piBeacon - DIY Beacon with a Raspberry Pi

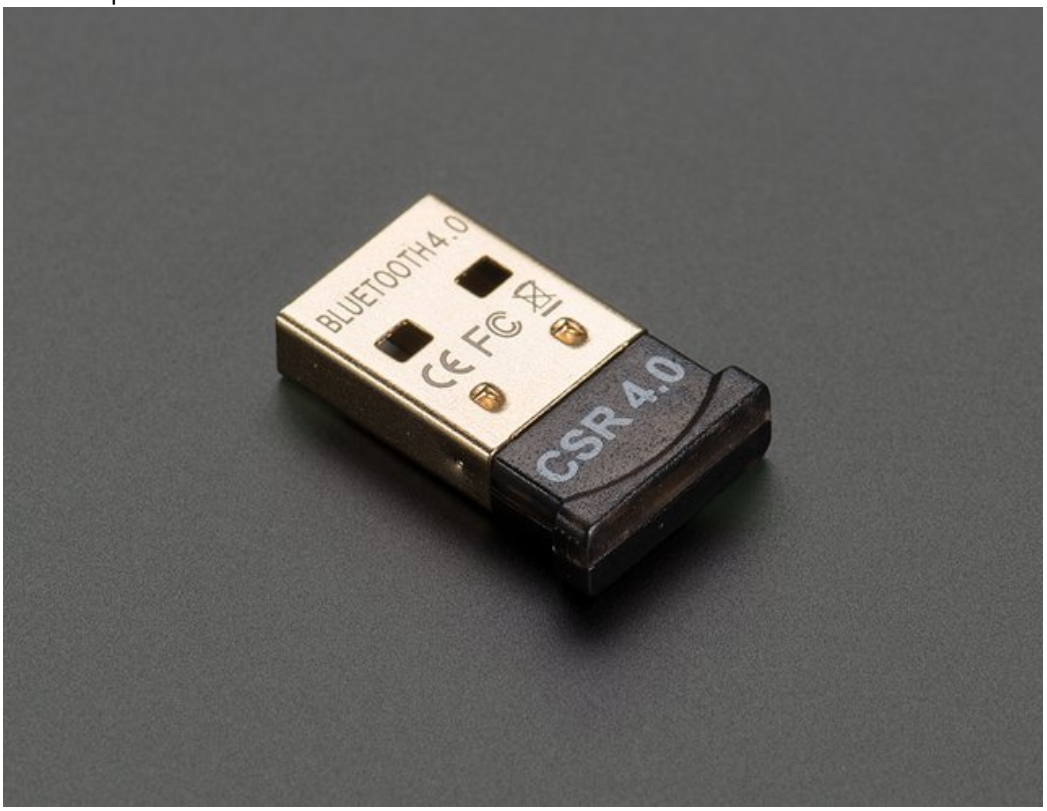
Created by Kevin Townsend

Last updated on 2016-09-14 06:14:32 AM UTC

Guide Contents

Overview

This learning guide will show you how you can take your Raspberry Pi (or almost any Linux-based device with a bit of poking and prodding) and turn it into an Beacon node using our Bluetooth 4.0 USB Module (http://adafru.it/1327) and the open source Bluez stack.



What You'll Need

- A Raspberry Pi (http://adafru.it/998) (any model should be OK)

- A Bluetooth 4.0 USB Module (http://adafru.it/1327) (not every module works with Bluez, though ours definitely does!)
- A iOS 7.0 based device (recent iPhone/iPad/iPod Touch) to test with
- An iBeacon reader app from the App Store to test with (TryLocate Beacon (http://adafru.it/rnD), but any free iBeacon watcher ought to be OK!)

Acknowledgement

A big thanks to Tony Smith at The Register for putting together histutorial (http://adafru.it/cYr) on configuring Bluez to transmit Beacon data!

What is a 'Beacon'?

'Beacons' are based on Bluetooth Low Energy (part of the new Bluetooth 4.0 standard), and at it's heart is a way to advertise location specific data one-way, or provide basic indoor navigation via individual Beacon nodes.

The way it works is actually very simple. Any BLE device typically advertises a certain amount of data to let other devices (like your phone) know that they exist and they're available. The advertising packet that these devices send out might include information like key services offered by the device, a human-readable short devices name, etc.

