COMP 332, Homework 7: *Routing and raw sockets*
Written due by 11:59pm on November 7, 2018
Programming due by 11:59p on November 14, 2018

---

1. Written problems (5 points)

Problem 1. *Consider the network graph shown in Figure 1. Use Dijkstra's algorithm to compute the shortest path from node u to every other node in the network. Show your computation using a table as was done in class. Your table should show:*

Step N' $D(s), P(s)$ $D(t), P(t)$ $D(v), P(v)$ ... $D(z), P(z)$

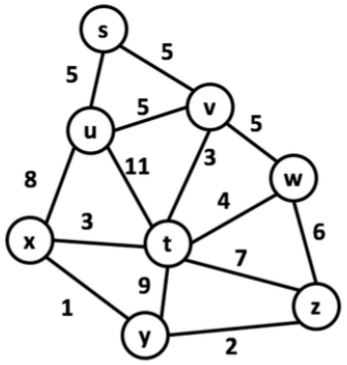


Figure 1. Network graph.

Solution:
See Figure 2.

Step	N'	D(s), P(s)	D(t), P(t)	D(v), P(v)	D(w), P(w)	D(x), P(x)	D(y), P(y)	D(z), P(z)
0	{u}	5,u	11,u	5, u	∞	8, u	∞	∞
1	{u, s}		11,u	5, u	∞	8, u	∞	∞
2	{u, s, v}		8,v		10, v	8, u	∞	∞
3	{u, s, v, t}				10, v	8, u	17, t	15, t
4	{u, s, v, t, x}				10, v		9, x	15, t
5	{u, s, v, t, x, y}				10, v			11, y
6	{u, s, v, t, x, y, w}							11, y
7	{u, s, v, t, x, y, z}							

Figure 2. Solution to Problem 1.

Problem 2.

a: *Briefly explain the count-to-infinity problem and why it can occur in distance vector routing.*

b: *Does the count-to-infinity problem occur when link costs decrease? Why or why not?*

c: *Does the count-to-infinity problem occur when two nodes are connected that did not previously have a link?*

d: *How does poisoned reverse address the count-to-infinity problem?*

e: *Describe a scenario where poisoned reverse does not address the count-to-infinity problem.*

Solution:

a: Since distance-vector routing only keeps track of the cost of a path and the first node for that path rather than all of the nodes in the path, it is possible for loops to occur. For instance, it may be that node X uses node Y to reach node Z. However, node Y will have no way of knowing this, and could, if the link cost from Y to Z increases, decide to route to Z via X, creating a loop.

b: No, the count-to-infinity problem does not occur when link costs decrease because decreasing the link cost won't cause a loop: either that link will be used instead (if a new lower cost path has been created), or the previous path will continue to be used.

c: Connecting two nodes with a link is equivalent to decreasing the link weight from infinite to finite weight, hence as per part (b), the count-to-infinity problem will not occur.

d: Poisoned reverse addresses the count-to-infinity problem by having nodes poison the reverse paths (by setting their path cost to infinity) for any of their neighbors that they use as the first hop on their path. So for instance, if node X routes to node Z via node Y, node X would advertise to node Y that node X's path cost to Z is infinite.

e: The count-to-infinity problem still exists when we have a loop of 3 or more nodes. Suppose we have two shortest paths W-X-Y-Z and X-Y-Z, with nodes W, X, and Y forming a triangle and Z hanging off Y. Y will not use X to reach Z, since that path is already poisoned. And X will not use W to reach Z since that path is already poisoned as well. Now suppose the link cost from Y-Z increases to infinity. Y could decide to use W to reach Z, since that path is not poisoned, not realizing that the path via W actually contains a loop: Y-W-X-Y-Z.

Problem 3. *Assume a distance-vector routing algorithm is used in a network of 60 nodes. If costs are recorded as 8-bit numbers and cost vectors are exchanged twice a second, how much capacity per (full-duplex) link is used by the distributed routing algorithm? Assume that each node has three links to other nodes.*

Solution: Distance vector size = $60 * 8 = 480$ bits (assuming an array representation for nodes 0 to 59). Thus, capacity needed 480*(2 times/second)*2(directions per link) = 1920 bps.

2. Coding and hands-on problems (25 points)

Problem 4. *The goal of this problem is to give you experience working with raw sockets and creating IP and ICMP headers. You will implement a 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 the most recent version of VirtualBox. VirtualBox is virtualization software that runs on your computer and permits you to run a virtual computer on your computer, in your case, a computer running Linux rather than Windows. You should read the beginning of Chapter 1 (First Steps) before continuing.*

```
https://www.virtualbox.org/wiki/Downloads
```

(2) *Download a linux iso. Go to* `http://releases.ubuntu.com/16.04/` *and download this version:* `ubuntu-16.04.4-desktop-amd64.iso`

(3) *Setup up a VM using instructions from here:* `https://www.virtualbox.org/manual/ch01.html`. *Choose the following options.*

```
Type: Linux
Version: Ubuntu (64-bit)
Memory size: 1024 MB
Create a virtual hard disk now: create
VDI (VirtualBox Disk Image)
Statically allocated
File location and size: 20 GB
```

(4) *Install guest additions, see instructions here:* `https://www.virtualbox.org/manual/ch04.html`. *This will allow you to resize the virtualbox window, copy paste from your machine to the desktop, and have a shared folder with your computer.*

(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.

