CHART Interferometry Write-up

Ethan Blake

August 13, 2026

Abstract

This is a record of getting two RTL-SDR V3 dongles working as a two-element interferometer at 1420 MHz, and of four days at Winona State where most of what I learned was what breaks. The short version: share one clock between the dongles, write 8-bit samples rather than 32-bit or the SD card drops whole buffers without telling you, add SoapyRTLSDR's 0.6 DC offset before normalizing, and read the center frequency back out of the metadata file rather than trusting the default in the source. The best capture of the trip detected in all 59 half-second chunks at lag zero with a median detection ratio of 19.6. That demonstrates two receivers locked together and correctly aligned; it does not yet demonstrate that what they share came from the sky.

1 Introduction

This "paper" relates to my visit to Winona State from August 4-7, 2026, working with CHART on connecting two radio telescopes together via interferometry. I will start with how to get everything to run and end with my process.

2 How to get interferometry running

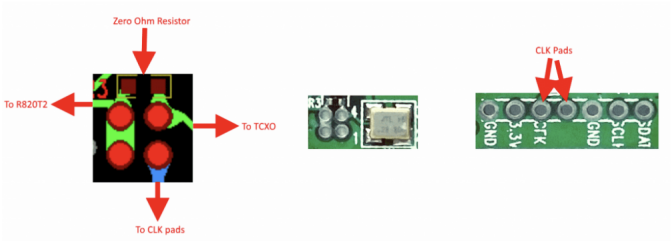
2.1 Supplies needed

- Powered USB hub with as many inputs as you are planning to have horns. I used the "Onn. USB 3.0 Hub with 4 USB 3.0 Ports" which worked well enough, if not a bit spotty at times.
- 2 RTL-SDR V3s
- 2x regular CHART telescope equipment (Horns, amplifiers, SMA cables)

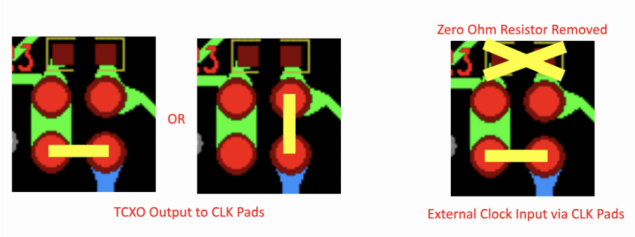
2.2 Soldering

First things first, you have to solder your two RTL-SDRs together. We used the V3 version. If you go here: https://www.rtl-sdr.com/rtl-sdr-blog-v-3-dongles-user-guide/ or look up RTLSDR V3, you should find instructions on how to solder there in feature 3.

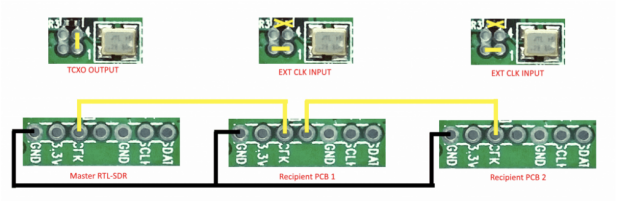
The gist of it is there is a little clock pad on each RTL. On the "receiver" RTL, you need to remove the 0 Ohm resistor and connect, via a jumper wire or solder bridge, the "to clock pad" and the "to R820T2". On the controller, do the same solder bridge/ jumper wire but DO NOT REMOVE THE 0 OHM RESISTOR. If you wish to have more than two radio telescopes, you should have only one controller, and the rest are receivers. Then connect the GNDs together (both grounds are labeled and are connected to each other; if you mess it up, just use the other one). Then connect the clocks. I have added the instructions directly from RTL below. (The captions come directly from the RTL-SDR website)



(a) How to connect the CLK jumpers.



(b) The first position outputs the dongle's clock to the CLK pads; the second accepts an external clock.



(c) CLK daisy chaining: one dongle's TCXO feeds two dongles with disconnected clocks.

Figure 1: RTL-SDR v3 clock modification steps.

