Project 2: Simple Traceroute　　Due: By midnight November 22, 2016

The goal of Project 2 is to give you experience working with raw sockets and creating and IP and ICMP headers. You will implement a highly simplified version of traceroute, bootstrapping from python code you have been given. Time-to-live values will be set as described below, and the responses (if any), processed.

Part 0: Set up Linux Virtual Machine and Get Started

Some operating systems, such as Mac OS X do not always work with raw sockets in the way that would be expect. Consequently, unless you have access to a Linux system already, you will need to set up a Linux Virtual Machine (VM)in order to test your code. As an added benefit, if you've never set up a virtual machine before, this will show you how to do it, and give you the flexibility of using Linux rather than your own operating system. **If you are unfamiliar with Linux, or have trouble setting up the VM, please come see me, the earlier the better, and I can help you**.

1. Download and install virtualbox for your operating system from:

   `https://www.virtualbox.org/wiki/Downloads`
2. Download a linux iso. Go to `http://releases.ubuntu.com/14.04/` and download this version:

   `ubuntu-14.04.4-desktop-amd64.iso`
3. Setup up a VM using instructions from here: `https://www.virtualbox.org/manual/ch01.html`. I recommend using 20 Gigabytes as the disk size.
4. Install guest additions, using the instructions here: `https://www.virtualbox.org/manual/ch04.html`. This will allow you to resize the virtualbox window, and has other user interface benefits.
5. Under the Devices tab for virtualbox, you may wish to enable shared folders, shared clipboards and, shared drag-and-drop, enabling you to easily switch between your personal machine and the VM.
6. Under the Devices tab, look at your network settings. Make sure that the network adaptor you have is "Bridged Adapter", rather than NAT. This is so that your VM has its own IP address, accessible from the broader Internet.

Base code for your traceroute program. To help you get started, you have been given initial code sketching out the components of what you need to do, including setting up the send and receive raw sockets, and creating an ICMP header (but not yet creating the IP header).

`http://vumanfredi.web.wesleyan.edu/2016/COMP360/lectures/code/icmp_traceroute.py`

This code has comments, indicating pieces that need to be filled in. As is, the code will not run. When run, it should be run with the command `sudo python3 icmp_traceroute.py`. The command `sudo` gives the process root control: this is required since raw sockets require you to be root to use them.

Part 1: Create and Send ICMP Echo Request Packet

To create an ICMP Echo Request packet, you will need to create an IP header and an ICMP header, concatenate them in the appropriate order, and send the result over a raw socket. The code you have been given lays out the structure of what you need to do, and creates the ICMP header that you need for you, inserting the correct Type and Code fields into the header. To create the IP header, you will need to construct it similarly to how the ICMP header is created.

ICMP header. RFC 792 on the Internet Control Message Protocol (https://tools.ietf.org/html/rfc792), lays out the following structure for the ICMP header for an ICMP Echo Request or Reply message. In the case of an Echo Reply message, the Data field will contain the entire Echo Request (IP header and ICMP header) that triggered it. The code you have been given has already created the ICMP Echo Request header for you, using the python struct module (more at `https://docs.python.org/3/library/struct.html`.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     Type      |     Code      |          Checksum             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           Identifier          |        Sequence Number        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     Data ...
   +-+-+-+-+-
```

IP header. Recall that RFC 791 on the Internet Protocol (https://tools.ietf.org/html/rfc791), lays out the following structure for the IP header, where each row represents 32 bits. You will need to create your IP header according to this format, and combine the IP header and ICMP header to get ICMP packet to send over the raw socket. You should use Wireshark to check that your final ICMP packet is correctly formatted (after sending it), containing the values you put in the header fields.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |Version|  IHL  |Type of Service|          Total Length         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |         Identification        |Flags|      Fragment Offset    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Time to Live |    Protocol   |         Header Checksum       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Source Address                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Destination Address                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Options                    |    Padding    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

When creating your IP header, you can assume no Options field is needed, and hence neither the Options, nor the Padding fields need be used. Thus your header will comprise 20 bytes. Some of the header fields will need to be initialized based on what the user wishes: these field values are already being initialized in `icmp_traceroute.py`, with the option of passing them in by the command-line. Look at the code to see what these values are. Note that I am not requiring you to compute the Checksum value in the ip header: you may put a zero in this field.

Set the source address to the IP address of your machine or Linux VM (find this using the `ifconfig` command). As the the destination address, I suggest you use `172.16.100.1` which is within the Wesleyan firewall and will return ICMP Echo Reply packets. To see which other destination IP addresses you could use, run the traceroute command and see the IP addresses of the first few hops that reply. For instance, I see the following when I run traceroute on the Wesleyan network:

```
$ traceroute www.wesleyan.edu
traceroute to www.wesleyan.edu (129.133.7.68), 64 hops max, 52 byte packets
 1  129.133.176.1 (129.133.176.1)  3.627 ms  3.432 ms  2.941 ms
 2  172.16.100.1 (172.16.100.1)  3.278 ms  6.671 ms  5.402 ms
```

Raw sockets. The necessary infrastructure to send and receive packets over raw sockets has already been set up for you, and should not need to be changed. If you'd like to understand more about raw sockets, please see `http://sock-raw.org/papers/sock_raw`.

Part 2: Receive and Parse ICMP Echo Reply packet

The code you have been given has already been setup to receive packets over raw sockets. You should use Wireshark to check that you receive a reply to your ICMP packet is correctly formatted. What you need to do is to take what is received from the raw socket and use `struct.unpack` to unpack the fields, first the fields in the IP header, and then the fields in the ICMP header. The header field values are used in Part 3.

Part 3: Compute and Print Traceroute Information

You should add timing code to record the time from sending an echo request to receiving an echo response. Format your program output in a similar way to what the real traceroute program produces, with one line per TTL value, although you will only need to send one echo request per TTL. For example:

```
[TTL]  [DESTINATION IP ADDRESS]  [RTT]
```

What to submit

- `icmp_traceroute.py` file. Your python code should be well-commented, check for exceptions and shut down nicely (e.g., clean up sockets and other state when the web client or web proxy terminates for some reason).
- Readme file indicating how to run your code, giving a description of example inputs and outputs from running your code, and a short description of how you tested and checked your code.
- Two screen shots from Wireshark: one showing the ICMP packet you created (with the IP and ICMP headers expanded), and one showing the ICMP response you received (with the headers similarly expanded).

How to submit

On wesfiles, I have created directories for each student in the class, named according to email usernames, and have set permissions so that only the corresponding student can access the folder. You should upload your project code and write up to your folder. Since you will be reusing the folder for all project submissions, I suggest creating sub-folders for each project and puting your code and write-up for Project 1 in the Project 1 sub-folder. The path to your folder is as follows, substituting your email username for USERNAME.

`https://wesfiles.wesleyan.edu/home/vumanfredi/web/2016/COMP360/submissions-proj/USERNAME`

Grading policy

(40 points) Correctly formatted ICMP Echo Request packet generated and sent over raw socket.

(20 points) Successful reception of ICMP Echo Reply packet received from raw socket.

(15 points) Correct parsing of ICMP Echo Reply packet.

(10 points) Computing and printing out information about traceroute done.

(10 points) Readme

(5 points) Wireshark screen shots.

I recommend writing out pseudocode for what you need to do, then adding comments for the pieces to fill in. That way, if something isn't working or you don't have time to finish something, I can see what you were trying to do and possibly give you partial credit.