Beacons take this short advertising frame, and appends a custom payload in the "Manufacturer Specific Data" field which includes a unique 128-bit UUID to identify companies or unique entities, as well as two 16-bit values ('Major' and 'Minor', or whatever you'd like to call them) that allow you to differentiate specific stores/premises (Major) and individual Beacon nodes (Minor).

That's basically it. All the rest of the magic is on the app side where your phone listens for these advertising frames, and when it detect something it estimates the distance to the node and displays some sort of alert.

It's terribly simple, but that's probably what makes it so interesting and also so inexpensive to implement!

How Does it Work in Pratice?

Essentially, all you need to do is insert a specific set of bytes into the optiona**Manufacturer Specific Data (http://adafru.it/cYs)** field of the advertising packet on your Bluetooth Low Energy device.

Inside this field, you need the following values:

- **ID** (uint8_t) - This will always be 0x02
- **Data Length** (uint8_t) - The number of bytes in the rest of the payload = 0x15 (21 in dec)
- **128-bit UUID** (uint8_t[16]) - The 128-bit ID indentifying your company/store/etc
- **Major** (uint16_t) - The major value (to differentiate individual stores, etc.)
- **Minor** (uint16_t) - The minor value (to differentiate nodes withing one location, etc.)
- **TX Power** (uint8_t) - This value is used to try to estimate distance based on the RSSI value

For example, the following is a valid iBeacon payload (separators added for clarity sake):

02 | 15 | E2 0A 39 F4 73 F5 4B C4 A1 2F 17 D1 AD 07 A9 61 | 00 00 | 00 00 | C8

The only other missing piece is that, following the Bluetooth standard, the Manufacturer Specific Data needs to be preceded by the Company Identifier (http://adafru.it/cYt). The company identifier for Apple, for example, is**0x004C**, which we'll use for the example above.

Compiling Bluez

In order to use your Raspberry Pi to send out Beacon data in the advertising frame, we'll need to install a few open source tools, mainly **Bluez**

Check if you already have Bluez

If you already have a modern verison of Bluez you do not need to do this step!

On your Raspberry Pi, try running

```
sudo apt-get install bluez
```

and then

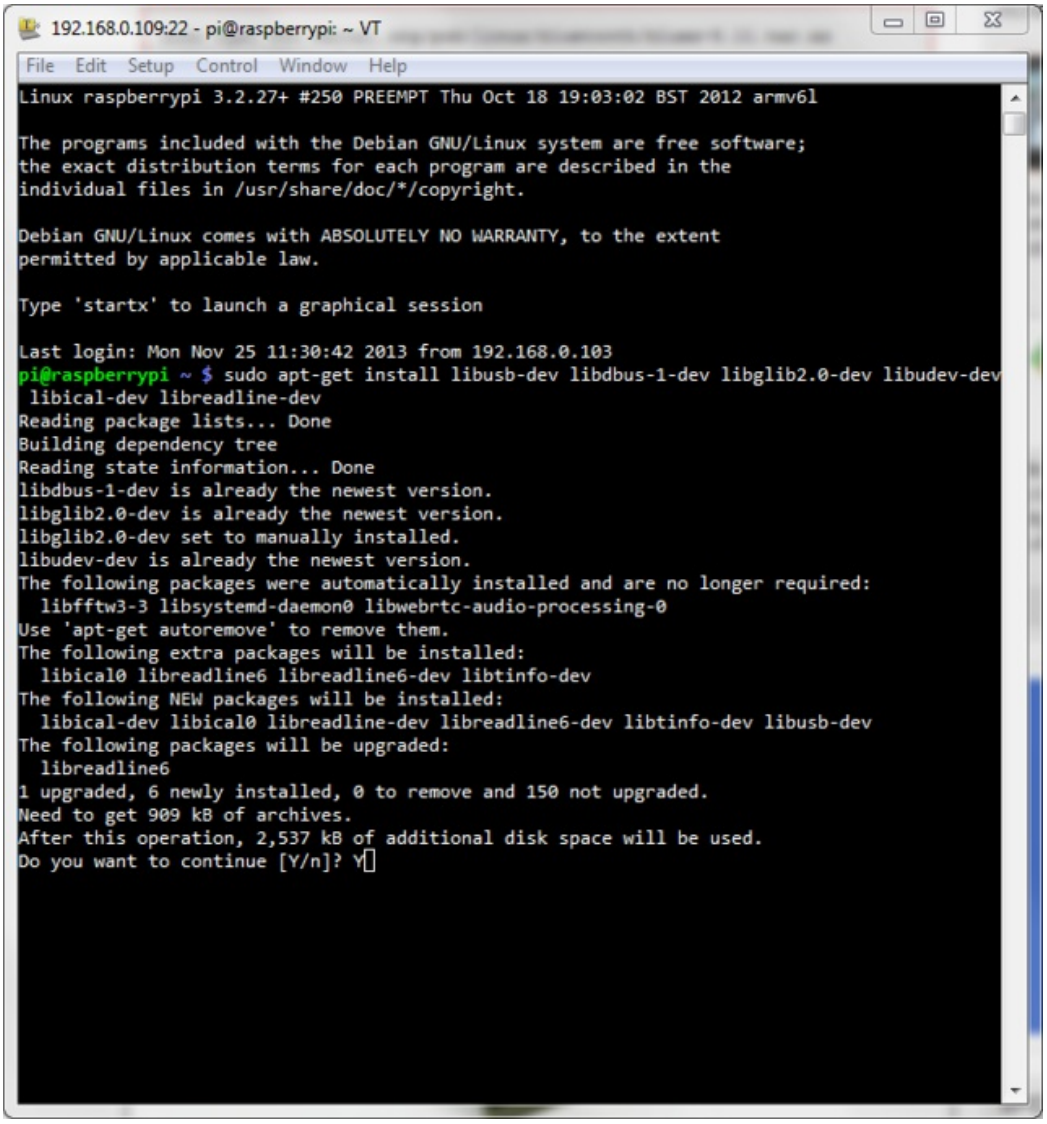
```
dpkg --status bluez | grep '^Version:'
```