2.3 Running Interferometry

In the Pi run:

```
sudo apt update && sudo apt full-upgrade -y && sudo apt install -y git \
    gnuradio soapysdr-module-rtlsdr soapysdr-tools rtl-sdr \
    python3-matplotlib python3-pytest

echo 'SUBSYSTEMS=="usb", ATTRS{idVendor}=="0bda", ATTRS{idProduct}=="2838", MODE:="0666"' | sudo tee
    ↪ /etc/udev/rules.d/rtl-sdr.rules
echo 'blacklist dvb_usb_rtl28xxu' | sudo tee /etc/modprobe.d/rtlsdr.conf
(sudo modprobe -r dvb_usb_rtl28xxu || true)
```

https://github.com/EthanpBlake-7/CHART-Interferometry

The next step is to download the repository located at the link above. Git clone it onto your Pi. In the repo, there should be capture.py and correlate.py. Run ls -l to make sure these files are not blank.

This next step can be annoyingly difficult. Plug in your hub's power source, then plug the power source into your hub if they are not directly connected. Next, plug in your controller to the hub. Run:

```
rtl_test
rtl_eeprom -d 0 -s 00000101
rtl_test
```

On the second `rtl_test` you should see something like (the first run will look similar with a different SN code:

```
kenyonphysics@Kenyon-Chart-Pi:~/src/CHART $ rtl_test
Found 1 device(s):
  0:  Realtek, RTL2838UHIDIR, SN: 00000102

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
[R82XX] PLL not locked!
Sampling at 2048000 S/s.

Info: This tool will continuously read from the device, and report if
samples get lost. If you observe no further output, everything is fine.

Reading samples in async mode...
Allocating 15 zero-copy buffers
lost at least 104 bytes
```

Once it gets here, you can control-C to stop the test. What you're looking for is the SN: 00000101. I have my controller set as 00000102 instead of 00000101. That's because I ran the `rtl_eeprom` before I soldered.

Now there is a very good chance that when you run `rtl_test` the first time, you get "No supported devices found"; this is where the hub gets annoying. The solution I found was to start unplugging things related to the hub and re-plugging things in (i.e., the power cable from the outlet or the power cable to the hub, or the RTL to the hub). After every re-plug, try `rtl_test`. If this still doesn't work, try changing outlets the hub power is connected to or switching directions it's facing. Eventually it'll work. Probably.

With the controller still plugged in, plug in the receiver RTL and run `rtl_test` again. Leave the controller connected: after the clock mod, the receiver has no oscillator of its own, so on its own it will not even enumerate. The receiver will come up with its factory serial, 00000001, which is how you know which device number to hand `rtl_eeprom`. Set it with `rtl_eeprom -d <n> -s 00000102`. Run `rtl_test` once more, and you should get:

```
kenyonphysics@Kenyon-Chart-Pi:~/src/CHART $ rtl_test
Found 2 device(s):
  0:  Realtek, RTL2838UHIDIR, SN: 00000101
  1:  Realtek, RTL2838UHIDIR, SN: 00000102

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
[R82XX] PLL not locked!
Sampling at 2048000 S/s.

Info: This tool will continuously read from the device, and report if
samples get lost. If you observe no further output, everything is fine.

Reading samples in async mode...
Allocating 15 zero-copy buffers
lost at least 36 bytes
^CSignal caught, exiting!

User cancel, exiting...
Samples per million lost (minimum): 5
```

This shows both RTLs are connected!! You are now good to set up each telescope as you would any CHART telescope.

2.4 Running the Code

There are a few commands that you can call when running on the RTLs. They are listed below with their defaults.

```
sample_rate = 2.048e6, seconds = 1, source = 'sim', center_freq = 1419.99e6, gain = 40.0, serials =
    ↪ None, biastee = True
```

Serials is set to None, but if your `rtl_test` ran properly, capture will find them automatically. The important commands are source, seconds, and sometimes center frequency. It's important to note that because of mean subtraction, there is a dip at the center frequency. This is an artifact of the setup and not a detection.

