EECS 325/425
Project 2.
Due December 1 before 11:59pm.
EECS325: 80 points + 20 bonus points; EECS425: 100 points

In this project, you will investigate the correlation between several common inter-host distance metrics: the hop count, the RTT, and – for 425 crowd -- the geographical distance. Before you do your measurements, you need to implement a program that measures hop distance and RTT to a remote host. As you know, the traceroute tool gives you not just the number of hops between the two hosts, but also the IP addresses of all the hops and the RTT between the sender and each hop. This is more than what we require, and therefore traceroute is too heavy weight for our purpose: traceroute sends out a lot of probes, while your tool should need many fewer probes. In fact, your tool should be able to measure the number of hops between the sender and destination *using only one probe*. To do this, you can exploit the fact that ICMP error messages include an initial portion of the datagram that triggered the error. So, if you send out a datagram to a destination to an unreachable UDP port and set the TTL *x* to this datagram, the residual TTL *y* that the datagram has when it reaches the destination will be returned to you in the ICMP payload. Then you know that the difference (x-y) represents the number of hops between the sender and the destination.

To implement your hops and RTT measurement tool, you need so-called "raw sockets" as they allow you to control the values of the IP header fields (which as you know from Project 1 are filled out by the kernel for you if you use regular UDP or TCP sockets) as well as receive ICMP messages. You can learn about raw sockets in detail at http://sock-raw.org/papers/sock_raw but necessary simple details are provided below.

Internet measurement experiments are often done at large scale (unlike our "toy" experiments with only 10 destinations) and take a very long time to complete. Therefore, we often try to combine measurements, think ahead, and extract as much information from a single experiment as we can. In this experiment, I would like you to combine the above measurement with answering another question: while ICMP stipulates that the ICMP error messages include only 28 initial bytes of the datagram that triggered the error, the rumor has it that in practice these error messages include much larger part of the datagram. I would like you to construct your probes to include some payload to make the probe datagrams have the maximum length of 1500 bytes (or 1472 of payload plus 8 bytes of UDP header plus 20 bytes of IP header). Then when you receive the ICMP response, see how much of the original datagram the response contains. Produce the result in your report as a table. Note: When we send measurement traffic to unsuspecting sites, we try to mark our traffic as explicitly as we can to distinguish it from attack traffic. So, in your 1472 bytes of payload, please start with text "measurement for class project. questions to student abc123@case.edu or professor mxr136@case.edu" and then pad the rest with some character, e.g., "a".

Thus, the output of your tool must include, for each destination, (a) the number of router hops between you and the destination, (b) the RTT between you and the destination, and (c) the number of bytes of the original datagram included in the ICMP error message.

Important notes:

(1) You will need to use python to implement this program. If you do not know it, just find some intro tutorial with lots of examples. You only need acquire rudimentary understanding of the language, just enough to implement the task at hand. Note: you can find on the Internet code that implements somewhat similar functions, such as ping and traceroute. You are welcome to read this code or even reuse parts of it *as long as you document carefully what code is yours and what you adopted from elsewhere.* **Your machine has python 3 installed; you need to access it using "python3" command** (instead of usual "python" command, which does not exist).

(2) You will need to have root privilege to work with raw sockets. You each have received your own (virtual) machine at your disposal (sent to you individually). **Extremely important: do NOT change the passwords associated with your VM. Doing so will invalidate your credentials and you will lose access!** There is no IDE on this machine – you would have to use an old-fashioned text editor and command-line terminal window to write and debug your code. You can of course try to develop/debug on your own machine but it will work well only if your machine runs linux: I am told Window's support for raw sockets is spotty so your mileage may vary. Rami (my PhD student) tried his implementation of this project on Windows 10 and it worked fine. Note: if you do use your own machine, I believe CaseGuest wi-fi blocks ICMP messages.

(3) You will need to create a datagram with custom values of some header fields for your probe. The easiest way to do it is to create a socket of type socket.SOCK_DGRAM and then use setsockopt() to change the fields of your socket that you need to change. Then you can send it with socket.sendto() without going through the trouble of building the rest of the headers yourself.

(4) You will also need to create a raw socket to receive ICMP messages. To create a raw socket for receiving an ICMP datagram, you can use the following function:

```
recv_sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)
```

after which you can get a packet from this socket by using the following:

```
icmp_packet = recv_sock.recv(max_length_of_expected_packet)
```

(5) A few other useful functions:

a. To extract a field from a packet, you can use "struct" module, in particular struct.unpack function. Keep in mind that whenever you extract a multi-byte integer (e.g., port number) you need to make make sure the order in which the bytes appear in the packet (the most significant byte first) matches the order in which these integers are represented in the computer you happen to use. A portable way to handle this is to use so-called "network order" when extracting the bytes from the packet. E.g., if packet[x:x+1] represents the two-bytes port number in a packet, your can convert it to int by the following:

```
port_from_packet = struct.unpack("!H", packet[x:x+2])[0]
```

Here, the first argument "!H" speficies format of the packet fragment being extracted, in this case signifying that the two bytes from packet appear in the network order ("!") and they represent an unsigned short ("H").

b. However, when you need to extract only a single byte, you can just convert it to integer using ord(byte) function.

