Socket Prog.3: ICMP Pinger - INET

bg1573@nyu.edu

Gao, Bo

Client.py is to have a ping function to ping the packed ICMP package exported.

Client.py must return a list of tuples. Precisely, the type of the list should be [(float, (integer, integer, integer, integer, integer, double))]. The first element of the tuple should be a float, and should be the delay of the ping in milliseconds. The second element of the tuple should be a 6-tuple, in which each element corresponds to an ICMP field from the pong packet (response) from the server. In order, they should be (type, code, checksum, ID, sequence number, data).

`client.py`

```
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 = (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 (None, None)
        timeReceived = time.time()
        recPacket, addr = mySocket.recvfrom(1024)
        # Fill in start
        # Fetch the ICMP header from the IP packet
        icmpHeader = recPacket[20:28]
        # check size
        pktFormat = 'bbHHhd'
        pktSizeWithData = struct.calcsize(pktFormat)

        pktFormat = 'bbHHh'
        pktSizeHeader = struct.calcsize(pktFormat)
        # print("pktSize", pktSize)
        icmpType, code, mychecksum, packetID, sequence = struct.unpack("bbHHh",
icmpHeader)
        data = struct.unpack("d", recPacket[28: 28 + pktSizeWithData - pktSizeHeader])

        # Precisely, the type of the list should be
        # [(float, (integer, integer, integer, integer, integer, double))].
        # The first element of the tuple should be a float,
        # and should be the delay of the ping in milliseconds. The second element of the
        # tuple should be a 6-tuple, in which each element corresponds to an ICMP field
from the pong
        # packet (response) from the server.
        # In order, they should be (type, code, checksum, ID, sequence, number, data).
        if icmpType == 0 and packetID == ID:
            bytesInDouble = struct.calcsize("d")
            timeSent = struct.unpack("d", recPacket[28:28 + bytesInDouble])[0]
            rtt = (timeReceived - timeSent) * 1000
            return rtt, (icmpType, code, mychecksum, ID, sequence, data[0])

        # Fill in end
        timeLeft = timeLeft - howLongInSelect
        if timeLeft <= 0:
```

```
            print("request timed out")
            return None, None


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)
    result = receiveOnePing(mySocket, myID, timeout, destAddr)
    mySocket.close()
    return result


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)
    resps = []
    print("Pinging " + dest + " using Python:")
    print("")
```

```
    # Calculate vars values and return them
    # Send ping requests to a server separated by approximately one second
    for i in range(0, 5):
        result = doOnePing(dest, timeout)
        resps.append(result)
        print(result)
        time.sleep(1)  # one second
    return resps


if __name__ == '__main__':
    ping("127.0.0.1")
```

Output, we saw the tuple logged

```
Pinging 142.250.217.99 using Python:

(8.593559265136719, (0, 0, 64512, 19, 1, 1650878215.2093914))
(8.626937866210938, (0, 0, 23287, 19, 1, 1650878216.219205))
(8.565425872802734, (0, 0, 47653, 19, 1, 1650878217.229005))
(8.565187454223633, (0, 0, 6659, 19, 1, 1650878218.2387636))
(8.702516555786133, (0, 0, 31187, 19, 1, 1650878219.2485251))
Received from server: [(0, 0, 64512, 19, 1, 1650878215.2093914), (0, 0, 23287, 19, 1,
1650878216.219205), (0, 0, 47653, 19, 1, 1650878217.229005), (0, 0, 6659, 19, 1,
1650878218.2387636), (0, 0, 31187, 19, 1, 1650878219.2485251)]
You logged: [(8.593559265136719, (0, 0, 64512, 19, 1, 1650878215.2093914)),
(8.626937866210938, (0, 0, 23287, 19, 1, 1650878216.219205)), (8.565425872802734, (0, 0,
47653, 19, 1, 1650878217.229005)), (8.565187454223633, (0, 0, 6659, 19, 1,
1650878218.2387636)), (8.702516555786133, (0, 0, 31187, 19, 1, 1650878219.2485251))]
```

Optional Ex1

Currently, the program calculates the **round-trip** time for each packet and prints it out individually. Modify this to correspond to the way the standard ping program works. You will need to report the **minimum, maximum, and average RTTs at the end of all pings from the client**. In addition, calculate the packet loss rate (in percentage).

Code, in `icmpClientCalRTT.py` in the repo

the key part of calculate min, max, and ave RTT is in `receiveOnePing`

```
def receiveOnePing(mySocket, ID, timeout, destAddr):
    global rtt_min, rtt_max, rtt_sum, rtt_cnt
    timeLeft = timeout
    while 1:
        startedSelect = time.time()
        whatReady = select.select([mySocket], [], [], timeLeft)
        howLongInSelect = (time.time() - startedSelect)
        if whatReady[0] == []:  # Timeout
            return (None, None)
        timeReceived = time.time()
        recPacket, addr = mySocket.recvfrom(1024)
        # Fill in start
        # Fetch the ICMP header from the IP packet
        icmpHeader = recPacket[20:28]
        # check size
        pktFormat = 'bbHHhd'
        pktSizeWithData = struct.calcsize(pktFormat)

        pktFormat = 'bbHHh'
        pktSizeHeader = struct.calcsize(pktFormat)
        # print("pktSize", pktSize)
        icmpType, code, mychecksum, packetID, sequence = struct.unpack("bbHHh",
icmpHeader)
        data = struct.unpack("d", recPacket[28: 28 + pktSizeWithData - pktSizeHeader])

        # Precisely, the type of the list should be
        # [(float, (integer, integer, integer, integer, integer, double))].
        # The first element of the tuple should be a float,
        # and should be the delay of the ping in milliseconds. The second element of the
        # tuple should be a 6-tuple, in which each element corresponds to an ICMP field
from the pong
        # packet (response) from the server.
        # In order, they should be (type, code, checksum, ID, sequence, number, data).
        if icmpType == 0 and packetID == ID:
            bytesInDouble = struct.calcsize("d")
            timeSent = struct.unpack("d", recPacket[28:28 + bytesInDouble])[0]
            rtt = (timeReceived - timeSent) * 1000

            rtt_cnt += 1
            rtt_sum += rtt
            rtt_min = min(rtt_min, rtt)
            rtt_max = max(rtt_max, rtt)
```

```
            return rtt, (icmpType, code, mychecksum, ID, sequence, data[0])

        # Fill in end
        timeLeft = timeLeft - howLongInSelect
        if timeLeft <= 0:
            print("request timed out")
            return None, None
```