The command I recommend running is as follows:

```
python capture.py --source rtl --seconds 30
```

After a dozen or so RTL outputs, the capture will start. After it's done taking the data, you will get two outputs, one from each RTL.

```
61440000 samples, var 0.0074235
61440000 samples, var 0.0068579
```

Above is an example output from a capture. First listed is the size of the file. It should be 2.048e6 times the number of seconds. The next is the var. Var is the variance of the complex samples. This tells us if the RTLs are actually reading anything. A value of 0 means they're reading nothing (in reality, a value of ∼0.000065 means the RTLs are connected but aren't reading anything). A value above 0.1 means the RTLs start clipping, which indicates that the signals are too large to be represented properly. So our goal is to get var around 0.005. My best was 0.0074. This is a clearly detectable signal that isn't too large to lose important information.

Another thing to look out for is the ratio between vars. If you are taking a clean capture and expect no nearby RF signals, the ratio of vars should be close to 1. I will discuss later what caused the ratio between the vars to be much greater than 1.

2.5 Pi Connect

A very cool website is Pi Connect (found by looking up Pi Connect or going to https://www.raspberrypi.com/software/connect/). If you are near WIFI that your Pi can connect to, you can screen share the Pi to a laptop or desktop so you don't need a separate monitor. Using this, I was able to sit in the shade instead of the sun while running all my commands. It also helps with mobile data-taking.

3 What we did August 4 - 7, 2026

3.1 Before Winona

I came to Winona State with a mostly complete Capture and Correlate but had only tested it on simulated data and GPS Antennas at 1575 MHz. I had used an RF splitter to get both RTLs to read the same signal to simulate them reading the same sky source. I also had the RTL-SDRs soldered (with the help of Dr. Paula Turner). Capture wrote 32-bit decimal and was writing at ∼32 MB/s to the Pi's SD card.

3.2 Day One

We set up two horns about 4 meters apart. We ran about 10 captures, each lasting between 4 and 60 seconds. At this point, the default center frequency was 1575.42 MHz. When I was writing the capture, I started by running all the code with the two RTLs connected to a single GPS antenna via an RF divider. This allowed me to use a common signal, thus mimicking the sky, without having to go out with two horns and set them up every time I wanted to test a little bit of data. The antennas measure 1575.42 MHz, so that was my default for the initial coding phase. When transitioning into the field, I forgot to change this default, which came back to bite me twice this week.

Our intentions were to start with a few basic runs over short times with both horns on and pointed at the same source; however, the receiver got disconnected. My receiver is labeled as 00000101 and my controller as 00000102. When the receiver got disconnected, the loop that picks out the serial numbers

looked for 00000101 and saw none and loudly failed. Our first run was a single detector run. At this point, I was planning on having some single detector runs. I later realized that I could just take each data stream individually as a single detector run, and then I'd have coincident single detector runs from each horn. We took a few more sets of data with both horns, with the controller connected but still at 1575.42 MHz. We got var ratios of 1 and 1.077. Which is exactly what we expect from this kind of run, as the horns don't actually pick up 1575.42 MHz. This isn't the full story though. By cross-correlating the signals, we get a ratio (the cross-correlation peak over its noise floor) between 160 and 240. This indicated that our receivers were receiving a similar signal, just not what we were expecting to see. This result was not seen in day two, so we assume it to be a source of RFI at the time of data taking.

After looking at the data and realizing my center frequency mistake, I went back out and took more data. I set up longer captures of 60 seconds so that we could take slices out of the data to analyze. We also implemented an RF generator. We set this to 1420.2 MHz and set it at specific locations. The goal of the RF generator was to have something we could use to correlate each signal.

The big issue we found from Day One's results was that the RF generator was too strong. The horns were completely drowned in the signals, and we couldn't see anything. On one capture, we had the RF generator in between the horns and got vars of 1.372 and 1.142. This is way higher than we can reasonably detect.

