ICMP Traceroute Lab

In this lab you will learn how to implement a traceroute application using ICMP request and reply messages. Students are strongly encouraged to first do the ICMP Ping lab before the ICMP Traceroute lab as it is done with the same approach. 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 become 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 code.
2. This lab requires the use of raw sockets. In some operating systems, you may need administrator/root privileges to be able to run your Traceroute program.
3. See the end of Lab 4 'ICMP Pinger' programming exercise for more information on ICMP.
4. This will not work for websites that block ICMP traffic.
5. You will have to turn your firewall or antivirus software off to allow the messages to be sent and received.

What to Hand in

You will hand in the complete code and screenshots of your Traceroute output for four different target hosts.

Skeleton Python Code for the ICMP Traceroute

```
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    = 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(string):
# In this function we make the checksum of our packet
# hint: see icmpPing lab
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.

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

# So the function ending should look like this

    packet = header + data
    return packet

def get_route(hostname):
```

```
	timeLeft = TIMEOUT
	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
					print("  *        *        *    Request timed out.")
				recvPacket, addr = mySocket.recvfrom(1024)
				timeReceived = time.time()
				timeLeft = timeLeft - howLongInSelect
				if timeLeft <= 0:
					print("  *        *        *    Request timed out.")

			except timeout:
				continue

			else:
				#Fill in start
				#Fetch the icmp type from the IP packet
				#Fill in end

				if types == 11:
					bytes = struct.calcsize("d")
					timeSent = struct.unpack("d", recvPacket[28:28 +
bytes])[0]
```

```
                                    print("  %d    rtt=%.0f ms    %s" %(ttl,
(timeReceived -t)*1000, addr[0]))


                              elif types == 3:
                                    bytes = struct.calcsize("d")
                                    timeSent = struct.unpack("d", recvPacket[28:28 +
bytes])[0]
                                    print("  %d    rtt=%.0f ms    %s" %(ttl,
(timeReceived-t)*1000, addr[0]))


                              elif types == 0:
                                    bytes = struct.calcsize("d")
                                    timeSent = struct.unpack("d", recvPacket[28:28 +
bytes])[0]
                                    print("  %d    rtt=%.0f ms    %s" %(ttl,
(timeReceived - timeSent)*1000, addr[0]))
                                    return


                              else:
                                    print("error")
                              break
                        finally:
                              mySocket.close()
get_route("google.com")
```

Optional Exercises

Currently the application only prints out a list of ip addresses of all the routers along the path from source to the destination. Try using the gethostbyname method to print out the names of each intermediate route along the route.