Program Assignment #2



|  |  |
| --- | --- |
| 과목 | 컴퓨터네트워킹\_01 |
| 담당 | 안 종 석 교수님 |
| 학과 | 컴퓨터공학과 |
| 학년 | 4학년 |
| 학번 | 2013112015 |
| 이름 | 임진섭 |



**Lab 4: ICMP Pinger Lab**

In this lab, you will gain a better 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).

**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. In “receiveOnePing” method, you need to receive the structure ICMP\_ECHO\_REPLY and fetch the information you need, such as checksum, sequence number, time to live (TTL), etc. Study the “sendOnePing” method before trying to complete the “receiveOnePing” method.
2. You do not need to be concerned about the checksum, as it is already given in the code.
3. 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.
4. See the end of this programming exercise for more information on ICMP.

**Testing the Pinger**

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.

**What to Hand in**

You will hand in the complete client code and screenshots of your Pinger output for four target hosts, each on a different continent.

**Skeleton Python Code for the ICMP Pinger**

## 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)

## 

## #Fill in start

## 

## #Fetch the ICMP header from the IP packet

## 

## #Fill in end

## 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)

## 

## 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(str(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

## # 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://sock-raw.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 = gethostbyname(host)

## print("Pinging " + dest + " using Python:")

## print("")

## # Send ping requests to a server separated by approximately one second

## while 1 :

## delay = doOnePing(dest, timeout)

## print(delay)

## time.sleep(1)# one second

## return delay

## 

## ping("google.com")Optional Exercises

1. 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).
2. 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 response error codes are 0: Destination Network Unreachable, 1: Destination Host Unreachable.

## Internet Control Message Protocol (ICMP)

*ICMP Header*

The ICMP header starts after bit 160 of the [IP](http://en.wikipedia.org/wiki/Internet_Protocol) 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.