(6) You will need to think **how you will match ICMP responses with the probes you are sending out (list all ways you could think of in your report and use the one that you found to work for you).** Note that your socket may receive unrelated packets since there is no port number to distinguish "your" packets from someone else's (i.e., another process running on your host). For example, one technique would compare the IP address of our measurement target to the source IP address of the ICMP response and match probes with ICMP responses when these IP addresses match. This technique is going to work most of the time but not all the time: sometime you may receive the ICMP response from an IP address

that is different from the address you are probing. (We have actually seen this behavior. I speculate it's due to the use of a load balancer at the destination. In this setup, when you resolve the destination's domain name, you get the IP address of the load balancer, which receives your packets and forwards them to one of the servers in a server cluster. However, the ICMP messages that the selected server generates, carry the server's own IP address, which is different from that of the load balance which your probe used.) Thus, you need to think of as many means for matching as you can because different techniques will work at different times. *You only need to implement one technique (but not the one I described above as an example) but describe as many as you can.*

(7) You need to allow for a possibility that you will get no answer **(list all possible reasons you can think of for not getting the answer when probing an arbitrary host)**; hence your program should not be stuck on reading from a socket forever. You will need to use either a "select" or "poll" call on the socket (read about them – google for "socket select") before reading from the socket. WARNING: do not process this error by changing the destination port number – you may get a nasty call from network admins! This will be interpreted as port scanning. I suggest you just try couple times and if you still get no answer, give up on this destination and produce some error message and move on to measuring the next site. You can check manually if it is your program's fault by trying to reach this destination using a standard ping measurement (there is a small chance that ping would be treated differently and experience a different outcome, but if you see that your tool produces no answer while ping succeeds on many destinations, this should increase your doubt in the correctness of your tool).

Your script should read the file "targets.txt" (collocated in the same directory as your script), which includes the set of websites you are exploring and accept no arguments, and obtain the IP address of each target using socket.gethostbyname method. The script should print out the result on standard output (in addition to any file that you find convenient to produce the plot below) that is understandable to the grader. Please name the script "distMeasurement.py".

Once you have your tool, measure the hop count as well as RTT to each of the alexa sites you worked with in earlier homework assignments. Produce a scatter graph to visualize the correlation: have hops on X-axis, RTT on Y-axis, and for each remote host, place a dot with corresponding coordinates on the graph. This is a typical technique to visualize correlation. Note, as mentioned, you may not get responses from some of the servers. In order to produce a meaningful scatterplot, pick some other sites that were not included in your original list of ten. Keep probing until you have around ten. You can say in your report that you are substituting these sites because your original sites did not respond.

425 crowd only (20 points; 20 bonus points for 325):

You will also need to measure geographical distance between your machine and your set of destinations, and to see how the geographical distance correlates with the other two distance metrics above. To measure the geographical distance, you should follow the following approach.

(a) Download a complete dataset of geographical locations that you can then use offline as you please. Namely, download GeoLite2 City dataset from http://geolite.maxmind.com/download/geoip/database/GeoLite2-City.mmdb.gz which you can access using python API from http://dev.maxmind.com/geoip/geoip2/downloadable/#MaxMind_APIs

You can install this package using "pip3 install geoip2" command.

(b) Write a script "geoDistance.py" that reads the file "targets.txt" and accept no arguments. The script must:

- Obtain the public IP address of your machine, by accessing the "checkip.amazonaws.com" URL and parsing the result. This IP address will change after each reboot, therefore access this URL from your python script. You can use urllib package for this; the following is a tutorial on how to use it: https://docs.python.org/3.5/howto/urllib2.html

- Using GeoIP, obtain coordinates of your machine's public IP address and of each of your targets in the "targets.txt" file (make sure you resolve domain names into IP addresses first). Search the Internet for the formula to compute the geographical distance between two points from the points' geographical coordinates.

- The script should print out the result on standard output (in addition to any file that you find convenient to produce the plots below) that is understandable to the grader.

(c) Produce two additional scatter graphs to visualize the correlation between number of hops and geo-distance and RTT and geo-distance. Compute the correlation coefficients between all three metric pairs.

Deliverables:

1. Submit to canvas: A single zip file with (a) all programs (make sure they are well commented and include instructions how to run them – arguments etc. Under-documented programs will be penalized.); (b) Project report that includes all measurement results, graphs, correlation coefficients, conclusions that you draw from your measurements, **and the answer to all questions in bold.**
2. Place your project into directory "<home-directory>/project2" in your VM. (You can use sftp to upload.) Then create a copy of that directory, "<home-directory>/project2grading". We will be testing your work by going into the project2grading directory and issuing the commands "python3 distMeasurement.py" and, for 425 folks, "python3 geoDistance.py".

Grading rubrics:

Code for measuring hop distance and RTT to a given destination: 25 points;
Code for obtaining the number of initial bytes from the original datagram included into the ICMP error message: 15 points;
Code for reading the set of destinations from a file and obtaining the above metrics to all the destinations: 15 points;
The writeup with answers to questions in bold (see above), data, and graphs (325 part): 25 points;
The geographic section: 20 points: Code to obtain geographic distances – 15 points; the portion of the report pertaining to this section – 5 points.