If you get something like **Version: 5.23-2+rpi2**
Where the version is greater than 5.11 you can skip this step.

1. Install Required Libraries

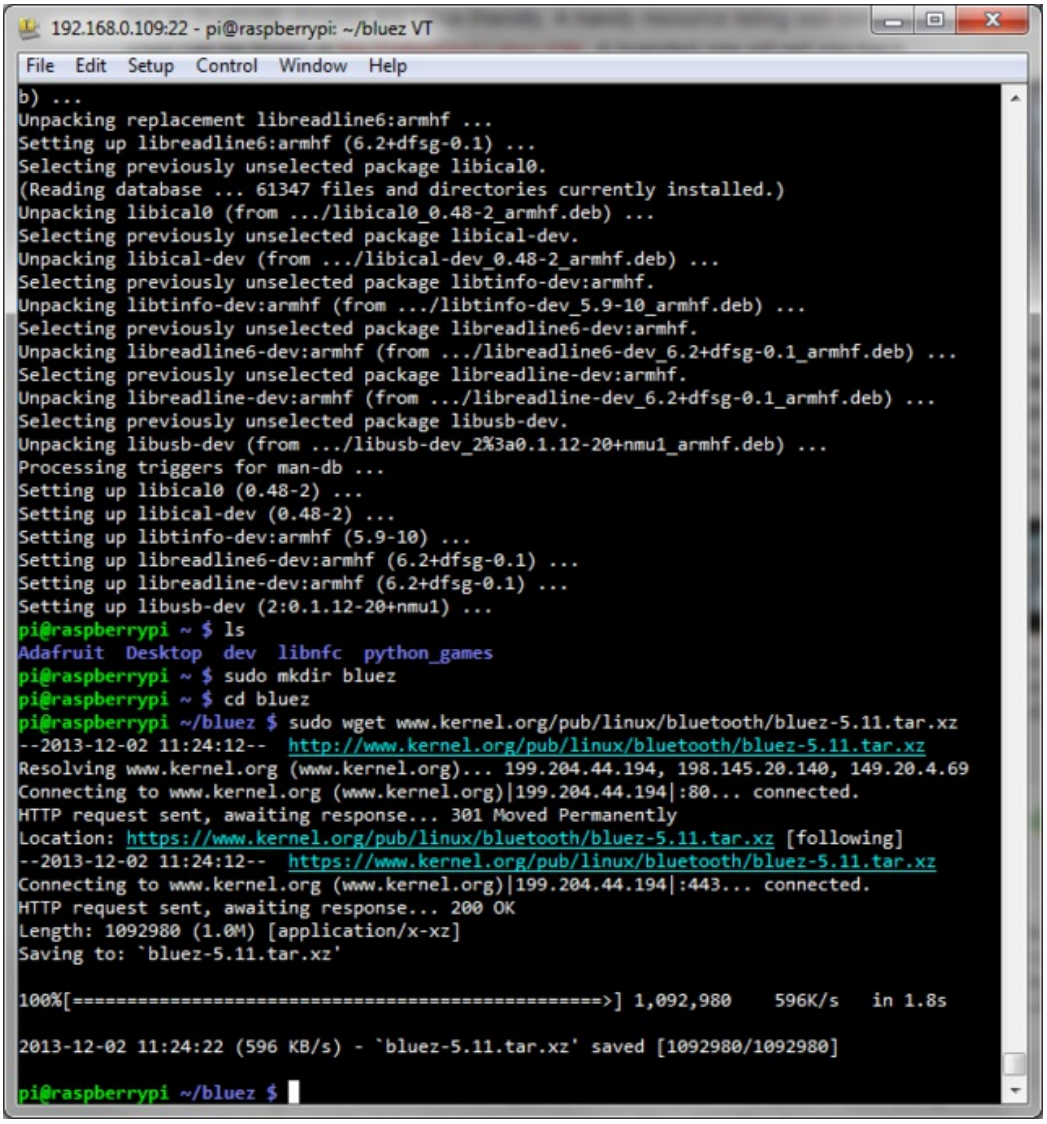
```
sudo apt-get install libusb-dev libdbus-1-dev libglib2.0-dev libudev-dev
sudo apt-get install libical-dev
sudo apt-get install libreadline-dev
```

