ECE 356 Fall 2021 HW 7

Group 3: Kyra Chan, Thomas Quintanilla, Tika Gergaia

November 11, 2021

ICMP Pinger Lab

1. (20 pts) Please fill in codes (marked with Fill in start and Fill in end) and comment the codes marked with #Explain each line to explain the purpose of each line you filled in. Please paste the codes and explanations from Fill in start to Fill in end here. After this, the program will be able to calculate the round-trip time for each packet and prints it out individually. You will hand in the screenshots of your Pinger output (the round-trip time or "Request timed out.") for five target hosts (umass.edu, alibaba.com, bbc.com, unimelb.edu.au, pretoriazoo.org), each on a different continent.

Round-trip time for umass.edu:

Round-trip time for alibaba.com:

Round-trip time for bbc.com:

Round-trip time for unimelb.edu.au:

Round-trip time for pretoriazoo.org:

Group3_ICMPPinger.py Code:

```
# Fill in start
# Fetch the ICMP header from the IP packet

# Set ICMP header to be bytes 20-28 from received packet
icmpHeader = recPacket[20:28]
# Unpack the ICMP header and split into each block
icmpType, icmpCode, icmpChecksum, icmpPacketID, icmpSequence = struct.unpack("bbHHh",
icmpHeader)

# If correct packet is received
if type != 8 and icmpPacketID == ID:
    # Calculate byte size of timestamp data
    bytesInDouble = struct.calcsize("d")
    # Unpack timestamp for time sent
    timeSent = struct.unpack("d", recPacket[28:28 + bytesInDouble])[0]

    # Calculate the time delay
    return timeReceived - timeSent

# Calculate if request timed out
timeLeft = timeLeft - howLongInSelect

# Time has exceeded limit, request has timed out
if timeLeft <= 0:
   return "Request timed out."

# Fill in end
```

2. (30 pts) 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 in the following ways:

   (a) 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).

Stats for umass.edu:

Stats for alibaba.com:

Stats for bbc.com:

Stats for unimelb.edu.au:

Stats for pretoriazoo.org:

(b) 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 message types are listed in Fig. 1. All messages shown in Fig. 1 are ICMP response error codes except echo reply (to ping). For simplicity, you only need to identify "Destination Network Unreachable", "Destination Host Unreachable", and "Destination Protocol Unreachable", and output other messages as "Other Errors".

Error results:

Please provide screenshots verifying that your ICMP Pinger program works as required. The screenshots will include the round-trip time for all packets, the minimum, maximum, and average RTTs, and the packet loss rate (in percentage). The screenshots will include ICMP response error when your program detect timeouts. (To get an ICMP response error, you can disconnect your computer from WiFi connection and set dest = "128.119.8.148" (IP address of umass.edu) in Skeleton Python Code for the ICMP Pinger.) Please paste the complete ICMP Pinger code here and comment the code explain the purpose of each line you filled in. Please submit the code (named as "GroupID_ICMPPinger.py") in Sakai.

Group3_ICMPPinger.py Code:

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

ICMP_ECHO_REQUEST = 8
timeRTT = [] # list to accumulate RTTs
packageSent = 0 #counter for number of sent packages
packageReceived = 0 # counter for number of received packages
def checksum(string):
    csum = 0
    countTo = (len(string) // 2) * 2
    count = 0
    while count < countTo:
        thisVal = string[count + 1] * 256 + string[count]
        csum = csum + thisVal
        csum = csum & 0xffffffff
        count = count + 2
    if countTo < len(string):
        csum = csum + ord(string[len(string) - 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):
    global packageReceived
    timeLeft = timeout
    while 1:
        startedSelect = time.time()
        whatReady = select.select([mySocket], [], [], timeLeft)
        howLongInSelect = (time.time() - startedSelect)
        if whatReady[0] == []: # Timeout - if packet is empty
            return "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]
        icmpType, icmpCode, icmpChecksum, icmpPacketID, icmpSequence =
        struct.unpack("bbHHh", icmpHeader)

        # Decomposition of error codes
        if icmpType != 0:
            if icmpType == 3:
                if icmpCode == 0:
                    return "Destination Network Unreachable"
                elif icmpCode == 1:
                    return "Destination Host Unreachable"
                elif icmpCode == 2:
                    return "Destination Protocol Unreachable"
            else:
                return "Other Errors"

        # If correct packet is received
        if icmpPacketID == ID:
            # Calculate byte size of timestamp data
            bytesInDouble = struct.calcsize("d")
            timeSent = struct.unpack("d", recPacket[28:28 + bytesInDouble])[0]
            timeRTT.append(timeReceived - timeSent) # append RTT time to the list
            packageReceived = packageReceived+1 #increase number of received packages by 1
            return timeReceived - timeSent

        timeLeft = timeLeft - howLongInSelect

        if timeLeft <= 0:
            return "Request timed out."
    # Explain each line
    # Fill in end


def sendOnePing(mySocket, destAddr, ID):
    global packageSent, timeRTT
    # 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
    packageSent=packageSent+1 #increase number of sent packages by 1
    # 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)
    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 = "128.119.8.148"
    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)
         if (len(timeRTT)>0):
             print("maxRTT: ", max(timeRTT)) #max of the list
             print("minRTT: ", min(timeRTT)) #min of the list
             print("averageRTT: ", float(sum(timeRTT)/len(timeRTT))); #average of all RTTs
             print("Package loss rate: ", (packageSent - packageReceived)/packageSent)
             #average rate of package loss
         else:
             print("maxRTT: ", 0)
             print("minRTT: ", 0)
             print("averageRTT: ", 0)
             print("average loss rate:", 0)
        time.sleep(1) # one second return delay
```