Output

we see `round-trip min/avg/max` and `loss rate` printed in output

```
WebLabs git/master*
(base) ❯ sudo python icmpClientCalRTT.py
Pinging 172.217.14.227 using Python:

(10.818719863891602, (0, 0, 53497, 62540, 1, 1650942501.7005372))
(8.665084838867188, (0, 0, 1920, 62540, 1, 1650942502.7128189))
(8.78000259399414, (0, 0, 13414, 62540, 1, 1650942503.725688))
(15.328168869018555, (0, 0, 24703, 62540, 1, 1650942504.738606))
(10.338068008422852, (0, 0, 11720, 62540, 1, 1650942505.7573109))
Ping lost rate calculation:
        package: sent = 5, receive = 5 lost = 0 ( 50 % lost rate)
--- google.co.il ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max 8.665/10.786/15.328 ms
```

Optional Ex2

Your program can only detect timeouts in receiving ICMP echo responses. Modify the Pinger program to parse the ICMP response error codes and display the corresponding error results to the user. Examples of ICMP response error codes are 0: Destination Network Unreachable, 1: Destination Host Unreachable.

Code

implemented in `icmpClientWithErrorCode.py` , added icmp error code at the head

```
TIMEOUT = -1  # Value returned if ping times out.
TYPE_ECHO_REPLY = 0  # ICMP type code for echo reply messages.
ICMP_ECHO_REQUEST = 8  # ICMP type code for echo request messages.
TYPE_DESTINATION_UNREACHABLE = 3  # ICMP type code for destination unreachable messages.

CODE_NETWORK_UNREACHABLE = 0  # ICMP subtype code for network unreachable messages.
CODE_HOST_UNREACHABLE = 1  # ICMP subtype code for host unreachable messages.
CODE_NETWORK_UNKNOWN = 7  # ICMP subtype code for network unknown messages.
CODE_HOST_UNKNOWN = 9 # ICMP subtype code for host unknown messages.
```

added parse `code` and `icmpType` in `receiveOnePing()`

```
def receiveOnePing(mySocket, ID, timeout, destAddr):
    timeLeft = timeout
    while True:
        startedSelect = time.time()
        whatReady = select.select([mySocket], [], [], timeLeft)
        howLongInSelect = (time.time() - startedSelect)
        if whatReady[0] == []:  # Timeout
            print("Destination Network Unreachable")
            return None, None

        timeReceived = time.time()
        recPacket, addr = mySocket.recvfrom(1024)
        # Fill in start
        # Fetch the ICMP header from the IP packet
        icmpHeader = recPacket[20:28]
        # check size
        pktFormat = 'bbHHhd'
        pktSizeWithData = struct.calcsize(pktFormat)

        pktFormat = 'bbHHh'
        pktSizeHeader = struct.calcsize(pktFormat)
        # print("pktSize", pktSize)
        icmpType, code, mychecksum, packetID, sequence = struct.unpack("bbHHh",
icmpHeader)
        data = struct.unpack("d", recPacket[28: 28 + pktSizeWithData - pktSizeHeader])

        # Precisely, the type of the list should be
        # [(float, (integer, integer, integer, integer, integer, double))].
        # The first element of the tuple should be a float,
        # and should be the delay of the ping in milliseconds. The second element of the
        # tuple should be a 6-tuple, in which each element corresponds to an ICMP field
from the pong
```

```
        # packet (response) from the server.
        # In order, they should be (type, code, checksum, ID, sequence, number, data).

        if packetID == ID and icmpType != ICMP_ECHO_REQUEST:  # If echo reply was
received. Return time of receipt
            bytesInDouble = struct.calcsize("d")
            timeSent = struct.unpack("d", recPacket[28:28 + bytesInDouble])[0]
            rtt = (timeReceived - timeSent) * 1000
            return rtt, (icmpType, code, mychecksum, ID, sequence, data[0])
            # If ID matches with sent ICMP packet ID then Check error type and return
respective info
        elif icmpType == TYPE_DESTINATION_UNREACHABLE:
            if code == CODE_NETWORK_UNREACHABLE:
                print("Destination Unreachable - Network Unreachable.")
                return None
            elif code == CODE_HOST_UNREACHABLE:
                print("Destination Unreachable - Host Unreachable.")
                return None

        # Fill in end
        timeLeft = timeLeft - howLongInSelect
        if timeLeft <= 0:
            print("request timed out")
            return None, None
```

Test ping a a non routable address

```
WebLabs git/master*  124s
(base) ❯ sudo python icmpClientWithErrorCode.py
Pinging 216.58.211.164 using Python:

Destination Unreachable - Host Unreachable.
(None, None)
Destination Unreachable - Host Unreachable.
(None, None)
Destination Unreachable - Host Unreachable.
(None, None)
Destination Unreachable - Host Unreachable.
(None, None)
```