Network Programming
Project 1: IPv4 Router Simulation

For this project, you will be constructing a simulation of some of the functions of an IPv4 router. This project may be done in the language of your choice, so long as that language has an open-source implementation. However, **please contact me first if that language is not one of the following:** C, C++, Perl, Python, or Ruby.

Part 1: Routing Table Implementation

Your program must read a routing table from the file `routes.txt` in the current working directory. Each line in this file represents a routing table entry. The routing table will have three whitespace-delimited columns. Column 1 will be an IPv4 prefix in the format `a.b.c.d/x`, where a.b.c.d is an IPv4 address and x is the prefix length. Column 2 will be an IPv4 gateway address, again in `a.b.c.d` format. If the gateway address is `0.0.0.0` this indicates that the network given by the prefix is directly connected. Column 3 will contain an interface identifier (i.e. `eth0`, `eth1`, `ppp0`). Use the routing table entries given to construct a binary search trie of the type described in class. Note that if the gateway address is 0.0.0.0, and the prefix length is 32, this indicates a direct, point-to-point connection.

Our simulation will not include ARP requests or replies; instead, we'll read the address mappings ARP would provide from a file `arp.txt`. Each line again represents a table entry and has two whitespace-delimited columns: the first is an IP address and the second is an Ethernet hardware address in xx:xx:xx:xx:xx:xx (hexadecimal) format. Read this file into a data structure of your choice - I'd recommend a map or hash table.

Next, you will read PDU header information from the standard input. Each line represents an IPv4 PDU and contains six whitespace-delimited fields as shown below.

Field	Value
0	Interface on which PDU arrived
1	Source IPv4 address (a.b.c.d)
2	Destination IPv4 address (a.b.c.d format)
3	IPv4 protocol number (decimal integer)
4	Time to live (decimal integer)
5	Source port number
6	Destination port number

For each PDU in the file, first use the routing table to determine the next hop destination and interface. Then, if needed, use the ARP table to determine the data-link layer address to be use. (If the routing table entry **for the next hop address** indicates that the interface is point-to-point, do not use the ARP table. You will need to do a second routing table lookup to determine this.) Decrement the TTL field. Finally, print a line in one of the following formats:

```
1.2.3.4:11->5.6.7.8:22 via 9.10.11.12(eth0-aa:bb:cc:dd:ee:ff) ttl 11
10.11.12.13:33->20.21.22.23:44 via 30.40.50.60(ppp0) ttl 9
1.2.3.4:11->5.6.7.8:22 discarded (TTL expired)
1.2.3.4:22->99.88.77.66:33 discarded (destination unreachable)
```

That is:

```
source:src_port->dest:dest_port via gateway(iface-l2address) ttl x
source:src_port->dest:dest_port via gateway(pppinterface) ttl x
```

Use the first format for ARP-capable (non-point-to-point) interfaces and the second for point-to-point. If the TTL reaches zero or no route is found in the table, print a message to that effect.

Part 2: NAT Implementation

For this part, you will be reading NAT rules from `nat.txt`. This file has two columns. The first column of this file is the interface name where the translation will be applied. The second column has the translated source address (a.b.c.d format). Create a table of these rules and a table of current translations. Now, before "sending" a PDU (printing its information), check the rules tables. If the packet is going out a translated interface, apply the translation and add an entry to the translation table. Finally, check incoming PDUs against the translation table and reverse the translation. Note that if the TTL expires, you should not apply the translation, but print a message that the PDU was discarded.

For this part, I'll provide a test wrapper, because some PDUs your program receives will depend on how it translates other PDUs. I will make every effort to provide this by Feb. 6 (3 weeks before the due date.)

Extra Credit

Implement one of the following features to earn up to 10 points of extra credit.

1. Implement your routing table using either path-compressed binary search tries or multiway tries as we discussed in class. (max 10 pts)
2. Modify your program so that, when given a command line switch, it reads and writes correctly-formatted Ethernet and IPv4 headers instead of text-based substitutes. (See RFC 791 and http://wiki.wireshark.org/Ethernet for more information.) (max 10 pts)
3. Implement anti-spoofing protection: Using the routing table, look up the **source** addresses of the incoming packets to ensure that they arrived on the proper interface. Discard any spoofed packets arriving on invalid interfaces. (max 5 pts)

Helpful Hints

You will probably want to convert the IPv4 addresses to an integer representation and use bitwise operators when constructing and searching the binary search tries.

To check if bit i is set (where MSB is bit 0, LSB is bit 31):

```
if (address & (1<<(31-i)) {
      /* bit i is 1 */
} else {
      /* bit i is zero */
}
```