TÉCNICO LISBOA

RC 17/18	LAB ASSIGNMENT	Number:	9
Protocols		Issue Date:	20 Nov 2017
ICMP Pinger in Python		Due Date:	30 Nov 2017

Preliminary Notes

The instructions in this document are applicable to Computers at the IST Labs, which are already configured with the required tools. 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. The instructions are suitable for machines running the following Operating Systems:

- Microsoft Windows from version 7 up to 10
- Apple macOS from versions 10.8 'Mountain Lion' up to 11.13 'High Sierra'
- Debian-based Linux, such as Ubuntu (recommended) from versions 12.04 'Precise' to 16.04 'Xenial Xerus'.

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

1 An ICMP Pinger in Python

In this lab, you will gain a better understanding of Internet Control Message Protocol (ICMP). You will learn to implement a Ping application using ICMP request and reply messages.

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 latency test. It works by sending a **Ping**, i.e., ICMP "echo request" packets to the target host, and then listening for ICMP "echo reply" replies. The "echo reply" is sometimes called a **Pong**. Ping measures the round-trip time, records packet loss, and prints a statistical summary of the echo reply packets received (the minimum, maximum, and the mean of the round-trip times and in some versions the standard deviation of the mean).

Your task is to develop your own Ping application in Python. Your application will use ICMP but, in order to keep it simple, will not exactly follow the official specification in RFC 1739. Note that you will only need to write the client side of the program, as the functionality needed on the server side is built into almost all operating systems. You should complete the Ping application 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).

2 Developing a ICMP Pinger in Python

The skeleton code for your ICMP Pinger is as follows (available to download from Fenix together with this Lab assignment). Your job is to complete the code, and then test your Pinger by sending **ping** requests to other hosts.

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.

```
from socket import *
import os
import sys
import struct
import time
import select
import binascii

ICMP_ECHO_REQUEST = 8

def checksum(string):
    csum = 0
    countTo = (len(string) // 2) * 2
    count = 0

    while count < countTo:
       thisVal = ord(string[count+1]) * 256 + ord(string[count])
       csum = csum + thisVal
       csum = csum & 0xffffffff
       count = count + 2

    if countTo < len(string):
       csum = csum + ord(string[len(string) - 1])
       csum = csum & 0xffffffff

    csum = (csum >> 16) + (csum & 0xffff)
    csum = csum + (csum >> 16)
    answer = ~csum
    answer = answer & 0xffff
    answer = answer >> 8 | (answer << 8 & 0xff00)
    return answer

def receiveOnePing(mySocket, ID, timeout, destAddr):
    timeLeft = timeout

    while 1:
        startedSelect = time.time()
        whatReady = select.select([mySocket], [], [], timeLeft)
```

```
39          howLongInSelect = (time.time() - startedSelect)
40          if whatReady[0] == []: # Timeout
41              return "Request timed out."
42
43          timeReceived = time.time()
44          recPacket, addr = mySocket.recvfrom(1024)
45
46             #Fill in start
47
48              #Fetch the ICMP header from the IP packet
49
50          #Fill in end
51          timeLeft = timeLeft - howLongInSelect
52          if timeLeft <= 0:
53              return "Request timed out."
54
55 def sendOnePing(mySocket, destAddr, ID):
56      # Header is type (8), code (8), checksum (16), id (16),
           sequence (16)
57
58      myChecksum = 0
59      # Make a dummy header with a 0 checksum
60      # struct -- Interpret strings as packed binary data
61      header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0,
           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
66      # Get the right checksum, and put in the header
67      if sys.platform == 'darwin':
68          # Convert 16-bit integers from host to network  byte
               order
69          myChecksum = htons(myChecksum) & 0xffff
70      else:
71          myChecksum = htons(myChecksum)
72
73      header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0,
           myChecksum, ID, 1)
74      packet = header + data
75
76      mySocket.sendto(packet, (destAddr, 1)) # AF_INET address
           must be tuple, not str
77      # Both LISTS and TUPLES consist of a number of objects
78      # which can be referenced by their position number within
```

```
        the object.
79
80 def doOnePing(destAddr, timeout):
81     icmp = getprotobyname("icmp")
82     # SOCK_RAW is a powerful socket type. For more details:
          http://sock-raw.org/papers/sock_raw
83
84     mySocket = socket(AF_INET, SOCK_RAW, icmp)
85
86     myID = os.getpid() & 0xFFFF  # Return the current process i
87     sendOnePing(mySocket, destAddr, myID)
88     delay = receiveOnePing(mySocket, myID, timeout, destAddr)
89
90     mySocket.close()
91     return delay
92
93 def ping(host, timeout=1):
94     # timeout=1 means: If one second goes by without a reply
          from the server,
95     # the client assumes that either the client's ping or the
          server's pong is lost
96     dest = gethostbyname(host)
97     print("Pinging " + dest + " using Python:")
98     print("")
99     # Send ping requests to a server separated by approximately
          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")
```

2.1 Additional Notes

Please consider the following when developing your code:

1. In `receiveOnePing` method, you need 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. This lab requires the use of raw sockets. In some operating systems, you may need administrator/root privileges to be able to run your Pinger program.

