Raspberry Pi SDR IGate

Last update ~~11/9/2015~~

7/3/2016 – Added note about /etc/modprobe.d/raspi-blacklist.conf.

9/22/2016 – Clarify IGate server rotate addresses.

5/28/2017 – Details about frequency calibration.

Warn against cheap versions with terrible frequency accuracy.

It's easy to build a receive-only APRS Internet Gateway (IGate) with only a Raspberry Pi and a software defined radio (RTL-SDR) dongle. Here's how.

Hardware Required

- **Raspberry Pi**

  I happened to use the model 2 so I can't say, with certainty that the earlier models would be fast enough to keep up. "top" shows about 93% cpu idle time so the older models are probably more than adequate.

  The procedure here is known to work with the Raspbian operating system. This was originally written for Wheezy and later tested with Jessie. Some adjustments might be required for other operating systems.

- **SDR Dongle**

  This connects to the USB port and an antenna. This is the one I used.

  http://www.amazon.com/NooElec-RTL-SDR-RTL2832U-Software-Packages/dp/B008S7AVTC

  There are many others that appear to be equivalent such as
  https://www.adafruit.com/products/1497

  I RECOMMEND THAT YOU AVOID THESE. Things have changed over the past couple years. The earlier (now cheaper) devices have terribly inaccurate frequency references. Mine is off by about 60 Parts Per Million (PPM) which comes out to around 8 kHz on 2 meters or 25 kHz on 70 cm.

Newer devices now claim to have an accuracy of 1 PPM or even 0.5 PPM and a temperature compensated oscillator (TXCO). I know hams are cheap, but spend the extra $4 on one of the newer models and avoid the aggravation of calibration and drifting.

Lots of great information here: http://www.rtl-sdr.com/about-rtl-sdr/

Recommendations on what to buy: http://www.rtl-sdr.com/buy-rtl-sdr-dvb-t-dongles/

Software Required

- **Dire Wolf**

  Install following the instructions in **Raspberry-Pi-APRS.pdf.**

  You can stop at the section called **Interface for Radio**. Here we are using the SDR dongle rather than a USB audio adapter.

  Don't worry about the configuration part because we will build our own configuration file here.

- **RTL-SDR** Library from http://sdr.osmocom.org/trac/wiki/rtl-sdr

Install it like this:

```
sudo apt-get update
sudo apt-get install cmake build-essential libusb-1.0-0-dev
cd ~
git clone git://git.osmocom.org/rtl-sdr.git
cd rtl-sdr
mkdir build
cd build
cmake ../ -DINSTALL_UDEV_RULES=ON -DDETACH_KERNEL_DRIVER=ON
make
sudo make install
sudo ldconfig
```

Wheezy:

I didn't find it to be necessary, with the software version I was using, but community feedback suggests adding the following to **/etc/modprobe.d/raspi-blacklist.conf** to prevent the use of undesired device drivers.

```
blacklist dvb_usb_rtl28xxu
blacklist dvb_usb_v2
blacklist rtl_2830
blacklist rtl_2832
blacklist r820t
```

Jessie:

The original empty /etc/modprobe.d/raspi-blacklist.conf seems to be fine.

Configuration

We need to construct a configuration file for Dire Wolf. Take the following text and put it into a text file, called **sdr.conf**, in your home directory. You might find a copy of this file in /usr/share/doc/direwolf/examples.

```
#
# Sample configuration for SDR read-only IGate.
#

# We might not have an audio output device so set to null.
# We will override the input half on the command line.
ADEVICE null null
CHANNEL 0

# Put your callsign in place of xxx below.
MYCALL xxx

# Pick appropriate servers for your geographical region.
#
#       noam.aprs2.net          - for North America
#       soam.aprs2.net          - for South America
#       euro.aprs2.net          - for Europe and Africa
#       asia.aprs2.net          - for Asia
#       aunz.aprs2.net          - for Oceania
#
# Change the following line if you are not in North America.
IGSERVER noam.aprs2.net

# You also need to specify your login name and passcode.
# This is the same passcode you would use with any other IGate
# application.  Contact the author if you can't figure out
# how to generate the passcode.

IGLOGIN xxx 123456

# That's all you need for a receive only IGate which relays
# messages from the local radio channel to the global servers.
```

Put your callsign and optional SSID in the two places that have xxx above. Put your actual passcode in place of 123456.

