CSE 3300/5299: Programming Assignment 2

ICMP Pinger Lab and Raw Sockets

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. You will also learn to use raw sockets.

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" messages to the target host and listening for ICMP "echo reply" messages. 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).

Goal: 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 are given part of the code; and you just need to complete the code (see below).

How to estimate RTTs: As a pinger, your application should estimate RTTs. To obtain a sample of RTT, you need two timestamps: sending time and receiving time. Think about how you can do this through ICMP "echo" and "echo reply" messages. Methodology: (1) Based on the specification for ICMP echo and reply (see pages 3 and 4), the corresponding reply should contain exactly the same content from the echo request. (2) Embed the sending time as the payload in a ICMP "echo" message, which will be contained in the payload of the corresponding ICMP "echo reply" message. (3) At the sender, it gets the time when an echo reply arrives, get the sending time embed in the echo reply message; the difference of these two timestamps is an instance of RTT.

Sending ping requests: Your pinger sends ping requests (i.e., ICMP "echo" messages) to a specified host. Two ping requests are separated by approximately one second.

Handling losses: 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 (enclosed in the assignment in HuskyCT) 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.

What to fill in:

1. Think about which part in recPacket (one of the return values of "mySocket.recvfrom(1024)" in receiveOnePing) is ICMP header. Hint: see information on ICMP on page 4.
2. Parse recPacket to obtain the various fields in ICMP header. Make sure that you check the various fields (including Type, Code, Checksum, ID, sequence number; see the fields on page 4). If the checking confirms that this is indeed an expected ICMP reply message, then get the sending time from the ICMP payload, and obtain an instance of RTT. Otherwise, print an error message on what fields are not as expected.

3. How to check checksum at the receiver? You can use MyChecksum function that has been provided to you. This function basically follows the algorithm that we described in class. To calculate checksum, extract the checksum value, zero the field and compute the checksum with this field being zero. Then compare the calculated value with the one that you extracted. Another way is that you calculate the checksum of the entire packet (ICMP header + payload).

Additional Requirements for CSE5299 Students

1. Modify your pinger to make it look more like the standard ping program: (i) provide '-c' option to specify the number of Echo request messages to use. (ii) provide '-t' option to specify TTL explicitly, and (iii) at the end (i.e., after sending a specified number of Echo request messages), report the minimum, maximum, and average RTTs at the end. In addition, calculate and show the packet loss rate (in percentage).
2. 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 (i) Destination Network Unreachable, (ii) Destination Host Unreachable.

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 writing the checksum function, 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

When you hand in your programming assignment, you should include the following (please submit to HuskyCT):

- A program listing containing in-line documentation. Uncommented code will be heavily penalized.
- Screenshots of your Pinger output for four target hosts, each on a different continent.
- A separate (typed) document describing the overall program design, a verbal description of "how it works", and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made). The format of the description file should be as follows (**If your turn-in does not follow the above format, you'll get 10 points off automatically**):

  Description: describe the overall program design and ``how it works"
  Tradeoffs: discuss the design tradeoffs you considered and made.
  Extensions: describe possible improvements and extensions to your program, and describe briefly how to achieve them.

Test cases: describe the test cases that you ran to convince yourself (and us) that it is indeed correct. Also describe any cases for which your program is known not to work correctly.

Grading policy

Program Listing	
works correctly	50 points (shown by testing results)
in-line documentation	10
quality of design	10
Design Document	
description	5
tradeoffs discussion	5
extensions discussion	5
Thoroughness of test cases	15
Total	100 points

Late Programs NOT allowed

Notes:

- A full 10 points for quality of design will only be given to well-designed, thorough programs. A correctly working, documented and tested program will not necessarily receive these 10 points.

A Few Words About Borrowing Code...

As many of you probably already know, there is a wealth of sample sockets code out on the Web and in books that you can use in your projects. Often learning from sample codes is the best way to learn. I have no problem with your borrowing code as long as you follow some guidelines:

- You may not borrow code written by a fellow student for the same project.
- You must acknowledge the source of any borrowed code. This means providing a reference to a book or URL where the code appears (you don't need to provide reference for the code that was given to you).
- In your design document, you must identify which portions of your code were borrowed and which were written by yourself. This means explaining any modifications and extension you made to the code you borrowed. You should also use comments in your source code to distinguish borrowed code from code you wrote yourself.
- You should only use code that is freely available in the public domain.

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.