CS3640 Programming Project-2

Due: Apr 30, 2021 midnight
Upload to ICON a single zip file containing all materials

Implement ICMP Ping **100 points**

Ping is a computer network utility 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 request” packets to the target host and listening for ICMP “echo reply” responses. 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).

This project requires you to develop your own Ping application. Your Ping will use ICMP but, in order to keep it simple, you do not have to exactly follow the official specification in RFC 1739. Also, 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 utility 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).

Skeleton Code and ICMP Details

We have provided a skeleton code below (tested in Python 2.7) that implements majority of the functionality. If you prefer to use a language other than Python, please discuss with the TA before implementing your solution.

You are required to fill in the missing code in the area marked with `#FillInStart` and `#FillInEnd` inside the routine **`receiveOnePing`**. Some important points: (i) study the **`sendOnePing`** method before trying to complete **`receiveOnePing`** (ii) this project uses raw sockets since ICMP does not run on transport layer protocols (iii) while we covered the basics of ICMP formats in lecture-21, you will the following ICMP details useful.

ICMP Header and Messages: The ICMP header starts after bit 160 of the IP header (since we are not using IP options).

Bit position	8 bits	8 bits	8 bits	8 bits
160	Type	Code	Checksum	
192	ID		Sequence	

- **Type** - ICMP type
- **Code** - Subtype to the given ICMP type
- **Checksum** - Error detecting data calculated from ICMP header + data by initially setting 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. All hosts must respond to echo requests with an echo reply containing the exact data received in the request message.

- Type should be set to 8
- Code should be set to 0
- ID and Sequence Number are used to match the reply with the request. 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 should be set to 0
- ID and Sequence Number are used to match the reply with the request, so return the same.
- The data received in the echo request must be entirely included in the echo reply

```
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 = ord(string[count+1]) * 256 + ord(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):
    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)
        #FillInStart
        #FillInEnd
        timeLeft = timeLeft - howLongInSelect

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

```
def sendOnePing(mySocket, destAddr, ID):
    # Header is type (8), code (8), checksum (16), id (16), sequence (16)
    # Make a dummy header with a 0 checksum
    myChecksum = 0

    # 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(str(header + data))

    # Get the right checksum, and put in the header
    if sys.platform == 'darwin':
        myChecksum = htons(myChecksum) & 0xffff
    else:
        myChecksum = htons(myChecksum)

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

    # 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.
    mySocket.sendto(packet, (destAddr, 1))


def ping(host, timeout=1):
    dest = gethostbyname(host)
    print("Pinging " + dest + " using Python:")
    print("")

    icmp = getprotobyname("icmp")
    myID = os.getpid() & 0xFFFF # Return the current process id

    # Send ping requests to a server separated by approximately a second
    while 1:
        mySocket = socket(AF_INET, SOCK_RAW, icmp)
        sendOnePing(mySocket, dest, myID)
        delay = receiveOnePing(mySocket, myID, timeout, dest)
        print(delay)
        mySocket.close()
        time.sleep(1)

    return delay
```

Testing and Submission

Running the server: First, test your client by sending packets to localhost (i.e., 127.0.0.1). Then, you should see how your Pinger application communicates across the Internet by pinging servers in two different continents.

Submission: You should submit the complete ping client code along with the screenshots of ping output for two target hosts, each on a different continent.

Note: we will not accept late submissions since it would interfere with technical interviews and also hinder our ability to compute the final grades for the class in time. However, we will accept and grade partial solutions submitted before the deadline.