Socket Programming Assignment 5: ICMP Pinger

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 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.

Additional Notes

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.
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.

Skeleton Python Code for the ICMP Pinger

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

ICMP_ECHO_REQUEST = 8

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

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

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

    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':
        myChecksum = socket.htons(myChecksum) & 0xffff  # Convert 16-bit
integers from host to network byte order.
    else:
        myChecksum = socket.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 = socket.getprotobyname("icmp")

    # SOCK_RAW is a powerful socket type. For more details see:
http://sock-raw.org/papers/sock_raw
    # Fill in start
    # Create Socket here
    # Fill in end

    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 = socket.gethostbyname(host)
    print "Pinging " + dest + " using Python:"
    print ""
    # Send ping requests to a server separated by approximately one second
    while 1:
        delay = doOnePing(dest, timeout)
        print delay
        time.sleep(1)  # one second
    return delay


ping("www.poly.edu")
```