The
Broadcasters' Desktop Resource
www.theBDR.net ... edited by Barry Mishkind – the Eclectic Engineer

Solutions

A Portable Raspberry Pi-based Interference Monitor

By Dan D'Andrea

[April 2021] When a listener cannot hear a station due to interference of any kind, it is a serious issue.

After all, listeners are ratings, and ratings are money.

Tracking down any intermittent problem can be a very frustrating task. Complaints of intermittent interference are particularly irksome, especially when they seem to be localized in an area inconvenient to monitor.

In this article, we will take a look at using a Raspberry Pi and an RFEngineers' *Watch Dog* for a broadcast engineering-related automation.

The project leads to a portable, battery-powered system, using a Python program and a few inexpensive components, along with the Raspberry Pi computer and the RFEngineers' *Watch Dog* receiver.

Once constructed, the system can be placed in a listener's home, place-of-work, or other location. It will then automatically gather data useful for tracking down the source of interference.

THEORY OF OPERATION

In this system the Pi is used to tune the *Watch Dog* to a desired frequency.

Then, an indicator of interference – the signal-to-noise ratio (SNR) for example – is monitored. If the SNR drops below the user defined threshold, interference is present. When such interference is detected the Pi records a few seconds of the station's audio to a WAV file, and station parameters, RSS, SNR, multipath, date and time, are written to a log file.

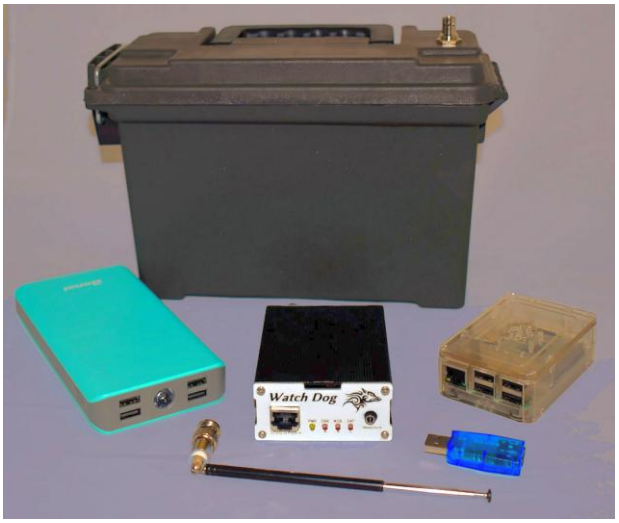
The process is repeated by re-tuning the *Watch Dog* to the First and Second Adjacent channels respectively. The *Watch Dog* is then re-tuned to the center frequency. If the station SNR is still below threshold the process is repeated.

After gathering a day or two of data the system can be relocated and run again. At the end of the process you will have audio recordings of the noise, signal strength measurements and other parameters from multiple locations. Armed with this information you should find tracking down the problem considerably less irksome!

THE PARTS LIST

Here is a list of the parts used:

- RFEngineers *Watch Dog* FM/AM/NOAA + RDS receiver, firmware v2.2.7,
- Raspberry Pi 3 Model B,
- Inexpensive USB sound card,
- Portable battery-based USB power supply,
- Ammo box (any similar container will do),
- Miscellaneous cables and an antenna.



WHY WE USED THE *WATCH DOG*

In addition to the Raspberry Pi, a key component of the system is a computer-controllable receiver. We use the *Watch Dog* receiver since we have extensive experience interfacing it with a Raspberry Pi.

The *Watch Dog* receiver is controlled through a USB (serial) port. As a serial device, it is very easy to connect and control in a variety of ways. This extensive interface is referred to as the *Watch Dog*'s "Serial API." Documentation can be found at http://www.RFEngineers.com/WD1.

The *Watch Dog*'s serial interface makes it ideally suited for automating with Python. We were able to the write the Python program in about two hours. Having access to previously developed Python code made the task even easier.

Our Python program, I*nterferenceMonitor.py*, is available free. You may adapt it to fit your specific needs.

PUTTING IT TOGETHER

We used a Raspberry Pi 3 Model B with a fresh install of Raspbian OS, but just about any Raspberry Pi should do. The Raspberry Pi was powered using a portable battery-based USB power supply.

The *Watch Dog* and sound card were connected to the Pi via two of its four main USB 2.0 connectors. An 1/8" male to 1/8" male stereo audio cable connected the *Watch Dog*'s "Headphone" jack to the "Microphone" jack on the USB sound card.

STEP-BY-STEP SOFTWARE INSTALL

The first step in getting *InterferenceMonitor.py* running is to find the *Watch Dog*'s serial port in the Pi. Run the following Linux command on the Pi to determine this:

1. *dmesg | grep tty*

Look for a line containing a message like USB ACM device. Copy down the full tty value – For example, *ttyACM0* – for a later step.

