Computer Networking
Socket Programming Assignment 5: ICMP Pinger
By:
Monil Shah
mds747

Code ICMP Pinger:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Apr  9 00:32:51 2017

@author: monilshah
"""

import socket
import os
import sys
import struct
import time
import select
import binascii


ICMP_ECHO_REQUEST = 8
timeRTT = []
packageSent =0;
packageRev = 0;

def checksum(str):
    padd = 0
    countTo = (len(str) / 2) * 2
    count = 0
    while count < countTo:
        thisVal = str[count+1] * 256 + str[count]
        padd = padd + thisVal
        padd = padd & 0xffffffff
```

```
        count = count + 2
    if countTo < len(str):
        padd = padd + ord(str[len(str) - 1])
        padd = padd & 0xffffffff
    padd = (padd >> 16) + (padd & 0xffff)
    padd = padd + (padd >> 16)
    answer = ~padd
    answer = answer & 0xffff
    answer = answer >> 8 | (answer << 8 & 0xff00)
    return answer

def receiveOnePing(mySocket, ID, timeout, destAddr):
    global packageRev,timeRTT
    timeLeft = timeout
    while 1:
        startedSelect = time.time()
        whatReady = select.select([mySocket], [], [], timeLeft)
        howLongInSelect = (time.time() - startedSelect)
        if whatReady[0] == []: # Timeout
            return "0: Destination Network Unreachable,"
        timeReceived = time.time()
        recPacket, addr = mySocket.recvfrom(1024)

    #Fill in start
        #Fetch the ICMP header from the IP packet
        icmpHeader = recPacket[20:28]
        requestType, code, revChecksum, revId, revSequence =
struct.unpack('bbHHh',icmpHeader)
        if ID == revId:
            bytesInDouble = struct.calcsize('d')
            #struct.calcsize(fmt) Return the size of the struct (and hence of the string)
corresponding to the given format.
        #struct.unpack(fmt, buffer[, offset=0]) Unpack the buffer according to the
given format. The result is a tuple even if it contains exactly one item. The buffer
must contain at least the amount of data required by the format (len(buffer[offset:])
must be at least calcsize(fmt)).
```

```
            timeData = struct.unpack('d',recPacket[28:28 + bytesInDouble])[0]
            timeRTT.append(timeReceived - timeData)
            packageRev += 1
            return timeReceived - timeData
        else:
            return "ID does not match"
        #Fill in end

        timeLeft = timeLeft - howLongInSelect
        if timeLeft <= 0:
            return "1: Request timed out."

def sendOnePing(mySocket, destAddr, ID):
    global packageSent
    # 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))
    packageSent += 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
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_RAW, icmp)
    #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,
    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("RTT:",delay)
        time.sleep(1)# one second
    return delay

ping("www.google.com")
```

Output:

```
Last login: Sun Apr  9 19:09:19 on ttys001
Monils-MBP:~ monilshah$ cd Desktop/Books
Monils-MBP:Books monilshah$ sudo python icmppinger1.py
Password:
Pinging 63.117.14.87 using Python:

RTT: 0.011679887771606445
RTT: 0.008288860321044922
RTT: 0.013656139373779297
RTT: 0.014020204544067383
RTT: 0.012704849243164062
RTT: 0.013041973114013672
RTT: 0.010017871856689453
RTT: 0.008729696273803711
RTT: 0.006855964660644531
RTT: 0.011595010757446289
RTT: 0.00765681266784668
RTT: 0.007298946380615234
RTT: 0.0106658935546875
RTT: 0.00801992416381836
```

ICMP Pinger Bonus:

Code:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Apr  9 19:11:45 2017

@author: root
"""

import socket
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 = str[count+1] * 256 + str[count]
        csum = csum + thisVal
        csum = csum & 0xffffffff
```

```
        count = count + 2
    if countTo < len(str):
        csum = csum + ord(str[len(str) - 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)
        howLongInSelect = (time.time() - startedSelect)
        if whatReady[0] == []: # Timeout
                return "Request timed out."
        timeReceived = time.time()
        recPacket, addr = mySocket.recvfrom(1024)
# Get the ICMPHeader from the IP address
        icmpHeader = recPacket[20:28]
       # bytes = struct.calcsize("d")
       # raw_TTL = struct.unpack("d", recPacket[8:8 + bytes])[0]
        # binascii -- Convert between binary and ASCII
        #TTL = int(binascii.hexlify(str(raw_TTL)), 16)
        icmpType, code, checksum, packetID, sequence = struct.unpack("bbHHh",
icmpHeader)
        if packetID == ID:
            bytes = struct.calcsize("d")
            timeS = struct.unpack("d", recPacket[28:28 + bytes])[0]
            return "Reply from %s: time=%f5ms  bytes=%d" % (destAddr,
(timeReceived- timeS)*1000 ,len(recPacket))
        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 = socket.htons(myChecksum) & 0xffff
    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: http://sock-raw.org/
papers/sock_raw
    mySocket = socket.socket(socket.AF_INET, socket.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=3):
    # 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("")
    print("------Pinging " + host + " with IP " + dest + " using Python------")
    print("")
    i=0;
    #Send ping requests to a server separated by approximately one second
    while i<3:
        delay = doOnePing(dest, timeout)
        print(delay)
        time.sleep(1)# one second
        i=i+1;
    return delay
ping("www.harvard.edu") #USA
ping("www.tum.de") #germany
ping("www.ldce.ac.in") #india
ping("www.unsw.com") #australia
```

Output:

```
Last login: Sun Apr  9 19:53:27 on ttys001
Monils-MBP:~ monilshah$ sudo python icmppinger.py
Password:

------Pinging www.harvard.edu with IP 104.16.155.6 using Python------

Reply from 104.16.155.6: time=7.2660455ms  bytes=36
Reply from 104.16.155.6: time=23.0131155ms  bytes=36
Reply from 104.16.155.6: time=6.5832145ms  bytes=36

------Pinging www.tum.de with IP 129.187.255.228 using Python------

Reply from 129.187.255.228: time=101.1319165ms  bytes=36
Reply from 129.187.255.228: time=102.2470005ms  bytes=36
Reply from 129.187.255.228: time=109.4141015ms  bytes=36

------Pinging www.ldce.ac.in with IP 166.62.10.189 using Python------

Reply from 166.62.10.189: time=246.8290335ms  bytes=36
Reply from 166.62.10.189: time=312.9508505ms  bytes=36
Reply from 166.62.10.189: time=331.8638805ms  bytes=36

------Pinging www.unsw.com with IP 202.58.60.194 using Python------

Reply from 202.58.60.194: time=332.4511055ms  bytes=36
Reply from 202.58.60.194: time=326.0111815ms  bytes=36
Reply from 202.58.60.194: time=334.9587925ms  bytes=36
```

ICMP Traceroute Lab

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Apr  9 18:05:54 2017

@author: root
"""
import socket
import os
import sys
import struct
import time
import select
import binascii
ICMP_ECHO_REQUEST = 8
MAX_HOPS = 30
TIMEOUT = 5.0
TRIES = 2
# The packet that we shall send to each router along the path is the ICMP echo
# request packet, which is exactly what we had used in the ICMP ping exercise.
# We shall use the same packet that we built in the Ping exercise
def checksum(str):
# In this function we make the checksum of our packet
# hint: see icmpPing lab
    padd = 0
    pcountTo = (len(str) / 2) * 2
    pcount = 0
    while pcount < pcountTo:
        thisVal = str[pcount+1] * 256 + str[pcount]
        padd = padd + thisVal
        padd = padd & 0xffffffff
        pcount = pcount + 2
    if pcountTo < len(str):
        padd = padd + ord(str[len(str)-1])
```