3.3 Day Two

For Day Two, we added a var check to make sure that the ratio of vars isn't too great. A var ratio that's too large could indicate that one horn is receiving too much information and thus will hit the ceiling of what we can detect. We lowered the gain of the RF generator. We set the horns up about 3-4 meters apart. I ran two runs at 1575 MHz to check the RFI signal we had found the day before and got no results. I then added `--center_freq` 1419.99 MHz to the command line. I ran a few runs with each RTL connected into the hub but not to their respective horn. This allows us to test the cross-spectrum function. We expect no correlation in the signals if only one horn is connected.

We split our capture into half-second chunks and found that each chunk had no discernible peak.

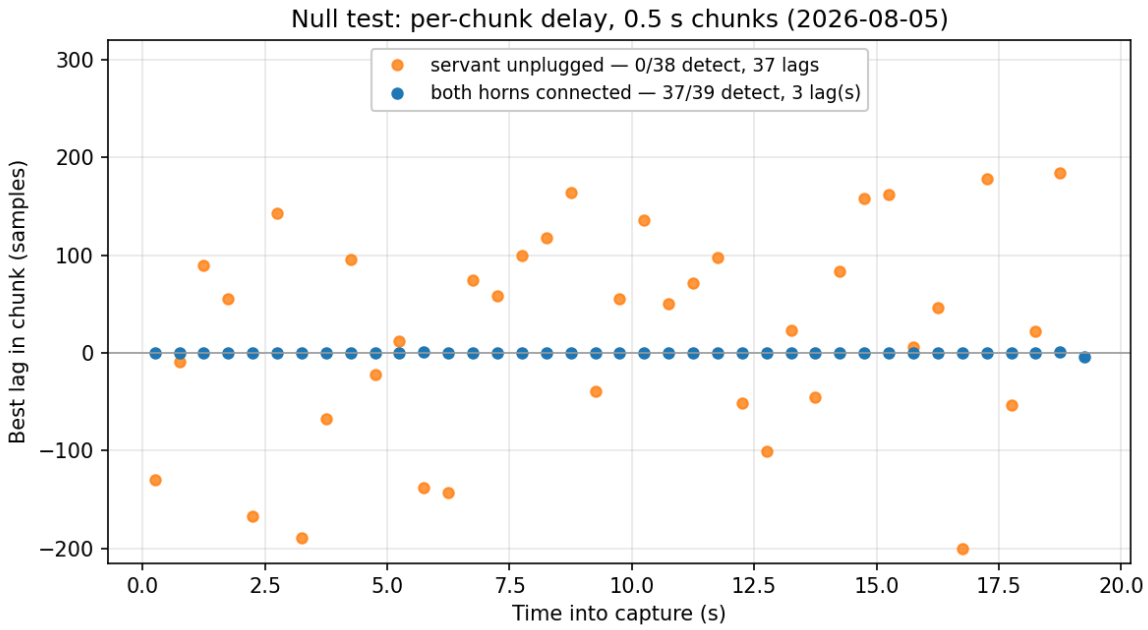


Figure 2: The blue dots are when both horns are connected. The lag is how far we have to shift one horn's data to line it up with the other's. Acquire finds the big offset between the two receivers first, and align takes it out, so whatever is left should sit at zero. It does, for 37 of the 39 chunks. When we disconnect a horn, that receiver only sees its own noise, so there is nothing in common to line up and the correlation has no peak. Correlate still reports a best lag for every chunk; it is just a different random one each time. None of those 38 chunks pass our detection threshold.

Next, we connected both horns and ran the RF generator on a much lower setting. We tested the power at different settings across three runs: -32.72 dBm, -3.2 dBm, +2.8 dBm. The generator was only on for part of the capture, so every one of the runs had its own generator-off control for free. We saw the RF generator in Fig. 3 right at the frequency we set it to: 1420.2 MHz.

The two louder settings did the same thing day one did. At -3.2 dBm and +2.8 dBm, the receiver clipped, with vars of 0.36 and 0.53 against the 0.007 we get on a clean capture. While the tone was on, the ratio came out at almost exactly 1. A single tone correlates just as well at every lag, so it raises the peak and the noise floor by the same amount, and the ratio has nowhere to go.

