Lab: Layer 2

Background: Frame of Reference

In the previous lab, we learned how to broadcast bits by switching a light bulb on and off. This light bulb system constitutes the physical layer of our simulated network protocols, much like the cables, hubs, wireless transmitters, etc., form the physical layer we use to connect to the Internet. This physical layer is referred to as layer 1. The BitHandler class you developed already was the first step on our journey into layer 2: the data link layer. The most widespread example of a layer 2 protocol is ethernet. In this lab, we'll implement our own version of the ethernet protocol.

So far we have used our BitHandler class to broadcast meaningless strings of bits. When we receive such a string, how can we know if it was meant for us, or who it came from, or if we have correctly identified the same bits intended by the sender?

At layer 2, a bit string is called a frame, and ethernet answers these questions by requiring certain information to be provided in designated parts of the frame (similar to the way addresses must appear on an envelope used to mail a letter). In our protocol, a frame will appear as specified in our wiki on moodle. Please take a look at that now.

The leading 0 is simply used for synchronization purposes, merely indicating the beginning of a frame.

Although each node on a LAN (segment) can see every frame that is sent, it is expected that only the designated recipient will examine the contents of the frame. Occasionally, however, it will be useful to designate that a frame is intended for all nodes on the network. The special destination address 11111111 will be used for this purpose.

To determine that a frame has been transmitted and read correctly, some redundant information is tacked on to the end of a frame, called a cyclic redundancy check (or checksum). In our protocol, we call our field the “parity” field. To compute the value, we'll find the total number of 1s in the frame and payload. The remainder when dividing this number by 256 will give us our parity value.

Exercise 1: Framed

Copy your files from lab1 into a new directory. In this directory, create a new file/class called Layer2Frame, which should store an instance variable for each field in our layer 2 protocol: the source and destination MAC addresses (probably stored as integers),
vlanId, length, etc., along with a payload String. Provide a constructor that takes in 4 values: destMac, srcMac, vlanId, and payload. The constructor can then compute the length and parity values. Also create getters for destMac, srcMac, vlanId, and payload.

In `Layer2Frame`, write a static method `toBinary` that takes in an integer value and an integer length, and returns a string of bits of the given length representing the given value in binary. You may assume that the value will fit in the number of bits given by length.

In `Layer2Frame`, write a static method `computeParity` that takes in a bit String, and a modValue, and returns the 8-bit String value for that string (the binary number representing the remainder when the number of 1s in the bit string is divided by modValue).

Create `Layer2Frame`'s `toString` method to create and return the bit string corresponding to this frame. Don’t forget to prepend the “0” to the beginning of it.

Finally, I’ve created a public static int constant called `BCAST_ADDR` and initialized it to the 8-bit all-1s value. This is going to be useful later.

At this point it would probably be wise to test your code in `Layer2Frame` to see if it creates legal-looking frames. Just comment out the code in `Test.java` and put in new code to create some `Layer2Frames` and print out the results of calling `toString()` on them.

**Exercise 3: Getting a Handle on Ethernet**

Create a file/class called `Layer2Handler`. It stores a variable `handler` of type `BitHandler`.

Create the class constructor to take a String `host`, integer `port`, and integer `macAddr`. The constructor creates a new `BitHandler`, passing in the host and port. The constructor must then set itself (“this”) as a listener of the `BitHandler`. (This is how we make the connection from layer 1 to layer 2.) Store the `macAddr` in an instance variable. Create a getter for the `macAddr`.

Create the `toString` method so as to return a string representation of the `macAddr` value.

Create a second constructor that takes only `macAddr` and uses default values for the other 2 parameters. Call the other constructor with these values. (See `BitHandler` for an example of this.)
**Exercise 5: Picture Frame**

Copy `Layer2Display.java` from `/home/cs/332/sp2014/lab2/`. The constructor creates a couple of fields that can be used to create a frame and then send it. Amend this code so we can create a value for `vlanId`. (Everything else should be good: the `Layer2Handler` will compute the length and parity fields from the other data.) Also, fix the code so that the mac address of this display’s handler is shown in the title of the `JFrame`.