You may need to type the above code in to make it work rather than doing a copy/paste, or manually remove a superfluous line feed between lines



2. Download Bluez

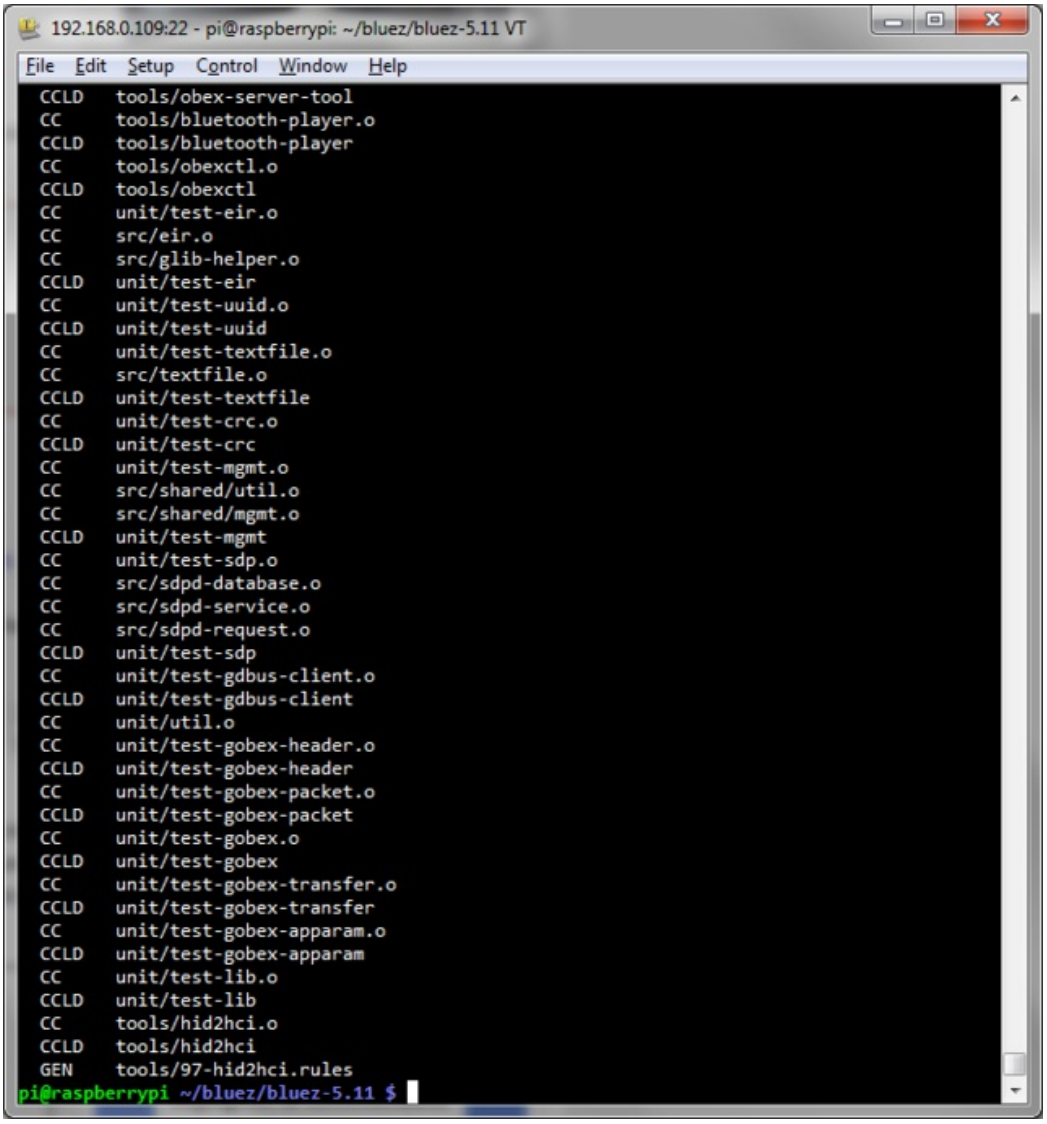
sudo mkdir bluez
cd bluez
sudo wget www.kernel.org/pub/linux/bluetooth/bluez-5.11.tar.xz



