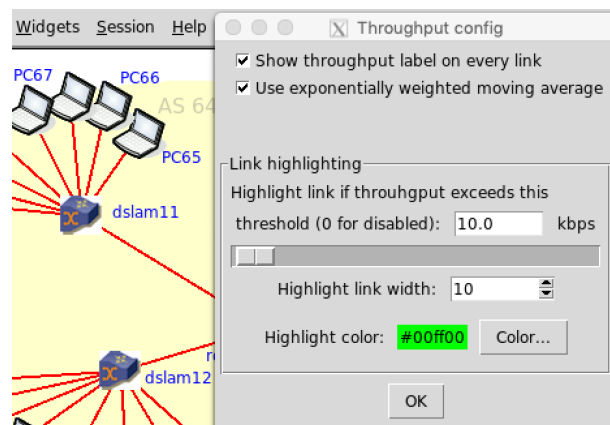


Figure 8: Configuring the Throughput Widget

## 6.1 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 9).



<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

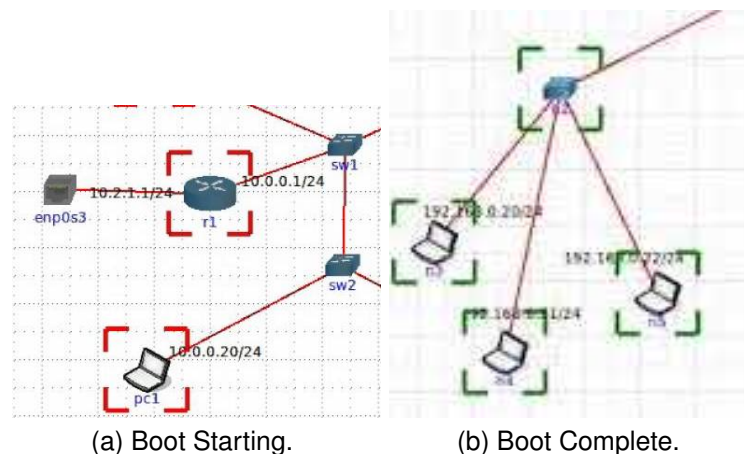


Figure 9: Starting the emulation.

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.

## 6.2 Running the UDPPing programs

Open a “bash” interface in **PC64**, **PC24** and in **SERVER05**. In **router1** select with the mouse right button the Wireshark sniffer tool and start a capture in the interface to **ds1am09**.

In the console of **SERVER05**, go to the “apps” folder and run the Ping server:

```
root@SERVER05:~$ cd /home/vagrant/apps
root@SERVER05:~/apps$ python UDPPingServer.py
```

Then, in the console of **PC64** go to the “apps” folder and run the Ping client (like the following example, replacing “IP\_srv” and “port\_srv” with the IP address and Port of the Ping server, and supplying a timeout value):

```
root@PC64:~$ cd /home/vagrant/apps
root@PC64:~/apps$ python UDPPingClient.py IP_srv port_srv timeout
```

Then, in the console of **PC24** go to the “apps” folder and also run the Ping client.

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

Analyse the Wireshark capture and try to identify the messages exchanged between the Ping Clients and the Ping Server. 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 Ping clients output while sending pings to the Ping Server.

### 6.3 EXPERIMENT 1: Analysing UDP

After stopping packet capture, set your packet filter so that Wireshark only displays the UDP packets sent and received at each PC node. Pick one of these UDP packets and expand the UDP fields in the details window of Wireshark.

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. Select one UDP packet from your trace. From this packet, determine how many fields there are in the UDP header. Name these fields.
2. By consulting the displayed information in Wireshark’s packet content field for this packet, determine the length (in bytes) of each of the UDP header fields.
3. The value in the **Length** field is the length of what? Verify your claim with your captured UDP packet.
4. What is the maximum number of bytes that can be included in a UDP payload?
5. What is the largest possible source port number?
6. What is the protocol number for UDP? Give your answer in both hexadecimal and decimal notation. To answer this question, you will need to look into the Protocol field of the IP datagram containing this UDP segment.
7. Examine a pair of UDP packets in which one of the PCs sends the first UDP packet and the second UDP packet is a reply to this first UDP packet. (Hint: for a second packet to be sent in response to a first packet, the sender of the first packet should be the destination of the second packet). Describe the relationship between the port numbers in the two packets.

### 6.4 EXPERIMENT 2: Analyzing TCP

In order to analyze some TCP packets we will use the WebServer and WebClient in Python that you implemented in LAB\_01 to get one file. You can use a TXT file of

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

your choice and put it in the “data/apps” folder, together with the “webserver.py” and “webclient.py” files.

On **SERVER05**, we will wait for requests at port 6789:

```
root@SERVER05:~/apps $ python webserver.py
Ready to serve in port 6789...
```

On one of the PCs in the network, for example (**PC64** or **PC24**) you will use the WebClient to request a file from the WebServer:

```
root@PCxx:~/apps$ python webclient.py IP_serv 6789 /file.txt
```

In Wireshark what you should see is a series of TCP and HTTP messages between the PC and the SERVER. You should also see the initial three-way handshake containing a **SYN** message.

