jLab 9: Network programming with Scapy

25 points

**Due: Before class on 4/27/2022**

**Objective of this lab:**

1. Learn about low level network programming with the Scapy python library

# Ping [10 points]

Scapy (<https://scapy.net/>) is a versatile Python networking library that allows us to do low-level packet manipulation and creation. From the Scapy website:

*“Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. It can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery…”*

We’re going to use Scapy and our understanding of networking protocols to make clones of the **ping** and **traceroute** programs.

1. Download the **skeleton.py** program from the D2L lab. Take some time to read and understand the program. Make sure you understand a few key things before you start programming:  
     
   - How the getopt library works to parse command-line arguments  
   - How to set options for layer 3 and layer 4 in the code  
   - Where in the code your packet is actually sent  
     
   Write a ping program that does the following:  
   1. Accepts an IP address on the command line with the switch –h using the getopt library  
   2. Accepts an optional number of pings on the command line with the switch –n using the getopt library, if not specified default to 4 pings  
   3. Sends the pings using the scapy library printing out info about each response and the time each took  
   4. Sets the correct sequence number, starting at 1 (each sequence starts at 1 and increases by 1 for each packet in the same program run  
   5. If a packet times out you don’t need to print timing data, you can just print that it timed out  
     
   Some helpful information for this program:   
   - You can get a summary of the response with – answer.summary() – this is what I use in my sample output  
   - I created two variables, start\_time and end\_time, you can use (end\_time – start\_time) to get the elapsed time in seconds.  
     
     
   Sample output:  
   **python3 ping.py –h 8.8.8.8 –n 4**  
   PING 8.8.8.8  
   IP / ICMP 8.8.8.8 > 192.168.1.230 echo-reply 0 / Padding: icmp\_seq=1 ttl=64 time=0.084s  
   IP / ICMP 8.8.8.8 > 192.168.1.230 echo-reply 0 / Padding: icmp\_seq=2 ttl=64 time=0.079s  
   IP / ICMP 8.8.8.8 > 192.168.1.230 echo-reply 0 / Padding: icmp\_seq=3 ttl=64 time=0.034s  
   IP / ICMP 8.8.8.8 > 192.168.1.230 echo-reply 0 / Padding: icmp\_seq=4 ttl=64 time=0.042s

# Traceroute [15 points]

1. Write a traceroute program that does the following:  
     
   1. Accepts an IP address on the command line with the switch –h using the getopt library  
   2. Accepts an optional maximum number of hops on the command line with switch –n using the getopt library, if not specified default to 30  
   3. Sends the ICMP traffic using the scapy library printing out info about each response  
   4. Prints the hostname of each found node (if available) or the IP address if no hostname is found  
   5. Stops when it has reached the specified destination  
     
   Some helpful information for this program:  
   - If you’re not sure how to configure the packets to do a traceroute check the lecture notes for the ICMP class  
   - You can use the getHostnameFromIP() function in skeleton.py to try to find a hostname from an IP address, this function returns False if no hostname was found  
   - The source IP address of the response is in **answer.src**, you’ll need this to know when to stop your loop  
     
   Sample output:  
   **python3 traceroute.py –h 8.8.8.8**  
   traceroute to 8.8.8.8, 30 hops max  
   1 ‘localhost.localdomain’ (192.168.1.1) 0.080s  
   2 96.120.9.85 0.022s  
   3 96.110.217.117 0.026s  
   4 96.108.7.6 0.028s  
   5 96.108.7.9 0.028s  
   6 be-34-ar01.mckeesport.pa.pitt.comcast.net (69.139.168.141) 0.038s  
   7 be-31631-cs03.pittsburgh.pa.ibone.comcast.net (96.110.42.169) 0.037s  
   8 be-1211-cr11.pittsburgh.pa.ibone.comcast.net (96.110.38.134) 0.034s  
   9 be-302-cr12.ashburn.va.ibone.comcast.net 0.050s  
   10 be-1312-cs03.beaumeade.va.ibone.comcast.net (68.86.84.153) 0.044s  
   11 be-3212-pe12.ashburn.va.ibone.comcast.net (96.110.34.118) 0.050s  
   12 66.208.228.90 0.047s  
   13 -  
   14 142.251.70.83 0.043s  
   15 dns.google (8.8.8.8) 0.044s