Only the weakest setting worked. At -32.72 dBm, the variance did not move at all, and we still detected in 24 of 27 chunks with the tone on.

Our data captured at 1419.99 MHz detects at lag zero, with vars of 0.008874 and 0.007448.

This was a successful capture! We detected two signals independently of each other, and their vars agreed with what we were looking for with a low-power RF transmitter.

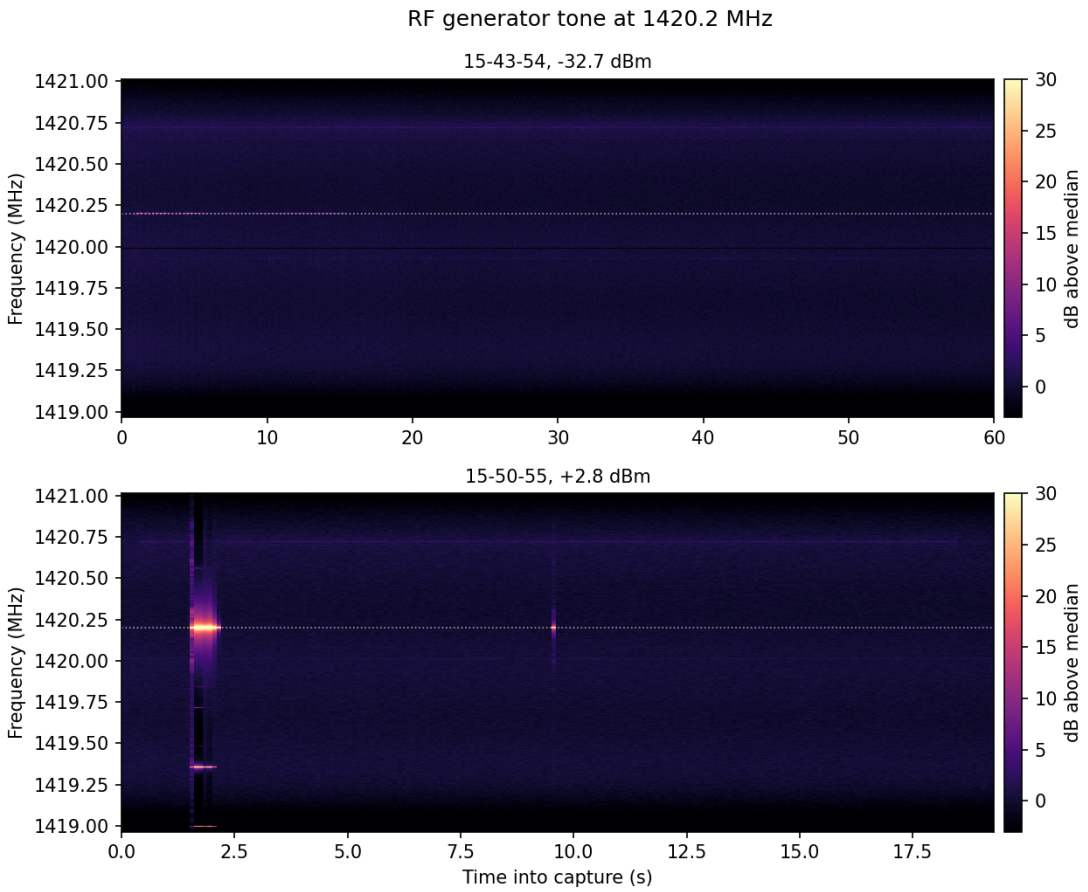


Figure 3: Both panels show the receiver's data as frequency against time, in dB above the median, so the shape of the bandpass drops out, and only what changes is left. The dotted line marks 1420.2 MHz, where we set our generator. On top is our weakest setting at -32.7 dBm. The tone is a thin line about 14 dB above the background, and nothing else in the band moves, which is why the sky detection survived in this run. On the bottom is our loudest setting at +2.8 dBm. The tone reaches 48 dB, but the important part is that it is not a line anymore. For about two seconds, the whole band lights up, with extra spikes near 1419.35 and 1419.0 MHz where nothing was transmitting. That is what clipping looks like. Since both RTLs do this at the same time, it correlates at every lag and drags our noise floor up to meet the peak. The faint line near 1420.72 MHz is in every capture we took, including the quiet ones, so it is a spur in the receiver or local RFI and not something we put there.

