CS 332, Fall 2017, Norman Nov. 20, 2017

Lab: Layer 3

Overview

In the previous lab, we simulated layer 2 communication on a LAN (local area network). In this lab, we will build layer 3, so that (eventually) we can forward packets across multiple networks – even networks that use different Layer 2s (as in L2GA or L2GB). Note: we are NOT implementing routing in this lab, so you do not have to implement a routing table, etc. Thus, when you finish this lab, you won't be able to send packets to other networks (yet), just to other hosts on the same network.

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 8-bit L3 addresses, consisting of a 4-bit network part and a 4-bit host part. For example, we'll represent the L3 address `1.3` with the binary value `0001.0011`. Each of our L3 packets will contain the fields we decided on together, as shown in our class wiki.

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!

Get Set up

Each team member should get the `lab2` files from an agreed-upon person – probably a person who got a very good grade on lab2. You might have this agreed-upon person fix any bug that was found by your professor before you make the copies.

Then, go look at `cs.calvin.edu/courses/cs/332/schedule.html` where in Week 10 you will find a link to javadoc output from my implementation of this lab. This Javadoc output should be used as a guide for this lab. <u>Each person on your team MUST work independently on her/his code, but you SHOULD test your implementations against each other to make sure they are interoperable.</u>

Do the following steps.

L3Address and L3Packet

Create a new class called `L3Address`, which should store two `int`s, 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.

L3Shim

The `L3Shim` in my old implementation has the same basic functionality as the `Layer2Handler` (and upcoming `L3Handler`). The `L3Shim` layer is the layer in my model that handles the small 2-bit field that all Layer 3 protocols had to define as their first two bits – to uniquely identify the type of protocol that follows those 2 bits. This is the class that will implemented the multiplexing and demultiplexing of layer 3 protocols within layer 2. You need to implement similar functionality in your layer 2 so that you demultiplex on the type field in layer 2, passing L3 packets up to your L3Listener, but making sure that layer 2 packets with a different type value are dropped.

L3Handler

Create a new class called `L3Handler`, which should store an `L3Address`, an `L2GXHandler,` and an `L3Listener`. The `L2GXHandler` 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.

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).

L3Display

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

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`. Please, please call the directory `lab3`, not `Lab3`, or `Lab 3`.

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