NEW YORK UNIVERSITY

Computer Networking – Assignment 5

Lab 5: ICMP Traceroute Lab

In this lab you will learn how to implement a traceroute application using ICMP request and reply messages. The checksum and header making are not covered in this lab, **refer to the ICMP ping lab** for that purpose, the naming of most of the variables and socket is also the same.

Traceroute is a computer networking diagnostic tool which allows a user to trace the route from a host running the traceroute program to any other host in the world. Traceroute is implemented with ICMP messages. It works by sending ICMP echo (ICMP type '8') messages to the same destination with increasing value of the time-to-live (TTL) field. The routers along the traceroute path return ICMP Time Exceeded (ICMP type '11') when the TTL field becomes zero. The final destination sends an ICMP reply (ICMP type '0') messages on receiving the ICMP echo request. The IP addresses of the routers which send replies can be extracted from the received packets. The round-trip time between the sending host and a router is determined by setting a timer at the sending host.

Your task is to develop your own Traceroute application in python using ICMP. Your application will use ICMP but, in order to keep it simple, will not exactly follow the official specification in RFC 1739.

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. You do not need to be concerned about the checksum, as it is already given in the assignment skeleton code.
2. This lab requires the use of raw sockets. In some operating systems (e.g. MacOS, Windows), you may need administrator/root privileges to be able to run your Traceroute program.
3. Local testing may require you to turn your firewall or antivirus software off to allow the messages to be sent and received. However, Gradescope is not impacted by this.
4. See the end of Lab 4 'ICMP Pinger' programming exercise for more information on ICMP.

NEW YORK UNIVERSITY

What to Hand in

Use your GitHub repository to upload the complete code for the assignment. **The name of the file you submit should be "solution.py".**

Testing the Pinger

Test your client by running your code to trace google.com or bing.com. Your output should return a list and meet the acceptance criteria format provided below.

Output Requirements (Acceptance Criteria)

Your code must produce the traceroute output in the format provided below for Gradescope to verify your code is working correctly.

1. Your trace must collect hop number, roundtrip time (rtt), host ip, and the hostname. If a hostname is not available for a host, you should provide an explicit hostname as "hostname not returnable". Also, if a host is timing out (not responding), you must record this in your trace list item with the text "Request timed out". Example provided below:

```
Example:    1    12ms   10.10.111.10 hop1.com
            2    30ms   10.10.112.10 hostname not returnable
            3    *      Request timed out
            4    5ms    10.10.110.1  target-host.com
```

2. Your get_route() function must return a nested list with trace output. That is, each trace row must be a list that includes the trace results as individual items in the list, which is also inside an overall traceroute list. Example provided below:

   Example: [['1', '12ms', '10.10.111.10', 'hop1.com'], ['2', '30ms', '10.10.112.10',
   'hostname not returnable'], ['3', '*', 'Request timed out'], ['4', '5ms', '10.10.110.1', 'target-host.com']]

   Note: Your output will be parsed to verify that it includes the relevant information, so if you do not provide the output of your function in a nested list, your solution will not work correctly and you will not receive points. Also, note that the example lists include all data as strings.

NEW YORK UNIVERSITY

Skeleton Code

Code is available below and on Google Drive: https://docs.google.com/document/d/17WddCRB1KFY6FwzGEW4oGaG67QjYwtDNhZGWMN3Ik4M/edit?usp=sharing

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

ICMP_ECHO_REQUEST = 8
MAX_HOPS = 30
TIMEOUT = 2.0
TRIES = 1
# 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(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 build_packet():
    #Fill in start
    # 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.
```

NEW YORK UNIVERSITY

```
    # Don't send the packet yet , just return the final packet in this function.
    #Fill in end

    # So the function ending should look like this

    packet = header + data
    return packet

def get_route(hostname):
    timeLeft = TIMEOUT
    tracelist1 = [] #This is your list to use when iterating through each trace
    tracelist2 = [] #This is your list to contain all traces

    for ttl in range(1,MAX_HOPS):
        for tries in range(TRIES):
            destAddr = gethostbyname(hostname)

            #Fill in start
            # Make a raw socket named mySocket
            #Fill in end

            mySocket.setsockopt(IPPROTO_IP, 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
                    tracelist1.append("* * * Request timed out.")
                    #Fill in start
                    #You should add the list above to your all traces list
                    #Fill in end
                recvPacket, addr = mySocket.recvfrom(1024)
                timeReceived = time.time()
                timeLeft = timeLeft - howLongInSelect
                if timeLeft <= 0:
                    tracelist1.append("* * * Request timed out.")
                    #Fill in start
                    #You should add the list above to your all traces list
                    #Fill in end
            except timeout:
                continue

            else:
                #Fill in start
                #Fetch the icmp type from the IP packet
                #Fill in end
                try: #try to fetch the hostname
                    #Fill in start
                    #Fill in end
                except herror:   #if the host does not provide a hostname
                    #Fill in start
```

```
            #Fill in end

        if types == 11:
            bytes = struct.calcsize("d")
            timeSent = struct.unpack("d", recvPacket[28:28 +
            bytes])[0]
            #Fill in start
            #You should add your responses to your lists here
            #Fill in end
        elif types == 3:
            bytes = struct.calcsize("d")
            timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]
            #Fill in start
            #You should add your responses to your lists here
            #Fill in end
        elif types == 0:
            bytes = struct.calcsize("d")
            timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]
            #Fill in start
            #You should add your responses to your lists here and return your list if your destination IP is met
            #Fill in end
        else:
            #Fill in start
            #If there is an exception/error to your if statements, you should append that to your list here
            #Fill in end
        break
    finally:
        mySocket.close()
```

@Edited by Chris Avalos 9.27.2020