```
        padd = padd & 0xffffffff
    padd = (padd >> 16) + (padd & 0xffff)
    padd = padd + (padd >> 16)
    answer = ~padd
    answer = answer & 0xffff
    answer = answer >> 8 | (answer << 8 & 0xff00)
    return answer
def build_packet():
# In the sendOnePing() method of the ICMP Ping exercise ,firstly the header of our
# packet to be sent was made, secondly the checksum was appended to the header
and
# then finally the complete packet was sent to the destination.
# Make the header in a similar way to the ping exercise.
# Append checksum to the header.
# Dont send the packet yet , just return the final packet in this function.
# So the function ending should look like this
    myChecksum = 0
    ID = os.getpid() & 0xFFFF
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, ID,
1)
    data = struct.pack("d", time.time())
    myChecksum= checksum(header + data)
    if sys.platform == 'darwin':
        myChecksum = socket.htons(myChecksum) & 0xffff
    else:
        myChecksum = socket.htons(myChecksum)
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, ID,
1)
    packet = header + data
    return packet
def get_route(hostname):
    timeLeft = TIMEOUT
    for ttl in range(1,MAX_HOPS):
        for tries in range(TRIES):
            destAddr = socket.gethostbyname(socket.gethostname())
#Fill in start
```

```
# Make a raw socket named mySocket
        icmp = socket.getprotobyname("icmp")
        mySocket= socket.socket(socket.AF_INET, socket.SOCK_RAW, icmp)
        mySocket.bind(("",12000))
#Fill in end
        mySocket.setsockopt(socket.IPPROTO_IP, socket.IP_TTL, struct.pack('I',
ttl))
        mySocket.settimeout(TIMEOUT)
        try:
            d = build_packet()
            mySocket.sendto(d, (hostname, 0))
            t= time.time()
            startedSelect = time.time()
            whatReady = select.select([mySocket], [], [], timeLeft)
            howLongInSelect = (time.time() - startedSelect)
            if whatReady[0] == []: # Timeout
                  print("* * * Request timed out.")
            recPacket, addr = mySocket.recvfrom(1024)
            timeReceived = time.time()
            timeLeft = timeLeft - howLongInSelect
            if timeLeft <= 0:
                print (" * * * Request timed out.")
        except socket.timeout:
            continue
        else:
            #Fill in start
            header = recPacket[20:28]
            type, code, checksum, p_id, sequence = struct.unpack("bbHHh", header)
            # Fetch the icmp type from the IP packet
            #Fill in end
            if type == 11:
                bytes = struct.calcsize("d")
                timeSent = struct.unpack("d", recPacket[28:28 + bytes])[0]
                print (" %d rtt=%.0f ms %s" % (ttl,(timeReceived -t)*1000, addr[0]))
            elif type == 3:
                bytes = struct.calcsize("d")
```

```
            timeSent = struct.unpack("d", recPacket[28:28 + bytes])[0]
            print (" %d rtt=%.0f ms %s" % (ttl,(timeReceived -t)*1000, addr[0]))
        elif type == 0:
            bytes = struct.calcsize("d")
            timeSent = struct.unpack("d", recPacket[28:28 +bytes])[0]
            print (" %d rtt=%.0f ms %s" % (ttl,(timeReceived -
timeSent)*1000,addr[0]))
            return
        else:
            print( "error")
        break
    finally:
        mySocket.close()
print("---www.google.com---")
get_route("www.google.com")
print("---www.twitter.com---")
get_route("www.twitter.com")
print("---www.facebook.com---")
get_route("www.facebook.com")
print("---www.yahoo.com---")
get_route("www.yahoo.com")
```

Output:

```
Monils-MBP:Desktop monilshah$ sudo python icmp.py
---www.google.com---
 1 rtt=1 ms 192.168.1.1
---www.twitter.com---
 1 rtt=1 ms 192.168.1.1
---www.facebook.com---
 1 rtt=1 ms 192.168.1.1
---www.yahoo.com---
 1 rtt=1 ms 192.168.1.1
```