2.2 Information on ICMP

The ICMP header starts after bit 160 of the IP header (unless IP options are used).

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.

Echo Request: The echo request 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 Identifier 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. 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.

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.

2.3 Starting your network with C.O.R.E

Please confirm that you have X11 server running (in Linux, it should already be running, but in case you use Windows, or macOS, you need to confirm that XMing or XQuartz, respectively are running. Now that we have everything configured, start the machine with `vagrant up`, and when system is ready, open the graphical user interface of C.O.R.E.:

```
:~$ vagrant up
...
...
:~$ vagrant ssh -c core-gui -- -X
Connecting to core "core-daemon"... connected!
```

In C.O.R.E. we need to create a simple network between two machines and start your simulation. This simple network will consist of one PC and one Server connected by a router, as illustrated in Figure 1. In order to create you network use the tools available at the left toolbar.

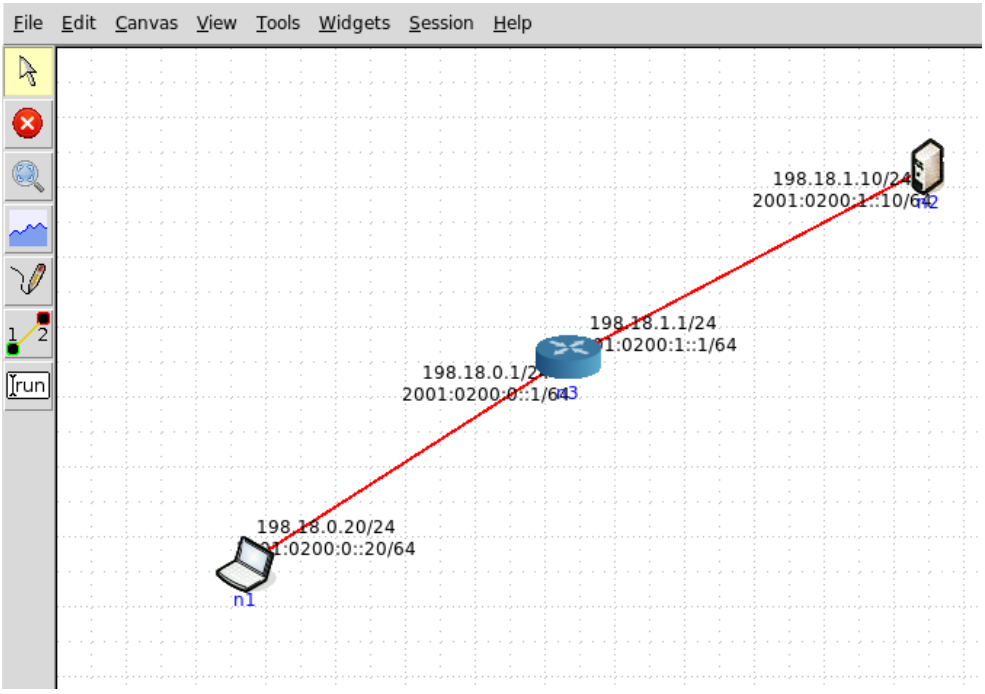

Figure 1: Simple network created on C.O.R.E.

Open a Terminal (with bash) for each one of your end systems and open a wireshark window in one of them. In the PC, go inside your shared folder and run your ICMP

Pinger:

```
:~$ cd web_shared
:~$ python pinger.py ip.of.server
```

First, test your PC by sending packets to localhost, that is, $127.0.0.1$. Then, you should see how your Pinger application communicates across the network by pinging other servers.

While using your Pinger application analyse with Wireshark the traffic being captured.

3 Finishing your Experiments

In order to stop the Virtual Machines and to verify the global state of all active Vagrant environments on the system, we can issue the following commands:

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

Confirm that the statuses of the VMs is 'powered off'.

In Lab computers, also do the following (answering "y"), in order to ensure the system is cleaned from your experiments. Note however that the configuration files remain stored in your private area in AFS:

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

Confirm that there are no VMs listed.

4 Report the results

The experiences in this lab are for evaluation. In order to report the results you achieved, proceed as follows. You will hand in the complete Pinger application python code along with the screen shots of the **pong** replies when sending **pings** to other hosts.

4.1 Lab Report

Create a Google Doc Lab Report using one of the Google Docs "Science Report" Template. In the report, explain what you did and include the relevant screen shots with

captions, the text results of commands or wireshark captures of the experiments, explaining them. Attach your Pinger Application python code.

Submission of your work:

Filename: Your report MUST be named **RC1718-LAB09-Report-IDofStudent** where "**IDofStudent**" must have the format "ist123456".

Python code: Your Python code file MUST be named **RC1718-LAB09-pinger-IDofStudent.py** where "**IDofStudent**" must have the format "ist123456".

Submission Form: Your work is submitted in a Google Form available from the Assignment published in Google Classroom.

Due Date: Please respect the due date of the assignment, considering the time limit of 23:59h.

WARNING. Incorrect submissions will not be considered for evaluation.