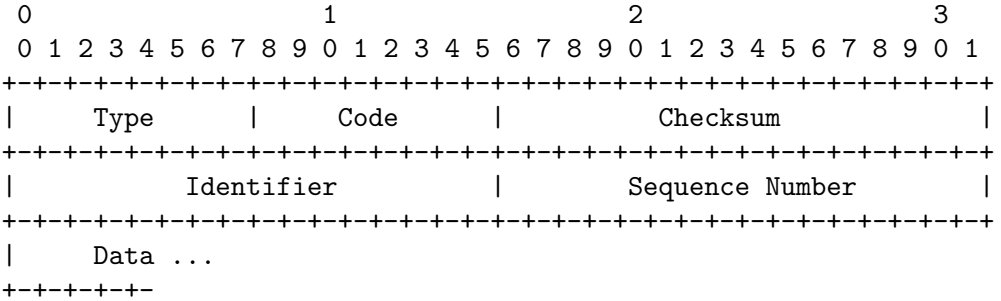
(6) *Under the Devices tab, look at your network settings. Make sure that the network adaptor you have is "Bridged Adapter", rather than NAT. This is so that your VM has its own IP address, accessible from the broader Internet.*

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 in* `icm_tracderoute.py`*, including setting up the send and receive raw sockets, and creating an ICMP header (but not yet creating the IP header).*

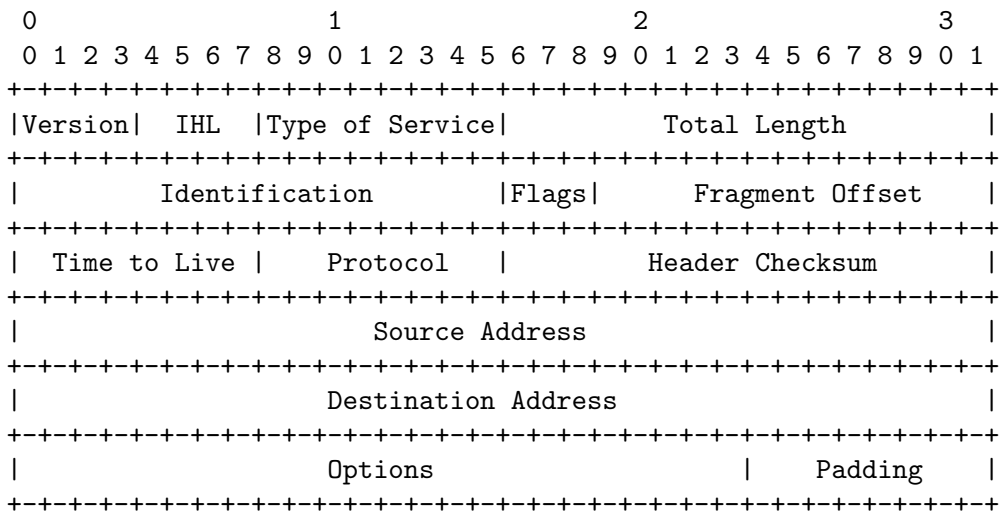
This code has comments, indicating pieces that need to be filled in. As is, the code will not run. When run, it should be run with the command `sudo python3 icmp_traceroute.py`*. The command* `sudo` *gives the process root control: this is required since raw sockets require you to be root to use them.*

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, 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 (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`*.*



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.*



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 for now you may put a zero in the checksum header field. You should set the source address to the IP address of your machine or Linux VM (find this using the* `ifconfig` *command).*

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 TTL Expired or 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.*

Note that if your IP packet does not contain your correct IP address, you will not get a response back. You can either manually enter your IP address (after checking for it with ifconfig). Or you can use the following in your code, after a socket has been created:

```
src_ip = send_sock.getsockname()[0]
```

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, see the output in Figure 3.*

```
traceroute to 8.8.8.8, 64 hops max
1 172.20.10.1 10 ms
2 10.167.45.49 67 ms
3 10.170.230.222 32 ms
4 10.170.230.227 39 ms
5 10.164.72.196 43 ms
6 10.164.165.75 45 ms
7 72.14.202.90 38 ms
8 108.170.248.33 50 ms
9 209.85.245.193 41 ms
10 8.8.8.8 40 ms
```

FIGURE 3. Example output for traceroute client.

Part 4: Checksum. *If you put 0 in the checksum field rather than actually computing the correct checksum value, some routers or servers may not respond back to your ICMP packet because your checksum is incorrect. Thus, our goal now is to correctly set the checksum. The value in the checksum field should be the one's complement sum of all 16-bit blocks in the header. For purposes of computing the checksum, the value of the checksum field is zero. Thus, to correctly set the checksum, perform the following steps. First, assume the value of the checksum field is zero. Then, divide the header into 16 bit blocks and sum them together. Because the sum might overflow beyond 16 bits, you'll want to add any carry bits back to the sum. Then take the 1s complement of this sum. Some issues you may run into is that your result is more than 16 bits or that your result is not in network byte order. More information on computing checksums at the following links (note that both the IP and the ICMP checksums are computed the same way).*

`http://www.faqs.org/rfcs/rfc1071.html`
`https://en.wikipedia.org/wiki/IPv4_header_checksum`

You may also find this reference on bit operations in python helpful.

`https://wiki.python.org/moin/BitManipulation`

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.

3. SUBMISSION

Upload your written work as `hw7.pdf` and your *.py files to the WesFiles directory I have created for you at the following URL. All files should include your name!

`https://wesfiles.wesleyan.edu/home/vumanfredi/web/comp332-f18/submissions/hw7/USERNAME`

You should replace `USERNAME` with your Wesleyan username. You will be asked to enter your Wesleyan username and password to access the page. Once the page opens, you should click on the "Open Web View" link that shows up on the page, and that should take you to a page that gives you options to upload files.