EECS 325/425
Project 2.
Due April 21 at 11:59pm.
EECS325: 45 points + 10 bonus points; EECS425: 55 points

(EECS 325 and 425 crowds: 45 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 different (x-y) represents the number of hops between the sender and the destination.

To implement your hops/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). Google for "raw socket tutorial" to learn the raw socket API.

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, the answer to which I myself don't know yet! The question is this: 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 them have the maximum length of 1500 bytes (or 1480 plus 20 bytes of IP header). Then when you receive the ICMP response, see how much of the original datagram the response contains. If each of you reports on 10 destinations, we will in aggregate have this information on 500+ sites, which will give me some idea of what's common.

Thus, the output of your tool must include (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*.

(2) You will need to have root privilege to work with raw sockets. You each will have your own (virtual) machine at your disposal (to be 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. (A word of caution: in the past years I suggested that windows users could install a virtual machine and run linux on

past years I suggested that windows users could install a virtual machine and run linux on that VM. But students had tons of trouble with this setup, so it appears not worth the trouble. Given that the code you need is not complicated, it’s just easier to develop on the machine we provide for you. Also, if you do use your own machine, I recall CaseGuest wi-fi blocks ICMP messages.)

(3) You will need to create a datagram with custom values of some header fields. 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. Alternatively, you can set an IP_HDRINCL socket option to prevent the IP layer from populating IP header for you, and then fill out the header.

(4) *You will need to think* ***how you will match ICMP responses with the probes you are sending out.*** *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). Further note that there is no guarantee that someone on the path (e.g., VPN if you run your program while on VPN, or a wi-fi access network, which utilizes network address translation, or a firewall) would not change some of your packet header fields. In fact, somewhere on the path between your assigned host and the Case network boundary some network component does reset TTL of outgoing datagrams to another value if the datagram’s own TTL is greater than 32. So, please assign the initial TTL value of 32 to your probes to the destinations.*

(5) 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. 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 the IP addresses you are exploring and accept no arguments. 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 (10 points; 10 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, use the web service at http://freegeoip.net/

Write a script that sends ten HTTP requests for IP addresses of your 10 destinations (e.g., http://freegeoip.net/xml/129.22.104.136) and parse the response (which in this case will come in XML format) to extract geographical coordinates. Search the Internet for the formula to compute the geographical distance between two points from the points’ geographical coordinates.

Your script should read the file “targets.txt” and accept no arguments. The script should print out the

Your script should read the file "targets.txt" and accept no arguments. 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. Please name the script "geoDistance.py".

Note: the above web-based service very easy to use, but as an insurance policy (in case the above site is down when you need it), you should also 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.

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

Deliverables:

1. Submit to blackboard: 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. 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 "python distMeasurement.py" and, for 425 folks, "python geoDistance.py".

Grading rubrics:

Code for measuring hop distance and RTT to a given destination: 15 points;
Code for obtaining the number of initial bytes from the original datagram included into the ICMP error message: 5 points;
Code for reading the set of destinations from a file and obtaining the above metrics to all the destinations: 5 points;
The writeup with answers to questions, data, and graphs: 20 points;
The geographic section: 10 points.