Project 5: ICMP Pinger

Due: 11:59pm, November 28, 2023

(total points: 100)

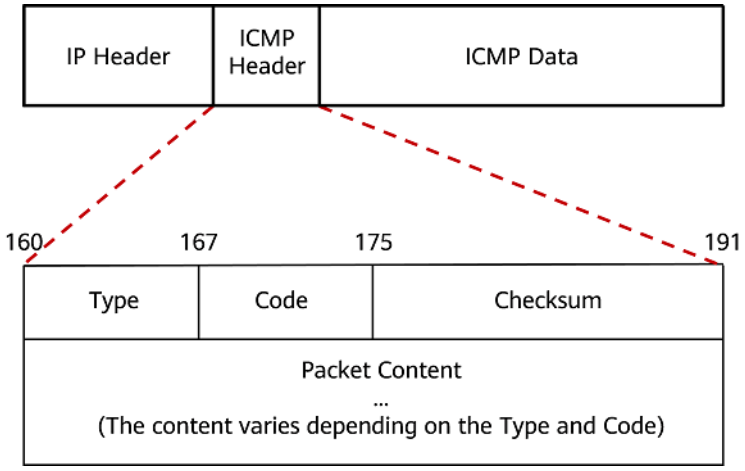
In this project, you will gain a understanding of Internet Control Message Protocol (ICMP). You will learn to implement a Ping application using ICMP request and reply messages.

Ping is a computer network application used to test whether a particular host is reachable across an IP network. It is also used to self-test the network interface card of the computer or as a latency test. It works by sending ICMP "echo reply" packets to the target host and listening for ICMP "echo reply" replies. The "echo reply" is sometimes called a pong. Ping measures the round-trip time, records packet loss, and prints a statistical summary of the echo reply packets received (the minimum, maximum, and the mean of the round-trip times and in some versions the standard deviation of the mean).

Your task is to develop your own Ping application in Python. Your application will use ICMP but, in order to keep it simple, will not exactly follow the official specification in RFC 1739. Note that you will only need to write the client side of the program, as the functionality needed on the server side is built into almost all operating systems.

You should complete the Ping application so that it sends ping requests to a specified host separated by approximately one second. **Each message contains a payload of data that includes a timestamp**. After sending each packet, the application waits up to one second to receive a reply. If one second goes by without a reply from the server, then the client assumes that either the ping packet or the pong packet was lost in the network (or that the server is down).

The below figure shows a structure of a ICMP packet, which include an IP header of 20 bytes, an ICMP header of 8 bytes, and ICMP data.



Below is the Python code skeleton for the client. The code is also given in a separate .py file for your convenience.

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

```
import time
import select
import binascii

ICMP_ECHO_REQUEST = 8



# In this function we make the checksum of our packet
def icmp_checksum(str_):
    str_ = bytearray(str_)
    csum = 0
    countTo = (len(str_) // 2) * 2

    for count in range(0, countTo, 2):
        thisVal = str_[count+1] * 256 + str_[count]
        csum = csum + thisVal
        csum = csum & 0xffffffff

    if countTo < len(str_):
        csum = csum + 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 recvPingPacket(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)


      # Fill in start
      #   one or multiple lines of your code
      #   You can use "struct.unpack" function with
      #   parameter "bbHHh" to decode the ICMP header
      # Fill in end


        if type != 8 and packetID == ID:
            bytesInDouble = struct.calcsize("d")
            timeSent = struct.unpack("d", recPacket[28:28 + bytesInDouble])[0]
            return timeReceived - timeSent

        timeLeft = timeLeft - howLongInSelect

```

```
        if timeLeft <= 0:
            return "Request timed out."

def sendPingPacket(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())



        # Fill in start
        #   one or multiple lines of your code
        #   Calculate the checksum on the data and the dummy header.
        # Fill in end


    # Get the right checksum, and put in the header
    if sys.platform == 'darwin':
        myChecksum = htons(myChecksum) & 0xffff
    #Convert 16-bit integers from host to network byte order.
    else:
        myChecksum = htons(myChecksum)

    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, ID, 1)

        # Fill in start
        #   one or multiple lines of your code
        #   use the socket to send the ICMP packet
        # Fill in end


def oneTimePing(destAddr, timeout):
    icmp = getprotobyname("icmp")
    #Create Socket here
    mySocket = socket(AF_INET, SOCK_RAW, icmp)

    myID = os.getpid() & 0xFFFF  #Return the current process i
    sendPingPacket(mySocket, destAddr, myID)
    delay = recvPingPacket(mySocket, myID, timeout, destAddr)
    mySocket.close()
    return delay

def ping(host, timeout=1):
    dest = gethostbyname(host)
    print ("Pinging " + dest + " using Python:")
    print ("")
    #Send ping requests to a server separated by approximately one second
    while 1:
        delay = oneTimePing(dest, timeout)
        print("delay: "+format(delay*1000, ".3f")+" (ms)")
        time.sleep(1)# one second
    return delay

#ping("127.0.0.1")
ping("google.com")
```

Question 1: [80 points]

Complete the above 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 calculation, 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 Pinger program. If you encounter issue with raw socket when you use integrated GUI compiler, please try to create the socket in `oneTimePing` function using `mySocket = socket(AF_INET, SOCK_DGRAM, icmp)`
3. See the appendix for more information on ICMP.

Testing your ping code

First, test your client by sending packets to localhost, that is, 127.0.0.1. Then, you should see how your Pinger application communicates across the network by pinging servers in different continents.

Question 2: [20 points]

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).

What to Hand in

You will hand in the complete client Python3 code to address the above two questions. You also need to provide a report (using the given template) with the screenshots of your Pinger output for four target hosts (see below).

```
ping ox.ac.uk

ping www.tsinghua.edu.cn

ping 8.8.8.8

ping www.up.ac.za
```

Appendix: Internet Control Message Protocol (ICMP)

ICMP Header

The ICMP header starts after bit 160 of the IP header (unless IP options are used).

Bits	160-167	168-175	176-183	184-191
160	Type	Code	Checksum	
192	ID		Sequence	

- **Type** - ICMP type.
- **Code** - Subtype to the given ICMP type.
- **Checksum** - Error checking data calculated from the ICMP header + data, with value 0 for this field.
- **ID** - An ID value, should be returned in the case of echo reply.
- **Sequence** - A sequence value, should be returned in the case of echo reply.

Echo Request

The echo request is an ICMP message whose data is expected to be received back in an echo reply (”pong”). The host must respond to all echo requests with an echo reply containing the exact data received in the request message.

- Type must be set to 8.
- Code must be set to 0.
- The Identifier and Sequence Number can be used by the client to match the reply with the request that caused the reply. In practice, most Linux systems use a unique identifier for every ping process, and sequence number is an increasing number within that process. Windows uses a fixed identifier, which varies between Windows versions, and a sequence number that is only reset at boot time.
- The data received by the echo request must be entirely included in the echo reply.

Echo Reply

The echo reply is an ICMP message generated in response to an echo request, and is mandatory for all hosts and routers.

- Type and code must be set to 0.
- The identifier and sequence number can be used by the client to determine which echo requests are associated with the echo replies.
- The data received in the echo request must be entirely included in the echo reply.