Choosing the IGate Server

Which server should you use? The current preferred way is to use one of these regional rotate addresses:

- noam.aprs2.net - for North America
- soam.aprs2.net - for South America
- euro.aprs2.net - for Europe and Africa
- asia.aprs2.net - for Asia
- aunz.aprs2.net - for Oceania

Each name has multiple addresses corresponding to the various servers available in that region. Why not just specify the name of one specific server? This approach offers several advantages:

- Simplicity – You don't need to change your configuration as new servers become available or disappear.
- Resilience – If your current server becomes unavailable, another one will be found automatically.
- Load balancing – Picking one at random helps to spread out the load.

Visit http://aprs2.net/ for the most recent information.

Run It

```
rtl_fm -f 144.39M - | direwolf -c sdr.conf -r 24000 -D 1 -
```

Note the "-" at the end of the line which means read audio from stdin. Alternatively this could have been done in the configuration file, using either of these forms:

```
ADEVICE stdin null
ADEVICE - null
```

You should now have a functioning receive only IGate.

Automatic Startup on Reboot

You might be tempted to put the command line above into /etc/rc.local. It's not that simple and there are some disadvantages. First, it runs as root. It's best to avoid running as root, when possible, because it's harder to accidentally trash your system. When /etc/rc.local is running, $PATH is set to /sbin:/usr/sbin:/bin:/usr/bin so it wouldn't find anything in /usr/local/bin. You would need to specify the full path. The current working directory and $HOME are both / so you would need to put the

configuration file there or specify a different location. The text output, for troubleshooting, is not readily available. Finally, it there is a failure, it won't restart until the next reboot.

My suggestion would be to modify the included **dw_start.sh** script so that it also runs rtl_fm. Look for this line and remove the # at the beginning.

```
#DWCMD="bash -c 'rtl_fm -f 144.39M - | direwolf -c sdr.conf -r 24000 -D 1 -'"
```

Run "crontab –e" to edit your crontab file. Assuming that you are running as user "pi" and you have a copy of dw-start.sh in the home directory, add a line like this:

```
* * * * *  /home/pi/dw-start.sh  >/dev/null  2>&1
```

This script will run once per minute. Dire Wolf is started automatically if not running already. If it crashes, or is terminated for any other reason, it will be restarted. A log of restarts can be found in `/tmp/dw-start.log`.

The User Guide contains more discussion about automatic start up.

Further Reading

Additional information can be found in the documentation directory:
https://github.com/wb2osz/direwolf/blob/dev/doc/README.md

- **Raspberry-Pi-SDR-IGate.pdf**

  The most recent version of this document.

- **Raspberry-Pi-APRS.pdf.**

  The Raspberry Pi has some special considerations that make it different from other generic Linux systems. Start here if using the Raspberry Pi, Beaglebone Black, cubieboard2, or similar single board computers.

- **User-Guide.pdf**

  This is your primary source of information about installation, operation, and configuration.

- **Successful-APRS-IGate-Operation.pdf**

  Dire Wolf can serve as a gateway between the APRS radio network and APRS-IS servers on the Internet.

  This explains how it all works, proper configuration, and troubleshooting.

Calibration

The older / cheaper RTL SDR dongles might be several kHz off and drift even more with temperature variations.

In one test, I found that it worked best when I specified a frequency 8 kHz lower than the desired frequency. That translates into about 56 parts per million (PPM).

As I mentioned near the beginning, there are now devices that claim to have 1 PPM or better accuracy and temperature compensated oscillators. If you are buying a new one, spend the extra $4 and avoid the aggravation that follows.

There are multiple ways to figure out the frequency error.

SDR#

Tune into a station of known frequency, and select the gear icon near the top left. Adjust the "frequency correction" setting until the signal strength peak lines up with the desired frequency. I used a nearby weather station at 162.525 MHz, figuring this would be more accurate than most amateur radio transmissions. The best setting turned out to be 61 or 62 PPM. That's pretty close to my trial & error estimate.

Easy Way

"rtl_test" can give us a crude measurement, based on the computer's clock.

```
$ rtl_test -p60
Found 1 device(s):
  0:  Realtek, RTL2838UHIDIR, SN: 00000001

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Supported gain values (29): 0.0 0.9 1.4 2.7 3.7 7.7 8.7 12.5 14.4 15.7
16.6 19.7 20.7 22.9 25.4 28.0 29.7 32.8 33.8 36.4 37.2 38.6 40.2 42.1
43.4 43.9 44.5 48.0 49.6
Sampling at 2048000 S/s.
Reporting PPM error measurement every 60 seconds...
Press ^C after a few minutes.
Reading samples in async mode...
real sample rate: 2048132 current PPM: 65 cumulative PPM: 65
real sample rate: 2048124 current PPM: 61 cumulative PPM: 63
real sample rate: 2048129 current PPM: 63 cumulative PPM: 63
real sample rate: 2048130 current PPM: 64 cumulative PPM: 63
real sample rate: 2048132 current PPM: 65 cumulative PPM: 63
real sample rate: 2048130 current PPM: 64 cumulative PPM: 64
real sample rate: 2048131 current PPM: 64 cumulative PPM: 64
```

```
real sample rate: 2048130 current PPM: 64 cumulative PPM: 64
real sample rate: 2048129 current PPM: 63 cumulative PPM: 64
real sample rate: 2048125 current PPM: 61 cumulative PPM: 63
real sample rate: 2048138 current PPM: 68 cumulative PPM: 64
```