3.4 Day Three

On day three, I decided to get a few more captures before going in depth into analysis. This is where the 1575 MHz came to bite me. I changed the value in the init but not in the arg parse to 1419.99 MHz. This meant that while I thought all my data was being taken in 1419.99 MHz, it was actually 1575 MHz.

When analyzing the data, I realized my mistake, but I also found another issue. Capture was writing 32-bit decimal, which was sometimes too much for the SD card to keep up with. When it fell behind, the radio threw away large swaths of data, about 64 ms of data at a time, which corresponded to 131,072 samples or one buffer. We call this slipping. The tricky part is that the file still comes out the same size as one that doesn't slip because the code counts samples it was handed, not the samples the radio actually took.

I found this issue going back through day two's data. The slip had looked like a source turning on and off. I had read one of the captures as an intermittent source, 38 out of 119 half-second chunks detecting with a median ratio of 2.8. Once I lined the two segments up separately, it was 38 out of 40 chunks with a median of 16.1. The same 38 chunks the whole time. They were the first 20 seconds,

and the other 81 weren't empty; they were just misaligned. The stream slipped about 20 seconds in.

What convinced me it was buffers and not a clock problem was that the shifts were always whole numbers of buffers. They were exactly 35, exactly 14, exactly 100 in different captures. A drifting clock would give you any number. Both dongles do it, since the shift goes in both directions depending on which one fell behind.

I had arbitrarily chosen 32-bit decimal (or fc32) as the write size. I first investigated whether this was the correct size. I found that the 32-bit samples were only finding 83 different values out of the millions expected from 32-bit. This told us that the data from the RTLs was 8-bit, or sc8. The unique sample ceiling for 8-bit is 256. 83 being much less led us to this discovery. Lowering to 8-bit means our slipping was nearly erased. The 32-bit captures lost 38 and 57 buffers while the 8-bit lost 0 and 2 buffers.

SoapyRTLSDR has a built-in 8-bit DC offset of 0.6, so we had to add 0.6 when normalizing. Without it, one of my strongest captures had a correlation ratio of 3.8 instead of 52.7. Since both streams carry the same DC offset, it correlates at every lag and raises the noise floor.

The plot in Fig. 4 compares four captures. Two were when we were writing fc32, on days one and two, and the others were from after the change to sc8 from day four. We show day four results because our day three captures were set to 1575 MHz and thus were deaf.

