Lab: Layer 3

Background: Limitations of Ethernet

In the previous lab, we simulated Ethernet switching on a LAN (local area network). In this lab, we will consider issues that arise when we attempt to connect many such networks together to form an internet, as pictured below.

We could connect such networks exclusively through the use of the Layer2 protocol. This plan would work fine in principle, but it suffers from a couple of practical limitations.

- Each switch would need to memorize the locations of every MAC address on the internet. This would require each switch to have an enormous amount of memory, and to engage in a time-consuming search before forwarding each frame, ultimately limiting the speed of the internet.

- There are many other layer 2 protocols (Token Ring, PPP, X.25, Frame Relay, ISDN) that may be more appropriate than ethernet for a particular network or for the infrastructure connecting networks together.
**Background: IP (Internet Protocol)**

The Internet Protocol was designed to address these issues. As you probably know, IP is based on packets (similar to Ethernet frames), and an IP packet can be transported as the payload of any layer 2 protocol.

```
+-----+-----+-----+-----+
|     |     |     |     |
|     |     | dest| src |
|     |     |     |     |
|     |     |     | payload |
|     |     |     |     |
|     |     |     |     |
+-----+-----+-----+-----+
```

*an IP packet embedded as the payload of an Ethernet frame*

Thus, IP is a layer 3 protocol, where layer 3 is the *network layer*. (In fact, IP is *the* layer 3 protocol.)

```
<table>
<thead>
<tr>
<th>Layer 1: Physical</th>
<th>Layer 2: Data Link</th>
<th>Layer 3: Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Light System</td>
<td>Ethernet</td>
<td>IP</td>
</tr>
</tbody>
</table>
```

**Background: IP Addresses**

A MAC address is a *physical address*, and plays a role similar to a person’s social security number. Why doesn't the telephone company let you call someone by dialing their social security number? That would require every telephone switch to know the location of everybody’s telephone in the world! That's why the phone company assigns you a *logical address*—a telephone number. When you dial 1-202-456-1111, the telephone switch at the other end of your phone line has no idea where to find the phone you’re calling, but it *does* know that the number starts with a 1, and is therefore a long distance call. Your call gets passed off to another switch, which looks at the 202 and connects you to a switch in Washington, DC. Another switch recognizes the 456 exchange and ultimately your call is connected to the White House by a series of switches, each of which knows only some of the information needed to route your call.
Routing telephone calls is made simple because telephone numbers are hierarchical. Likewise, IP addresses are hierarchical. IP addresses (such as 205.214.169.35) consist of 4 octets (8-bit values), where the first octets identify a network, and the last octet identifies a particular host machine. Hence, IP routers can route packets without knowing the locations of all IP addresses.

In our simplified Layer 3 (L3) implementation, we'll use 16-bit L3 addresses, consisting of two 8-bit values. For example, we'll represent the L3 address 1.3 with the binary value 00000001.00000011. We'll refer to the first part of the address as the network, and the second part as the host. Each of our L3 packets will contain a destination L3 address, source L3 address, a length, and payload, as shown in our class wiki. (The type of the payload at layer 4 will be defined in a layer 4 shim layer that is the first bits of the layer 3 payload – just like we did with layer2/layer3.

In our protocol we are going to use the universally unique Layer 2 (“MAC”) address as the host part of the L3Address. The network part of the address will be uniquely assigned to each network that we create. Combining these two into an L3Address means every L3Address will be unique, and we won’t have to create an ARP-like protocol to get the Layer 2 address for an L3Address!

**Exercise: Get Set up**

If you wish, copy the files from /home/cs/332/sp2014/lab3/* to your own lab3 working directory. (Or, you’ll may start with your implementation of lab2 if you wish.) Notice that I’ve provided all the code for lab1 and lab2 there, plus I have code there for my personal implementation of a Switch, including a GUI that shows the flashing lights on each switch port. I’ve also left you the javadoc output from my implementation of this lab, for your to use as a guide.

**Exercise: L3Address and L3Packet**

Create a new class called L3Address, which should store two ints, representing the network and host portions of an address. See the javadoc for my implementation. Implement all the methods defined in that documentation.

Now, do the same for the L3Packet.java class. Make sure you store the src and dest address fields as L3Addresses. The length field is an integer and the payload is a String.

**Exercise: L3Shim**
The L3Shim has the same basic functionality as the Layer2Handler (and upcoming L3Handler). The L3Shim layer is the layer in our model that handles the small 2-bit field that all Layer 3 protocols must define as their first two bits – to uniquely identify the type of protocol that follows those 2 bits. This is the class that will implement the multiplexing and demultiplexing of layer 3 protocols within layer 2. Using the javadoc provided, implement L3Shim.java and L3ShimListener.java. Note that my L3Shim class stores a 4-element array of L3ShimListeners. This size is hard-coded, since we have defined the L3 shim to be 2 bits wide.

**Exercise: L3Handler**

Create a new class called L3Handler, which should store an L3Address, an L3Shim, and an L3Listener. The L3Shim variable is a reference to the object “below” this layer and the L3Listener object is the object called back when a L3 packet is received at this object and needs to be passed up to the higher layer.

Implement all the methods, as documented from my implementation.

Write the L3ShimListener interface, which defines the packetReceived() method.

**Exercise: L3Listener**

Create an interface called L3Listener, which should require the implementation of the method packetReceived. This method should take in an L3Handler (the handler that received the packet) and an L3Packet (the packet itself).

**Exercise: L3Display**

Create a class L3Display that functions similarly to the Layer2Display. The documentation from my implementation explains it all.

**Exercise: Test**

Look at Test.java to see how I tested my code. Yours should work too!

Submit your code in /home/cs/332/current/<yourid>/lab3.

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