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.

8. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection? What is it in the segment that identifies the segment as a SYN segment?
9. What is the sequence number of the SYNACK segment sent in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How this value was determined? What is it in the segment that identifies the segment as a SYNACK segment?
10. Consider the TCP segment not SYN as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments?
11. What is the length of each of the first six TCP segments?
12. Are there any retransmitted segments in the trace file? What did you check for in order to answer this question?

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

## 6.5 Running the ICMP Pinger programs

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 ICMP Pinger to target any Host in the network (by its IP address):

```
root@PC16:~$ cd /home/vagrant/apps
root@PC16:~/apps$ python ICMPPinger.py target_IP_address
```

Analyse the Wireshark capture and try to identify the messages exchanged between the ICMPPinger and the target Host.

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 ICMP Pinger output while sending pings to the target Host.

## 6.6 EXPERIMENT 3: Analysing ICMP

After stopping packet capture, set your packet filter so that Wireshark only displays the ICMP packets sent and received at the PC node.

Zoom in on the first packet (sent by the ICMPPinger). 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.

Expanding 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.

13. What is the IP address of the PC host where the ICMP Pinger was run? What is the IP address of the destination host?
14. Why is it that an ICMP packet does not have source and destination port numbers?
15. Examine one of the **ping** request packets sent by that PC. What are the ICMP type and code numbers? What other fields does that ICMP packet have? How many bytes are the checksum, sequence number and identifier fields?

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

16. Examine the corresponding ping reply packet (the **pong**). What are the ICMP type and code numbers? What other fields does that ICMP packet have? How many bytes are the checksum, sequence number and identifier fields?

## 6.7 EXPERIMENT 4: Analyzing ICMP with traceroute

In order to analyze some other types of ICMP packets we will use the **traceroute** program. On **PC16**, start a new capture with Wireshark and run the traceroute to a host in the network, for example **PC64**:

```
root@PC16:~/apps $ traceroute PC64_IP_address
```

In Wireshark what you should see is a series of UDP and ICMP messages. The output from the Traceroute program shows that for each TTL value, the source program sends **three probe packets**. Traceroute displays the RTTs for each of the probe packets, as well as the **IP address of the router** that returned the ICMP TTL-exceeded message. Note also that these ICMP error packets contains many more fields than the Ping ICMP messages.

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.

17. What is the IP address of the PC host where the Traceroute program was run? What is the IP address of the target destination host?
18. The ICMP message probes were UDP packets. What is the protocol number of the probe packets?
19. Examine the returned ICMP packet. Is this different from the ICMP ping query packets in the first part of this lab? If yes, how different it is?
20. Examine the ICMP error packet. It has more fields than the ICMP echo packet. What is included in those fields?
21. Examine the last three ICMP packets received by the source host. How are these packets different from the ICMP error packets? Why are they different?

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

## 7 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
```

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

## 8 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:

### 8.1 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);
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;

### 8.2 Specific Procedure

For this LAB Assignment you will provide the Python code you have developed for both the UDPPing server and the UDPPing Client as well as for the ICMPPing. Additionally, you will also provide results from using the Wireshark Tool, from interacting with the UDPPing Server and also from using both the ICMPPing and the Traceroute program. Additionally some screenshots are also needed, as follows:

1. **Experiment 1:**
  - (a) Submit the Python code for the UDPPing Client in the TSUGI Form where asked;

<b>RC 19/20</b>	<b>LAB ASSIGNMENT</b>	<b>Guide.Part #:</b>	2.0
Transport Layer		<b>Issue Date:</b>	14 Oct 2019
TCP, UDP, ICMP - UDP and ICMP Ping in Python		<b>Due Date:</b>	22 Oct 2019
Author: Prof. Rui Cruz		<b>Revision:</b>	1.0

- (b) Capture a screenshot of the output of the UDPPing Client to submit in the TSUGI Form where asked;
- (c) Report (content of a small text file) your **analysis of the UDP** protocol, with the answers to Questions 1 to 7, to submit in the TSUGI Form where asked;

## 2. Experiment 2:

- (a) Capture a screenshot of the output of the WebClient request to submit in the TSUGI Form where asked;
- (b) Report (content of a small text file) your **analysis of the TCP** protocol with the answers to Questions 8 to 12, to submit in the TSUGI Form where asked;

## 3. Experiment 3:

- (a) Submit the Python code for the ICMPPing in the TSUGI Form where asked;
- (b) Capture a screenshot of the output of the ICMPPing to submit in the TSUGI Form where asked;
- (c) Report (content of a small text file) your **analysis of the ICMP** protocol when using the ICMPPing with the answers to Questions 13 to 16, to submit in the TSUGI Form where asked;

## 4. Experiment 4:

- (a) Capture a screenshot of the output of the traceroute tool, to submit in the TSUGI Form where asked;
- (b) Report (content of a small text file) your **analysis of the UDP and ICMP** protocols when using the traceroute program, with the answers to Questions 17 to 21, 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.