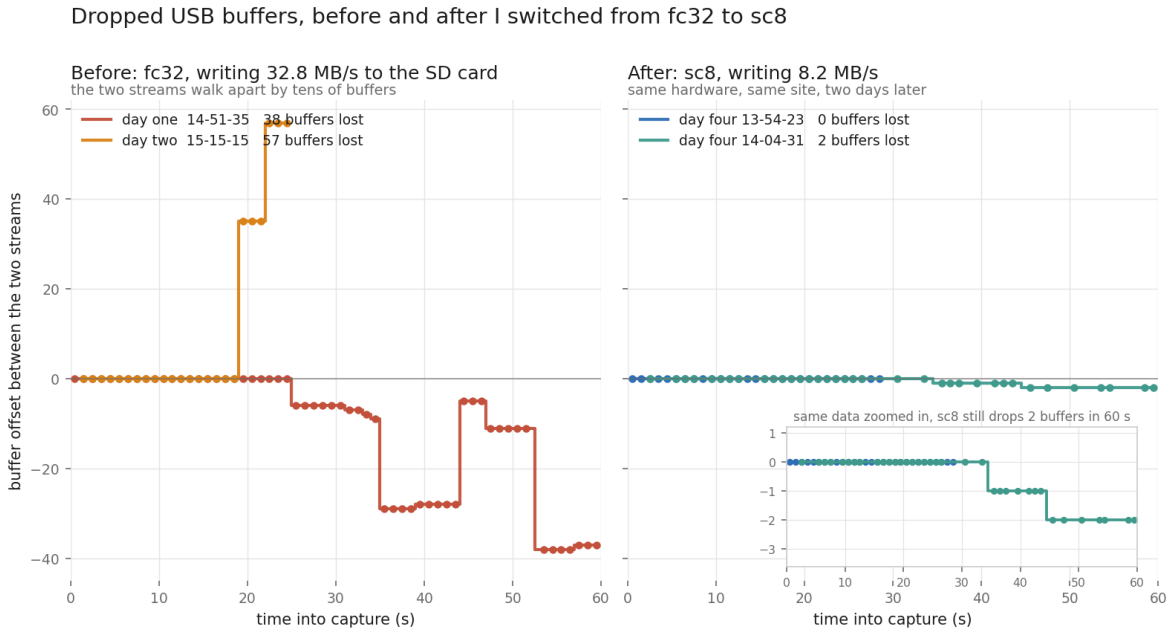


Figure 4: One buffer is 131,072 samples, which is 64 ms of data. Each point is the buffer offset where that one second of data lines up. I only plot a second if it detects above a detection ratio of 6, because if a second does not detect, I cannot tell where it lines up. The sign just says which dongle fell behind. Two other sc8 captures are not shown: 13-46-08 loses its signal after 11 s, and 14-10-12 was railed and never detects, so neither one can tell me an alignment.

3.5 Day Four and Conclusion

On day four, I fixed the center frequency defaults. This changed our vars from the receiver noise values to actual signal values, which is what is expected when we are actually measuring a signal. I added a checker that reads the frequency of a capture from the metadata files that are produced after a capture. This check will not fix all captures but can lead a user in the right direction if the center frequency is set wrong.

We rigged up a new RF transmitter. We had a Raspberry Pi and ADALM-Pluto SDR on a broom emitting a broadband spectrum across 1420 ± 1 MHz. Our broadband signal pulsed at random intervals for 10 seconds. Our goal with this was to get signals we could use to calibrate our two antennas and make sure they were aligned. We only ran for the first 10 seconds so we could check alignment and then use the rest of the data to analyze our correlation. We ran this test on either side of the horn set

up and once in the middle. One issue we ran into with this new setup was that our transmitted signal was too hot and caused our received signal to clip. This made our regular signal significantly smaller than the transmitted signal, and our signal ratios dropped from the 100s at the transmitted pulses to less than 3 everywhere else.

The best capture of the trip happened on this day! All 59 half-second chunks had a lag of exactly zero, with a median ratio of 19.6 and a max ratio of 28.98. The vars for this capture were 0.0074 and 0.0069. Both were within an acceptable range and reasonable relative to each other.

We also pointed our telescopes at Betelgeuse, a star we believed to be near enough to the galactic plane that we might get hydrogen 21 cm data from. Where we were was not a conducive environment (buildings and trees were in the way), and we did not get any usable data from this attempt.

We went back to the lab to analyze our data. We found that our capture with no transmitter was our best capture, and we were able to plot both antennas and the cross spectrum. As seen in Fig. 5, our cross spectrum is louder than the noise floor. This is indicative of correlation in the two antennas.