3. Unzip and Compile Bluez

Next you need to actually build Bluez on the Pi. This step may take a while, but should work without any hiccups if you properly installed all the libraries in step one above:

```
    sudo unxz bluez-5.11.tar.xz
sudo tar xvf bluez-5.11.tar
cd bluez-5.11
sudo ./configure --disable-systemd
sudo make
sudo make install
```



4. Insert the USB Module and Reset

Once Bluez has been built, shut down your computer with **sudo shutdown -h now** and once its Halted, **insert your Bluetooth 4.0 USB Module (http://adafru.it/1327) and then restart the Raspberry Pi** so that all of the changes we have made can take effect.

Adding Beacon Data

In the '**bluez-5.11**' folder that we previously created, we can start entering the mandatory Beacon data and advertising it using **hcitool**, which we built when compiling Bluez.

1. Check for your USB Module

This should give you a list of devices on your system:

tools/hciconfig (if you compiled bluez)

or

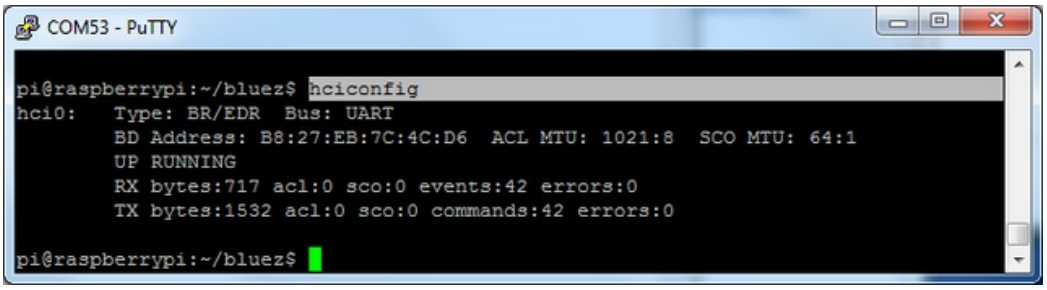
hciconfig (if you apt-get'd bluez)

If everything is properly configure you will see your Bluetooth 4.0 USB Module like this:

```
pi@raspberrypi ~/bluez/bluez-5.11 $ tools/hciconfig
hci0:   Type: BR/EDR  Bus: USB
        BD Address: 00:1A:7D:DA:71:12  ACL MTU: 310:10  SCO MTU: 64:8
        DOWN
        RX bytes:495 acl:0 sco:0 events:21 errors:0
        TX bytes:89 acl:0 sco:0 commands:21 errors:0

pi@raspberrypi ~/bluez/bluez-5.11 $
```

On a Raspberry Pi 3 you'll see the Bus is UART, not USB!



2. Enable the USB Device

Next you can enable the device with the following commands, turning off device scanning since this can cause problems when advertising.