Next, run the following two commands on the Pi to install some prerequisite software (more information is here). Note the spaces, but watch the wrap:

2. *sudo apt-get install python-dev libportaudio0 libportaudio2 libportaudiocpp0 portaudio19-dev*

3. *sudo pip install pyaudio*

4. Now, install the *InterferenceMonitor.py* program directly into the Pi from our GitHub repository:

5. Open up the *InterferenceMonitor.py* program in your favorite text editor and change the following parameters:

a. FREQ – This is the primary frequency to monitor, in MHz. For example, 89.1.

b. MIN_SNR – This is the minimum acceptable SNR, in dB. For example, 10. If the SNR drops below this value then the setup will begin recording interference measurements and audio samples and continue to do so until the SNR returns to this level or higher.

c. POLL_SECS – This is how often the program will query the *Watch Dog* via its serial interface to get the latest SNR reading. Leaving this at the default value of 2 seconds should be fine for most uses.

d. WATCH_DOG_PORT – Put in the value that you found above in Step 1. For example, if the value you found was "*/dev/ttyACM1*" then you would change the value of WATCH_DOG_PORT to: */dev/ttyACM1*.

e. USB_AUDIO_DEVICE_INDEX – Leave this parameter set to the default value of -1. You will need to run the *InterferenceMonitor.py* program to get the proper value for this setting. That will happen in Step 6.

f. AUDIO_SECONDS – This is how many seconds of audio will be recorded on each channel (co-channel and adjacent channels) while interference is detected. The Raspberry Pi will continue to record this many seconds of audio from each channel as long as the interference is present.

6. Run the *InterferenceMonitor.py* program. It will display all of the audio devices that it detects and then stop execution because the USB_AUDIO_DEVICE_INDEX setting has not been changed from its default value of -1 in Step 5e above.)

The last few lines of output should contain a list of available audio devices.

```
Available audio devices:

Index 0: bcm2835 ALSA: - (hw:0,0)
Index 1: bcm2835 ALSA: IEC958/HDMI (hw:0,1)
Index 2: USB Audio Device: - (hw:1,0)
Index 3: sysdefault
Index 4: default
Index 5: dmix

Please set USB_AUDIO_DEVICE_INDEX to an
appropriate index from the above list
and re-run this program
pi@raspberrypi:~/Watch-Dog-Python $
```

In our case the sound was available as "USB Audio Device" with an Index value of 2.

7. Reopen up the *InterferenceMonitor.py* program in your favorite text editor and now change the USB_AUDIO_DEVICE_INDEX setting to the Index value obtained from this step.

MAKING IT ALL HAPPEN

You should now be ready to run the program.

Simply execute the following command:

python InterferenceMonitor.py

Once the program was running, disconnecting the antenna from our *Watch Dog* receiver was enough for us to simulate interference.

Re-connecting the antenna to the receiver caused the *InterferenceMonitor.py* program to return to its default state.

You can see here how the simulated interference period left us with a handful of audio samples (WAV files) and a spreadsheet of instrument readings (CSV file) from the period.

```
Starting RFEngineers, Inc. Watch Dog
Interference Monitor for 89.1 MHz

SNR at or above threshold: 30/10
SNR at or above threshold: 28/10
SNR at or above threshold: 28/10
SNR at or above threshold: 28/10
SNR at or above threshold: 27/10
SNR at or above threshold: 31/10
SNR below threshold: 0/10
2021-03-15 22:37:04,89.1,0,0,88
2021-03-15 22:37:09,88.9,0,0,80
2021-03-15 22:37:14,89.3,0,0,80
2021-03-15 22:37:19,88.7,0,0,72
2021-03-15 22:37:24,89.5,0,0,76
2021-03-15 22:37:29,89.1,0,0,80
2021-03-15 22:37:34,88.9,0,0,80
2021-03-15 22:37:39,89.3,0,0,76
2021-03-15 22:37:44,88.7,0,0,88
2021-03-15 22:37:49,89.5,0,0,84
SNR is no longer below threshold
SNR at or above threshold: 30/10
SNR at or above threshold: 30/10
SNR at or above threshold: 30/10
```

OPTIONAL SYSTEM IMPROVMENTS

A Raspberry Pi UPS will prevent the file system from being corrupted if the battery runs down before you can halt the system.

Similarly, adding a USB thumb drive for sound and data file storage will reduce the chance of SD card corruption in the event of an unexpected shutdown.

And, setting up WiFi Access on the Pi will allow you to grab the data or make adjustments to the parameters without disturbing the system.

SUMMARY

A Raspberry Pi and a bit of Python code can be a great “Swiss Army knife” for broadcast engineers.

Our total time invested in building this tool was less than 4 hours.

This article will hopefully get you thinking about other opportunities and ways for uniquely solving issues that arise in your day to day work.

- - -

Dan D'Andrea is the “software guy” at RFEngineers.

He is an amateur radio operator, embedded systems enthusiast, Software-Defined Radio (SDR) hobbyist, and a professional software developer with over 20 years of industry experience.

You can contact Dan at: dan@rfengineers.com

- - -

Return to The BDR Menu