Lab: Router Lab

Background: Limitations of Ethernet

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.

Exercise: Get Set up

Copy the files from /home/cs/332/sp2014/lab4router/* to your own lab4 working directory. (Or, you may start with your implementation of lab3 if you wish.) As before, I’ve left you the javadoc output from my implementation of this lab, for you to use as a guide.

Exercise: 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 that comment above the function that gives a very bad little picture of the network topology the code sets up. 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 on the first LAN, with unique 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.

Notice also how we add default routes to the routers. We’ll have to implement the code to do that.

And, we create 2 L3Hosts, each with 1 interface, and each being displayed with an L3HostDisplay GUI. I’m giving you the code for the L3HostDisplay. You may want to inspect that now and make sure you understand it. Note that I’ve changed the GUI so that now you can type any letters into the data field and they will be converted to ASCII values and sent over the network.
**Exercise: Subclass L3Handler**

I’ve provided you lab3’s implementation of L3Handler.java. But, look at L3Handler’s javadoc (the javadoc is from my implementation of lab4router, so it doesn’t match the code in L3Handler.java). You can see I’ve updated my comments there for a few things. I’ve also made the class abstract. The class will be implemented 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.

**Exercise: 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.

**Exercise: 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 stores, in essense, 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("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.

**Exercise: 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.

**Exercise: Now, to the host, with the most**

Next, we need a class to represent a host object that can have one or more interfaces (we could implement a multi-homed host with this class). 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* (!) the code for addRoute(), and the 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.

**Exercise: Test and Submit your code**

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

**Exercise: 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!