```
#ping("umass.edu")
#ping("alibaba.com")
#ping("bbc.com")
#ping("unimelb.edu.au")
ping("pretoriazoo.org")
```

ICMP Traceroute Lab

1. (20 pts) Please fill in code marked with Fill in start and Fill in end and comment the code marked with Explain each line to explain the purpose of each line you filled in. Please paste the code and explanation from Fill in start to Fill in end here if you don't change other parts of the code. If you change other parts of the code, please paste the complete code here and comment the code explain the purpose of each line you filled in. You will also hand in screenshots of your Traceroute output for five target hosts (umass.edu, alibaba.com, bbc.com, unimelb.edu.au, pretoriazoo.org), each on a different continent. Your output will include the TTL, RTT, and the IP address of each intermediate route along the route. Please submit the code (named as "GroupID_ICMPTraceroute.py") in Sakai.

   Traceroute for umass.edu:

```
umass.edu
 1 rtt=10 ms 10.194.0.2
 1 rtt=8 ms 10.194.0.2
 * * * Request timed out.
 * * * Request timed out.
 3 rtt=20 ms 10.236.254.238
 3 rtt=6 ms 10.236.254.238
 4 rtt=16 ms 10.236.242.130
 4 rtt=86 ms 10.236.242.130
 5 rtt=24 ms 10.236.254.102
 5 rtt=48 ms 10.236.254.102
 6 rtt=93 ms 128.109.1.241
 6 rtt=10 ms 128.109.1.241
 7 rtt=37 ms 128.109.9.33
 7 rtt=56 ms 128.109.9.33
 * * * Request timed out.
 * * * Request timed out.
 * * * Request timed out.
 9 rtt=120 ms 163.253.2.84
 10 rtt=60 ms 163.253.1.117
 10 rtt=83 ms 163.253.1.117
 11 rtt=30 ms 163.253.1.45
 11 rtt=29 ms 163.253.1.45
 12 rtt=98 ms 192.5.89.221
 12 rtt=35 ms 192.5.89.221
 13 rtt=35 ms 192.5.89.58
 13 rtt=55 ms 192.5.89.58
 14 rtt=36 ms 18.2.8.90
 14 rtt=42 ms 18.2.8.90
 15 rtt=35 ms 69.16.1.0
 15 rtt=48 ms 69.16.1.0
 16 rtt=34 ms 192.80.83.113
 16 rtt=111 ms 192.80.83.113
 17 rtt=32 ms 128.119.0.78
 17 rtt=41 ms 128.119.0.78
 18 rtt=31 ms 128.119.8.148
```

Traceroute for alibaba.com:

```
alibaba.com
 1 rtt=6 ms 10.194.0.2
 1 rtt=6 ms 10.194.0.2
 * * * Request timed out.
 * * * Request timed out.
 3 rtt=10 ms 10.236.254.226
 3 rtt=46 ms 10.236.254.226
 4 rtt=81 ms 10.236.242.130
 4 rtt=10 ms 10.236.242.130
 5 rtt=48 ms 10.236.254.102
 5 rtt=74 ms 10.236.254.102
 6 rtt=23 ms 128.109.1.241
 6 rtt=42 ms 128.109.1.241
 7 rtt=72 ms 128.109.9.21
 7 rtt=21 ms 128.109.9.21
 8 rtt=53 ms 4.15.180.73
 8 rtt=66 ms 4.15.180.73
 * * * Request timed out.
 9 rtt=162 ms 4.69.206.69
 10 rtt=137 ms 4.14.100.90
 10 rtt=316 ms 4.14.100.90
 11 rtt=25 ms 47.246.112.201
 11 rtt=31 ms 47.246.112.201
 * * * Request timed out.
 * * * Request timed out.
 * * * Request timed out.
 * * * Request timed out.
 14 rtt=103 ms 47.246.136.125
```

Traceroute for bbc.com:

```
bbc.com
 1 rtt=5 ms 10.194.0.2
 1 rtt=9 ms 10.194.0.2
 * * * Request timed out.
 * * * Request timed out.
 3 rtt=22 ms 10.236.254.226
 3 rtt=20 ms 10.236.254.226
 4 rtt=49 ms 10.236.242.130
 4 rtt=47 ms 10.236.242.130
 5 rtt=42 ms 10.236.254.102
 5 rtt=44 ms 10.236.254.102
 6 rtt=66 ms 128.109.1.241
 6 rtt=29 ms 128.109.1.241
 * * * Request timed out.
 7 rtt=35 ms 128.109.9.33
 * * * Request timed out.
 * * * Request timed out.
 * * * Request timed out.
 * * * Request timed out.
 10 rtt=26 ms 163.253.1.129
 10 rtt=51 ms 163.253.1.129
 * * * Request timed out.
 * * * Request timed out.
 12 rtt=19 ms 151.101.192.81
```

Traceroute for unimelb.edu.au:

```
unimelb.edu.au
 1 rtt=11 ms 10.194.0.2
 1 rtt=48 ms 10.194.0.2
 * * * Request timed out.
 * * * Request timed out.
 3 rtt=43 ms 10.236.254.238
 3 rtt=5 ms 10.236.254.238
 4 rtt=7 ms 10.236.242.130
 4 rtt=5 ms 10.236.242.130
 5 rtt=10 ms 10.236.254.102
 5 rtt=40 ms 10.236.254.102
 6 rtt=32 ms 128.109.1.241
 6 rtt=75 ms 128.109.1.241
 7 rtt=23 ms 128.109.9.21
 7 rtt=44 ms 128.109.9.21
 8 rtt=83 ms 4.15.180.73
 8 rtt=40 ms 4.15.180.73
 9 rtt=59 ms 4.15.112.22
 9 rtt=43 ms 4.15.112.22
 10 rtt=20 ms 108.162.236.171
 10 rtt=49 ms 108.162.236.171
 11 rtt=39 ms 108.162.236.171
 11 rtt=56 ms 108.162.236.171
 12 rtt=38 ms 108.162.236.171
 12 rtt=56 ms 108.162.236.171
 * * * Request timed out.
 * * * Request timed out.
 * * * Request timed out.
 * * * Request timed out.
 15 rtt=177 ms 43.245.43.59
```

Traceroute for pretoriazoo.org:

```
pretoriazoo.org
 1 rtt=5 ms 10.194.0.2
 1 rtt=18 ms 10.194.0.2
 * * * Request timed out.
 * * * Request timed out.
 3 rtt=66 ms 10.236.254.238
 3 rtt=48 ms 10.236.254.238
 4 rtt=47 ms 10.236.242.130
 4 rtt=47 ms 10.236.242.130
 5 rtt=52 ms 10.236.254.102
 5 rtt=40 ms 10.236.254.102
 6 rtt=52 ms 128.109.1.241
 6 rtt=45 ms 128.109.1.241
 7 rtt=51 ms 128.109.9.21
 7 rtt=47 ms 128.109.9.21
 8 rtt=57 ms 4.15.180.73
 8 rtt=69 ms 4.15.180.73
 * * * Request timed out.
 * * * Request timed out.
 10 rtt=189 ms 213.19.192.134
 10 rtt=104 ms 213.19.192.134
 * * * Request timed out.
 * * * Request timed out.
 12 rtt=266 ms 41.66.132.246
 12 rtt=347 ms 41.66.132.246
 13 rtt=309 ms 197.189.193.1
 * * * Request timed out.
 * * * Request timed out.
 13 rtt=265 ms 197.189.193.1
```

Group3_ICMPTraceroute.py Code:

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

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
    csum = 0
    countTo = (len(string) // 2) * 2
    count = 0
```

```
    while count < countTo:
        thisVal = string[count + 1] * 256 + string[count]
        csum = csum + thisVal
        csum = csum & 0xffffffff
        count = count + 2

    if countTo < len(string):
        csum = csum + ord(string[len(string) - 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 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.

    #Fill in start

    # Make the header in a similar way to the ping exercise.
    # Start checksum at 0
    myChecksum = 0
    # gets ID of client
    myID = os.getpid() & 0xFFFF

    # Makes a dummy header with 0 as checksum
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, myID, 1)
    # Packs timestamp data into struct
    data = struct.pack("d", time.time())
    # Append checksum to the header.
    # Don't send the packet yet, just return the final packet in this function.


    # Calculates checksum on data and dummy header.
    myChecksum = checksum(header + data)

    # Get right checksum based on platform
    if sys.platform == 'darwin':
        # Convert 16-bit integers from host to network byte order
        myChecksum = htons(myChecksum) & 0xffff

    else:
        myChecksum = htons(myChecksum)

    # Creates header with real checksum
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, myID, 1)
```

```
    #Fill in end

    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
            # Sets up type of protocol to initialize socket
            icmp = getprotobyname("icmp")
            # Initializes socket
            mySocket = socket(AF_INET, SOCK_RAW, icmp)
            # 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] == []:
                    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
                icmpHeader = recvPacket[20:28]
                ICMPType, Code, myChecksum, myID, sequence = struct.unpack("bbHHh",
                icmpHeader)
                #Fill in end

                if ICMPType == 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 ICMPType == 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 ICMPType == 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()

print("--------------------------------------------")
print ("umass.edu")
get_route("umass.edu")
print("--------------------------------------------")
print ("alibaba.com")
get_route("alibaba.com")
print("--------------------------------------------")
print ("bbc.com")
get_route("bbc.com")
print("--------------------------------------------")
print ("unimelb.edu.au")
get_route("unimelb.edu.au")
print("--------------------------------------------")
print ("pretoriazoo.org")
get_route("pretoriazoo.org")
```