If you're using the compiled bluez, add tools/ before each call to hciconfig

```
sudo hciconfig hci0 up
sudo hciconfig hci0 leadv 3
sudo hciconfig hci0 noscan
```

Then run the hciconfig tool again and you should see that the device is marked as **UP** and **RUNNING**:

```
hciconfig
```

```
pi@raspberrypi ~/bluez/bluez-5.11 $ sudo tools/hciconfig hci0 leadv
pi@raspberrypi ~/bluez/bluez-5.11 $ sudo tools/hciconfig hci0 noscan
pi@raspberrypi ~/bluez/bluez-5.11 $ tools/hciconfig
hci0:   Type: BR/EDR  Bus: USB
        BD Address: 00:1A:7D:DA:71:12  ACL MTU: 310:10  SCO MTU: 64:8
        UP RUNNING
        RX bytes:1008 acl:0 sco:0 events:45 errors:0
        TX bytes:236 acl:0 sco:0 commands:45 errors:0
```

3. Enter the Beacon Advertising Data

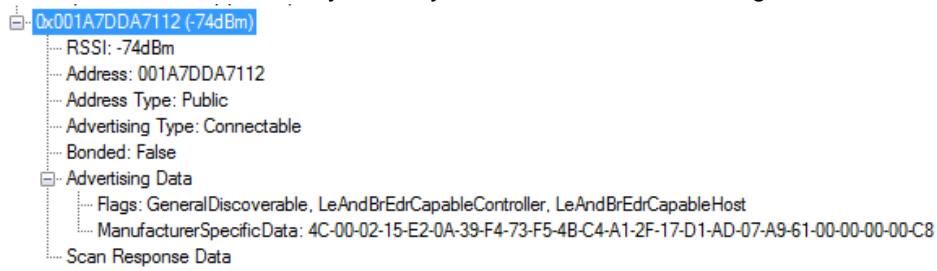
The last thing to do is to enter the Beacon advertising data, which we can do with the following command (which should all be on one line):

sudo hcitool -i hci0 cmd 0x08 0x0008 1E 02 01 1A 1A FF 4C 00 02 15 E2 0A 39 F4 73 F5 4B C4 A1 2F 17 D1 AD 07 A9 61 00 00 00 00 C8 00

FF identifies the start of the Manufacturer Specific Data,**4C 00** is Apple's company ID (0x004C), and then you can see the rest of the Beacon payload until **C8**.

Results on a Bluetooth Debugger

Just to show that this actually works, you can see the results using a Bluetooth Low Energy sniffer below:

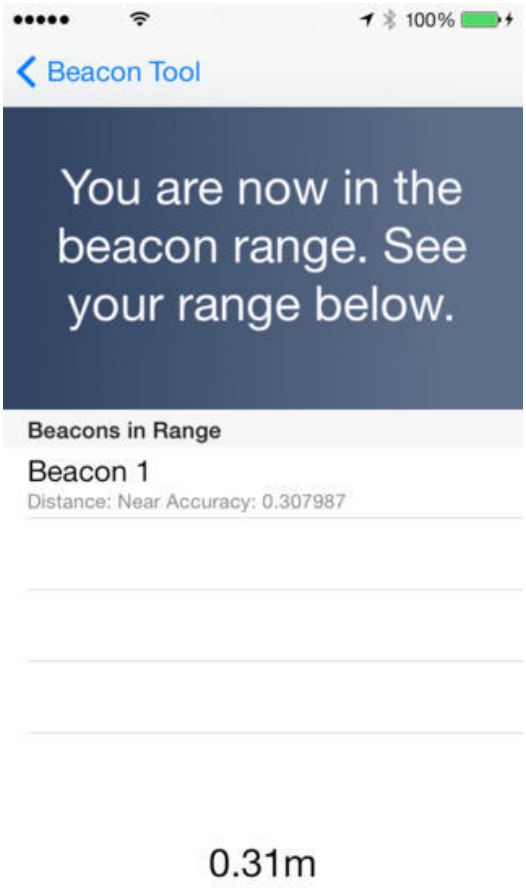


And here is the raw advertising frame from a different sniffer:

Testing it on iOS

To test that this actually works you'll need an iOS 7 based iPad/iPhone/iPod Touch, and a Beacon application.

Start the app up, going into 'Listen' mode, and you should see a screen similar to the capture below, where the range will go in and out depending on your proximity to the node:



Beacon Toolkit only searches for specific Beacon UUIDs -- the same UUID used in this tutorial. If you use a different UUID, you will have to find a different tool to test on your iOS device.