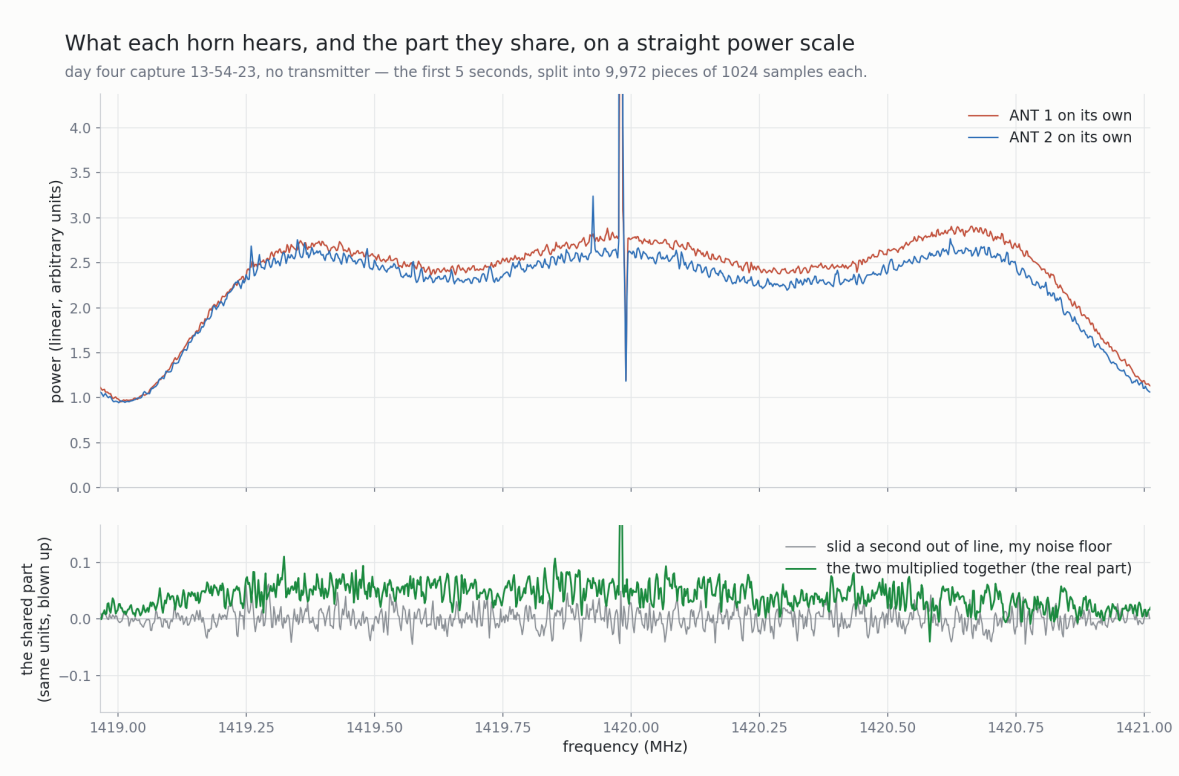


Figure 5: TOP: Plot of each horn's individual signals. Antenna 1 in red is slightly above Antenna 2 in blue. The narrow feature at 1419.99 MHz is the radio's own DC spike, which appears as a downward notch because each chunk's mean is removed before the FFT. The secondary peak at 1419.98 MHz, more prominent in antenna 2 but shows up in antenna 1 as well, is ambient RFI and runs off the top of the panel. BOTTOM: Plot of the real part of the cross spectrum between the two signals in green. In gray, I slid one of the antennas' signals by one second. This uncorrelates the signals and shows us what random noise would look like. The correlated signal is about 3.3 times the noise floor. This indicates that our signals are in fact correlated.

Acknowledgements

Thanks to Dr. Maddie Wade for advising me on this project and many other projects while at Kenyon this summer.

Thanks to Dr. Paula Turner for help soldering the clock modification onto the two dongles, and for the advice that kept this project pointed in a useful direction.

Thanks to Dr. Adam Beardsley, at Winona State who hosted my August 4–7 visit, helped set up

and move the horns, ran the transmitter back and forth across the courtyard, and lent the RF signal generator. Most of the results here came out of runs that took two people.

This work builds on the CHART project, whose capture and analysis code I adapted for the two-receiver case, and on the RTL-SDR Blog documentation for the V3 clock modification, whose figures are reproduced in Section 2.2.