Replace the comment `/* SEND LAYER2 FRAME HERE */` in `actionPerformed` with code for sending the appropriate frame. Do this by calling the `Layer2Frame()` constructor and then using the handler to send the frame. (Note that this code won’t compile yet, as there are many missing pieces – including the `send()` method in the handler!)

Now, go to `Layer2Handler.java` and implement the `send()` method. This method takes a `Layer2Frame` and converts it to its string representation. Next, the code should repeatedly wait for `BitHandler.HALFPERIODs` until the lower layer handler is silent. When it is silent, it calls the handler’s `broadcast()` method to send the packet. Note that with our implementation you actually don’t have to worry about collisions coming up from the lower layer. If your code waits until the handler is `isSilent()`, then start sending, you won’t see a collision.

**Exercise 6: Parsing a Received Layer2 Frame**

The code for generating a Layer 2 frame is done. But, we need code to receive a frame.

Add a constructor to `Layer2Frame` that takes in a string of bits and parses them to find (and store) the frame’s addresses, vlan, length, and payload. (Writing a `toDecimal` method will be quite helpful here.) This constructor should also look for errors in the packet – does it start with the required “0”? Is the length correct? Is the parity field correct? The code should throw an exception (`IllegalArgumentException`) if the packet is invalid for any reason. (The code does NOT check if the destination address is correct, however. That will be done elsewhere.)

**Exercise 7: Hark! Layer2 Speaketh**

Create a file/interface called `Layer2Listener` that has the following method implemented.

```java
void frameReceived(Layer2Handler h, Layer2Frame f)
```
Edit Layer2Handler so that it stores a (single) Layer2Listener. Add an API so that a listener can be registered: similar to how it is done in BitHandler. We’ll make Layer2Display be a Layer2Handler listener in the next step.

First, however, we need a way for Layer 2 to receive a packet (from Layer 1). I.e., we have to make our Layer2Handler be a listener of Layer 1. Look at BitDisplay.java and notice how it registers itself to be a BitHandler listener. Add similar code to Layer2Handler to be a BitHandler listener. (Notice that this code only supports one listener for an object… At some point we may want to generalize this so that we can have a list of listeners.)

Whenever it receives bits, the Layer2Handler will parse those bits into a Layer2Frame. Then, it should check if the frame is for itself. If not, it drops the frame and goes on. If the frame is for itself, it checks if there is a listener registered, and if so, passes that frame up to its Layer2Listener, by calling the listener’s frameReceived method. Here is the big picture (in this picture, replace in your mind the word Ethernet with Layer2).

![Diagram of Layer2Frame processing](diagram.png)

Finally, modify the Layer2Display class so that it listens for layer2 frames. When a frame is received, the code simply builds up a string and displays it in the displayField field. You may make this text as nice as you want.
(In fact, if you want to make the Layer2Display much fancier, be my guest. It could, e.g., have fields that are used for generating frames, and separate fields that display the values of fields in received frames.)

**Testing**

Update your Test.java file so that you can test if your code sends and receives layer 2 frames correctly.

Submit all your code in `/home/cs/332/current/<yourid>/lab2`. Recall that Prof. Norman is a, how shall we say it kindly, “stickler” for documentation and clean code.

**Honors: Create A Layer2 Switch**

Create a new class called `Switch`. Write a method called `addSegment`, which should take in the IP address and port number of a light system, and create a `Layer2Handler` for that segment. Now go ahead and complete the `Switch` class, so that it receives frames and forwards them according to the algorithm we read about in the book. Make sure that your `Switch` also times out entries in your port# → MACaddr cache.

To test your `Switch`, you'll need to create multiple `LightSystems`. You can run multiple `LightSystems` on the same computer by constructing each `LightSystem` with a different port number. For example, you might run one on `LightSystem.DEFAULT_PORT` and another on `LightSystem.DEFAULT_PORT + 1`. 