That's quite close to the SDR# result so that seems good.

https://medium.com/@rxseger/sdr-calibration-via-gsm-fcch-using-kalibrate-and-lte-cell-scanner-on-rtl-sdr-and-hackrf-193a7fb8a3eb is not fond of that and concentrates on using "kalibrate."

More Accurate Way

For a more accurate assessment, we will use "kalibrate" which looks for cell phone towers (that have very strict frequency standards) and calculates an error from that.

```
sudo apt-get install automake
sudo apt-get install libtool
sudo apt-get install libfftw3-dev
sudo apt-get install librtlsdr-dev
sudo apt-get install libusb-1.0.0-dev
git clone https://github.com/steve-m/kalibrate-rtl
cd kalibrate-rtl
./bootstrap
./configure
make
sudo make install
```

Now run it. You might have to try different bands until you find something.

```
$ kal -v -s GSM-R
Found 1 device(s):
  0:  Generic RTL2832U OEM

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Exact sample rate is: 270833.002142 Hz
kal: Scanning for E-GSM-900 base stations.
channel detect threshold: 12828.180924
E-GSM-900:
 chan: 991 (928.4MHz + 8.456kHz) power: 14547.02
```

After finding a nearby station, or two, we can we can get a measurement of how far off we are.

```
$ kal -c 991
Found 1 device(s):
  0:  Generic RTL2832U OEM

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
```

```
Exact sample rate is: 270833.002142 Hz
kal: Calculating clock frequency offset.
Using E-GSM-900 channel 991 (928.4MHz)
average   [min, max] (range, stddev)
+ 6.523kHz  [-1040, 8576]  (9615, 2895.904541)
overruns: 0
not found: 828
average absolute error: -7.026 ppm
```

```
$ kal -c 1002
Found 1 device(s):
  0:  Generic RTL2832U OEM

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Exact sample rate is: 270833.002142 Hz
kal: Calculating clock frequency offset.
Using E-GSM-900 channel 1002 (930.6MHz)
average   [min, max] (range, stddev)
- 33.280kHz  [-34333, -24720]  (9613, 2772.821289)
overruns: 0
not found: 652
average absolute error: 35.762 ppm
```

```
$ kal -c 963
Found 1 device(s):
  0:  Generic RTL2832U OEM

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Exact sample rate is: 270833.002142 Hz
kal: Calculating clock frequency offset.
Using GSM-R-900 channel 963 (922.8MHz)
average   [min, max] (range, stddev)
- 4.631kHz  [-33793, 10213] (44006, 15451.044922)
overruns: 0
not found: 48
average absolute error: 5.018 ppm
```

Based on that, is it -7 or 36 or 5? None are close to earlier measurements.

The alleged better technique was a big disappointment (to put it mildly) in my limited experience.

Automatic Method

Here is a script that will perform automatic calibration periodically:
https://github.com/khaytsus/direwolf-init

Let's try it and see what happens. The strongest weather station around here is at 162.525 MHz.

```
$ rtl_power -c .1 -f 162.515M:162.535M:64 -i 2 -g 10 -e 10m noaa.csv
$ findppm.pl 0 162525000 noaa.csv
Best PPM is -29
```

That doesn't make sense.

What if we start off with our earlier estimate and see if we can refine it with this?

```
$ rtl_power -c .1 -f 162.515M:162.535M:64 -i 2 -g 10 -p 60 -e 10m noaa2.csv
$ findppm.pl 60 162525000 noaa2.csv
Best PPM is 31
```

I don't know what to make of that.

Calibration - Conclusion

My initial trial & error guess for best results gave me about 56. Rtl_test came up with 63 or 64. SDR# says 61 or 62. The others are all over the place and don't make sense. I'm going with 62.

Once you have determined the frequency error, in PPM, add it to the "rtl_fm" command with the "-p" option. e.g.

```
$ rtl_fm -p 62 -f 144.39M - | direwolf -c sdr.conf -r 24000 -D 1 -
```

Or you could just buy newer device, with better accuracy, and not waste time with this.