Lab: Router Lab

Background

In the previous lab, we created a Layer 3 Host, and we could create, send, and receive L3 packets. But, we didn't have any routing implemented, so really it wasn't all that useful. In this lab, we'll implement a router, thus allowing us to send packets across multiple networks. Additionally, we'll implement a L3Host that supports a routing table, and optionally, multiple interfaces on different networks.

Step 1: Get Set Up

I am giving you my implementation of the last lab – Layer 3 Lab. Please note these things:

- As in the last lab, my code has a `Layer2Endpoint`. A `Layer2Endpoint` is a subclass of a `L2Handler`. Your code probably just had a `L2Handler`. So, wherever you see "`Layer2Endpoint`" in my Javadoc, think "`L2Handler`".
- You are going to need to implement these classes – I'll guide you through these in this lab, but you might want to take a look at them now:
  - `L3Iface.java`
  - `L3RoutedIface.java`
  - `RoutingTable.java`
  - `Router.java`
  - `L3Host.java`
- `L3Host.java` is a little strange. It is an `L3Listener` (meaning that it will receive packets from the `L3Handler`), but it also has-a `L3Listener` field, which means another object can register to get packets from the `L3Host` object. That other object will be an `L3HostDisplay`.

My code for layer 3 is in `/home/cs/332/sp2016/layer3.` Documentation for this lab is in `/home/cs/332/sp2016/lab4router`. You will find there the documentation for my implementation, along with the code for `Test.java` and `L3Display.java`.

Step 2: Inspect the Final Test

To understand where we are going and what we have to implement, take a look at `Test.java's testRouting`() function – the only function called by `main`(). Notice

the comment above the function that gives a very bad little picture of the network topology the code creates. Now, look at the first part of the code. First, we create 3 LANs ("`LightSystems`"). Then, we create a router "r1". Then, we create a `Layer2Endpoint` (you should change this to a `L2Handler`) on the first LAN, with MAC address 11. We pass that object to a new `L3RoutedIface` object, which also takes the network id as a parameter (1). That object gets added as an interface to router r1. The code continues on from there.

Notice also how we add default routes to the routers.

And, we create 2 `L3Hosts`, each with 1 interface, and each being displayed with an `L3HostDisplay` GUI. I'm giving you my code for the `L3HostDisplay`. You may want to inspect that now and make sure you understand it. Note that the GUI allows the user to type any letters into the data field, and they will be converted to ASCII values and sent over the network.

Step 3: Subclass L3Handler

You have already implemented an `L3Handler` in the last lab.

But, look at my `L3Handler's` Javadoc. You can see I've updated my comments there for a few things. I've also made the class abstract. The class will be subclassed by `L3Iface` and `L3RoutedIface`, which only implement a constructor (which only calls the `super`() constructor) and the `dropReceivedPacket`() method. Create those classes now, and implement them, conforming to the Javadoc I've provided for you. Also, change L3Handler's `dropReceivedPacket`() to just throw a `RuntimeException`.

Now, change `L3Handler's send`() method to take the second parameter `nextHop`, and use it in its call to `handler.send`(). The code uses `nextHop.getHost`() to get the MAC address that is passed down in `handler.send`().

Add a simple `toString`() for `L3Handler` to just return a string saying "this is the L3 handler with *this* L3 address", or something like that.

Step 4: Router Class

Now, create a `Router.java` class. A `Router` instance holds an `ArrayList<L3RoutedIface>`, a `RoutingTable` instance, and a `String` name that is a nice human-readable name for this router. Write the constructor code, and write the code for `addL3Iface`(). For the latter, don't try to implement the code to call the routing table's `addDirectRoute`() call yet.

Now, implement the `packetReceived`() method, following the Javadoc I've provided for you. The last step in `packetReceived`() is a call to `route`(), which we haven't implemented yet.

Step 5: Routing Table Class

The `RoutingTable` class encapsulates all the functionality of storing a routing table, adding routes to it, and finding a best route given a destination L3 address. The class has an inner class `Entry` that represents, in essence, a single row in the table. Using the provided Javadoc, define the `RoutingTable` class and define and implement its inner class `Entry`. Note that my code has this line:

```
this.isLocal = nextHop.equals(new L3Address(0, 0));
```

which requires you to create an `equals`() method in `L3Address`.

In the outer class (`RoutingTable`), define two instance variables, `table`, an `ArrayList<Entry>` and `defaultRoute`, an `Entry`. Create the (empty) constructor, and write the code for `route`(). Note that the code in `route`() is not very complicated when we don't have subnet masks, etc…

In fact, write all the code for the class. The Javadoc should be very helpful to guide you in this.

Step 6: Back to the Router

Now that you've implemented the `RoutingTable` class, go back to your `Router` class. The first thing to do is to go to the `addL3Iface`() method and add a call to `addDirectRoute`(). This is consistent with how IP and IP interfaces work – when an interface is added on a machine, a direct route is added to the routing table.

Next, implement the `addRoute`() and `addDefaultRoute`()s methods, which just call the routing table's method, passing in the `Router` instances `ifaces` as the last parameter.

Next, implement `route`(). The Javadoc is extensive, so you should be able implement it from that. My code required the addition of a method to `L3Packet` called `decrHopsLeft`().

Finally, make sure your call to `route`() from `packetReceived`() looks correct.

Step 7: Now, to the host, with the most

Next, we need a class to represent a host object that can have one interface. Create a class called `L3Host`(). An `L3Host` is not that different from a `Router`, except that packets that are received are not routed. They are just passed up to any listener that has registered itself.

Look at the Javadoc and see the instance variables the class needs. A lot of the code in this class is similar or identical to the code in the Router class. In fact, I ***copied*** (Yuck!) the code for the two `addDefaultRoute`()s from my `Router` implementation (I couldn't figure out a nice way to share the code without copying it).

Implement all the methods in `L3Host`.

Step 8: Test and Submit your code

At this point, I think all the code should be implemented. See if it works!

NOTE NOTE NOTE: I will be testing the code with a more complicated network setup – e.g., with multiple hosts on the same network, and with a host on the network with the 2 routers. You should make sure you code works for these kinds of more complicated set ups.

Step 9: (Optional) A Deficiency

One thing I'm not happy about is that I don't have a really nice way to include in my print statements the identifier of the object that is generating the statement. For example, I see messages like these:

```
route: for dest 1.19 sending directly via iface 1.11
```

That's nice, but it doesn't say which "device" – r1, r2, host1, host2, etc. – is generating this message. If anyone can figure out a nice way to do this, I'm all ears.

Submit your code in `/home/cs/332/current/<yourid>/lab4router`.

Make sure your documentation is thorough and complete. Make sure you write beautiful hospitable code: good spacing, good consistent indentation, etc. Make me proud!

See the end of this doc for the Grading Rubric.

Step 10: For those doing the course for honors

You need to also add a host to the topology that is *multi-homed* – i.e., it is on two different networks with two different interfaces. To do this, make the `L3Host` support having multiple interfaces. This host should not do any routing between interfaces. Add code to the `Test.java` file to add this multi-homed host to the network.

Grading Rubric

20 points total:

- 14 points for code working
- 6 points for code beauty – good indentation, documentation, etc.

Extra credit:

- +5 points for using your own code submitted for Layer 3.
- +1 point for each 12 hours early you submit your code, up to a total of 8 points (4.5 days early).
    - E.g., If due date is May 3, 23:59:59, and you submit sometime between May 3, 0:00:00 and May 3, 11:59:59, you get +1 points. Submit between May 2, 11:59:59 and May 2, 23:59:59, and you get +2 points.