|  |
| --- |
| ICMPping.py 소스코드 |
| **import** os **import** sys **import** struct **import** time **import** select **import** socket **import** binascii  ICMP\_ECHO\_REQUEST = 8   **def** checksum(str):  csum = 0  countTo = (len(str) / 2) \* 2   count = 0  **while** count < countTo:  thisVal = str[count + 1] \* 256 + str[count]  csum = csum + thisVal  csum = csum & 0xffffffff  count = count + 2   **if** countTo < len(str):  csum = csum + str[len(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** receiveOnePing(mySocket, ID, timeout, destAddr):  **global** rtt\_min, rtt\_max, rtt\_sum, rtt\_cnt  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  # Fetch the ICMP header from the IP packet* type, code, checksum, id, seq = struct.unpack(**'bbHHh'**, recPacket[20:28])  **if** type != 0:  **return 'expected type=0, but got {}'**.format(type)  **if** code != 0:  **return 'expected code=0, but got {}'**.format(code)  **if** ID != id:  **return 'expected id={}, but got {}'**.format(ID, id)  send\_time, = struct.unpack(**'d'**, recPacket[28:])   rtt = (timeReceived - send\_time) \* 1000   ip\_header = struct.unpack(**'!BBHHHBBH4s4s'**, recPacket[:20])  ttl = ip\_header[5]  saddr = socket.inet\_ntoa(ip\_header[8])  length = len(recPacket) - 20   **return '{} bytes from {}: icmp\_seq={} ttl={} time={:.3f} ms'**.format(length, saddr, seq, ttl, rtt)  *# Fill in end* 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)* 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'**:  myChecksum = socket.htons(myChecksum) & 0xffff  *# Convert 16-bit integers from host to network byte order.* **else**:  myChecksum = socket.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  # 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 = socket.getprotobyname(**"icmp"**)  *# SOCK\_RAW is a powerful socket type. For more details see: http://sock-raw.org/papers/sock\_raw  # Fill in start  # Create Socket here* mySocket = socket.socket(socket.AF\_INET, socket.SOCK\_RAW, icmp)  *# Fill in end* 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):  **global** rtt\_min, rtt\_max, rtt\_sum, rtt\_cnt  rtt\_min = float(**'+inf'**)  rtt\_max = float(**'-inf'**)  rtt\_sum = 0  rtt\_cnt = 0  cnt = 0  *# 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 = socket.gethostbyname(host)  print(**"Pinging "** + dest + **" using Python:"**)  print(**""**)  *# Send ping requests to a server separated by approximately one second* **while** 1:  cnt += 1  print(doOnePing(dest, timeout))  time.sleep(1)  **if** cnt == 10:  **break** ping(**"127.0.0.1"**) |
| 기본적으로 주어진 코드 부분에 Fill in start와 Fill in end 사이에 추가적인 코드를 넣었다. 먼저 type, code, checksum, id, seq = struct.unpack('bbHHh', recPacket[20:28]) 는 structs의 형태를 Ptyhon에 알맞게 변환하기 위해서 사용하였다. 그리고 다음의 if문들은 packet을 받아왔을 때 발생하는 오류를 나타내기 위한 부분이다. 다음으로 ip 부분의 header를 얻기 위해 struct.unpack을 이용하여 ip의 header를 얻어온다.  전체적인 구조는 receiveOnePing에서 ping이 들어오면 length, sequence number, time to live, round trip times를 출력한다. 맨 마지막 부분의 while문에서 10번의 ping을 받도록 하였다. |
| **결과화면(127.0.0.1)** |
|  |
| 127.0.0.1일 때의 결과화면이다. |
| **결과화면(google.com)** |
|  |
| google.com에 대한 결과화면이다. 다른 대륙에 대해 ping을 보내기 때문에 처음에 수행했던 127.0.0.1보다 더 높은 rtt값이 출력되었다. |
| **결과화면(ebay.uk)** |
|  |
| 유럽대륙에 있는 ebay.uk에 대한 결과화면이다. 아메리카 대륙에 있는 google.com보다 더 높은 rtt값이 출력되었다, |
| **결과화면(au.int)** |
|  |
| 아프리카 대륙에 있는 au.int에 대한 결과화면이다. 모든 대륙을 통틀어 가장 높은 rtt값이 출력되었다. |
| Optional Exercises 1 |
| *# Fill in start # Fetch the ICMP header from the IP packet* type, code, checksum, id, seq = struct.unpack(**'bbHHh'**, recPacket[20:28]) **if** type != 0:  **return 'expected type=0, but got {}'**.format(type) **if** code != 0:  **return 'expected code=0, but got {}'**.format(code) **if** ID != id:  **return 'expected id={}, but got {}'**.format(ID, id) send\_time, = struct.unpack(**'d'**, recPacket[28:])  rtt = (timeReceived - send\_time) \* 1000 rtt\_cnt += 1 rtt\_sum += rtt rtt\_min = min(rtt\_min, rtt) rtt\_max = max(rtt\_max, rtt)  ip\_header = struct.unpack(**'!BBHHHBBH4s4s'**, recPacket[:20]) ttl = ip\_header[5] saddr = socket.inet\_ntoa(ip\_header[8]) length = len(recPacket) - 20  **return '{} bytes from {}: sequence number={} time to live(TTL)={} round trip times(RTT)={:.3f} ms'**.format(length, saddr,  seq, ttl, rtt) *# Fill in end*  **while** 1:  cnt += 1  print(doOnePing(dest, timeout))  time.sleep(1)  **if** cnt == 10:  print(**'--- ping statistics ---'**.format(host))  print(**'{} Packets transmitted, {} Packets received, {:.1f}% Packet loss rate'**.format(cnt, rtt\_cnt, 100.0 - rtt\_cnt \* 100.0 / cnt))  print(**'RTT min/avg/max {:.3f}/{:.3f}/{:.3f} ms'**.format(rtt\_min, rtt\_sum / rtt\_cnt, rtt\_max))  **break** |
| 추가적으로 receiveOnePing에서 rtt\_max, rtt\_min, rtt\_sum을 정의해주고, rtt\_sum값을 업데이트 한다. 그리고 마지막 부분에 Packet 송신에 관한 정보와, RTT min avg,max 값을 얻기 위해 추가적으로 수정을 해주었다, |
| **결과화면(127.0.0.1)** |
|  |
| **결과화면(google.com)** |
|  |
| **결과화면(ebay.uk)** |
|  |
| **결과화면(au.int)** |
|  |