NEW YORK UNIVERSITY

Computer Networking Assignment 4

Lab 4: ICMP Pinger Lab

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 ICMP “echo reply” packets to the target host and 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).

Code

Below you will find the skeleton code for the client. You are to complete the skeleton code. The place where you need to fill in code is marked with #Fill in start and #Fill in end. In addition, you will need to add a few lines of code in order to calculate minimum time, average time, maximum time, and stdev time and print the results like in the operating system.

NEW YORK UNIVERSITY

Additional Notes

1. In the “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.
3. 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.
4. See the end of this programming exercise for more information on ICMP.

Testing the Pinger

First, test your client 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 servers
in different continents.

What to Hand in

Use your GitHub repository to upload the complete code for the assignment.

Example of a correct output:

- “No.no.e” is a non-valid domain, therefore, the return value (vars) of ping is ['0', '0.0', '0', '0.0']. Please make sure you use this exact value.
- “Google.co.il” is a valid domain, therefore, the return value is a list that can be created in the following way:

```
vars = [str(round(packet_min, 2)), str(round(packet_avg,
2)),str(round(packet_max, 2)), str(round(stdev(stdev_var),2))]
```

The method “ping” must return a Python list with the above values for a valid and a non-valid domain. [min,avg,max,stdev]. Return values must be in milliseconds.

NEW YORK UNIVERSITY

```
maorzeevi@Maors-MBP gradescopePinger %
maorzeevi@Maors-MBP gradescopePinger % sudo python3 solution.py
Password:
Pinging 92.242.140.21 using Python:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

--- no.no.e ping statistics ---
4 packets transmitted, 0 packets received, 100.0% packet loss
round-trip min/avg/max/stddev = 0/0.0/0/0.0 ms
Pinging 172.217.165.131 using Python:

Reply from 172.217.165.131: bytes=36 time=6.4840325ms TTL=117
Reply from 172.217.165.131: bytes=36 time=13.7128835ms TTL=117
Reply from 172.217.165.131: bytes=36 time=6.7760945ms TTL=117
Reply from 172.217.165.131: bytes=36 time=5.9728625ms TTL=117

--- google.co.il ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 5.97/8.24/13.71/3.67 ms
maorzeevi@Maors-MBP gradescopePinger %
```

Skeleton:
https://drive.google.com/file/d/1aik9efTBDvbqNmsU6DMyZWlyXDd4-HkF/view?usp=sharing

```
from socket import *
import os
import sys
import struct
import time
import select
import binascii
# Should use stdev

ICMP_ECHO_REQUEST = 8


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

```
    count = 0

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

    if countTo < len(string):
        csum += (string[len(string) - 1])
        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)
        howLongInSelect = (time.time() - startedSelect)
        if whatReady[0] == []:  # Timeout
            return "Request timed out."

        timeReceived = time.time()
        recPacket, addr = mySocket.recvfrom(1024)

        # Fill in start

        # Fetch the ICMP header from the IP packet

        # Fill in end
        timeLeft = timeLeft - howLongInSelect
        if timeLeft <= 0:
            return "Request timed out."


def sendOnePing(mySocket, destAddr, ID):
    # Header is type (8), code (8), checksum (16), id (16), sequence (16)
```

```
    myChecksum = 0
    # Make a dummy header with a 0 checksum
    # struct -- Interpret strings as packed binary data
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, ID, 1)
    data = struct.pack("d", time.time())
    # Calculate the checksum on the data and the dummy header.
    myChecksum = checksum(header + data)

    # Get the right checksum, and put in the header

    if sys.platform == 'darwin':
        # Convert 16-bit integers from host to network  byte order
        myChecksum = htons(myChecksum) & 0xffff
    else:
        myChecksum = htons(myChecksum)


    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, ID, 1)
    packet = header + data

    mySocket.sendto(packet, (destAddr, 1))  # AF_INET address must be tuple, not
str


    # Both LISTS and TUPLES consist of a number of objects
    # which can be referenced by their position number within the object.

def doOnePing(destAddr, timeout):
    icmp = getprotobyname("icmp")


    # SOCK_RAW is a powerful socket type. For more details:
http://sockraw.org/papers/sock_raw
    mySocket = socket(AF_INET, SOCK_RAW, icmp)

    myID = os.getpid() & 0xFFFF  # Return the current process i
    sendOnePing(mySocket, destAddr, myID)
    delay = receiveOnePing(mySocket, myID, timeout, destAddr)
    mySocket.close()
    return delay


def ping(host, timeout=1):
    # timeout=1 means: If one second goes by without a reply from the server,
# the client assumes that either the client's ping or the server's pong is lost
    dest = gethostbyname(host)
    print("Pinging " + dest + " using Python:")
```

```
    print("")
    # Calculate vars values and return them
    #  vars = [str(round(packet_min, 2)), str(round(packet_avg, 2)),
str(round(packet_max, 2)),str(round(stdev(stdev_var), 2))]
    # Send ping requests to a server separated by approximately one second
    for i in range(0,4):
        delay = doOnePing(dest, timeout)
        print(delay)
        time.sleep(1)  # one second

    return vars

if __name__ == '__main__':
    ping("google.co.il")
```

Internet Control Message Protocol (ICMP)

ICMP Header

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.