How to process dual-site SETI observations with I-LOFAR and LOFAR-SE

Charlie Giese

July 9, 2021

Contents

1 Introduction

This document is intended to instruct students in processing the data resulting from dual-site SETI observations with LOFAR-SE and I-LOFAR, as well as how the resulting data products can be compared for anti-coincidence testing. For each step of the process I will try and provide both a general explanation as well as the specific commands used.

In order to use the data processing pipeline, you will need to modify your personal `.bashrc` file. This file is sourced by the shell upon login and essentially sets up the shell for your use. I recommend copying and pasting the last 8 lines from `/home/charlesg/.bashrc` to your files. This will add the requisite directories to your `PATH` and other environment variables. Note, you will need to replace every instance of `$HOME` with `/home/charlesg` in those lines. There is another useful bash script, `/usr/local/pulsar64/pulsar.bash`, which activates some of the general purpose tools installed by default on the BL machines. Note that you will need to enter `source  /.bashrc` after doing `source /usr/local/pulsar64/pulsar.bash`. Also, activating the pulsar environment breaks something in Python, and you will need to enter `unset PYTHONPATH` to make Python work again.

In order to transfer files between the two compute nodes, I recommend using `rsync.` It has a simple syntax;
`rsync [OPTIONS] user@remote1:/path/to/source`
`user@remote2:/path/to/destination`. This is simplest when the files you need to transfer are in the directory `/home/user` as these files can be accessed through the headnodes, `blh0`. For Onsala, you can connect to the headnode directly, but for Birr you will need to set up a multi-hop SSH connection. For details on how to do this, please see Dominic Le Duc's guide which is pinned in the interns channel on Slack.

2 Data Processing

2.1 Identifying the Data

The first processing step is to find the location of the data obtained from any one observation. Unfortunately the locations differ depending on which station you are currently using. The files we are looking for will be in Zstandard format, with a .zst file extension. These files will usually have file names like

```
udp_SE607_16070.blc00.bl.pvt.2021-04-21T13:45:00.000.zst
```

The filename signifies the station at which it was recorded (`SE607` corresponds to Onsala), the date and time at which the observation began, as well as which port this file corresponds to. Our data will generally always cover 4 ports, with identifiers 0-3. As such each observation will produce 4 files named something like

```
udp_SE607_16070.blc00.bl.pvt.2021-04-21T13:45:00.000.zst
udp_SE607_16071.blc00.bl.pvt.2021-04-21T13:45:00.000.zst
udp_SE607_16072.blc00.bl.pvt.2021-04-21T13:45:00.000.zst
udp_SE607_16073.blc00.bl.pvt.2021-04-21T13:45:00.000.zst
```

All 4 files are required for processing to begin. At I-LOFAR, data will be found in `/datax2/obs/<observation_name>/udp...zst` (Note, if you're using ILISA to perform the observation the data will be located in the same directory as in the Swedish case). This data can then be accessed by logging onto blc00 and navigating to the directory in question. At the Swedish station things are a little more tricky. The file corresponding to each port is in a different directory; `/datax/laneX/....` where X corresponds to the port number. On the swedish node, you can also find metadata pertaining to the observation. Metadata is found in `/datax/Projects/proj21/....` where proj21 is the project number for all SETI observations.

These .zst files are simply compressed streams of UDP packets which are produced by the station.

2.2 GUPPI RAW

The ultimate goal of the processing the data is to produce two data products for every observation in the form of filterbanks. A filterbank, in this instance, a 2D stream of radio frequency channels versus time. We want to produce 2 filterbanks with varying time and frequency resolutions, one for Pulsar and FRB research, and one for SETI. A general outline of the this process can be outlined as:

- Locate compressed UDP data streams
- Convert data in GUPPI Raw format with `lofar_udp_guppi_raw`
- Convert .raw files produced above into filterbanks using `rawspec`

The suite of software used in processing is all contained in a singularity image on blc00 at each station. These images are located in `/datax2/obs/singularity/` at both stations.

We will first use a command-line interface (CLI) created by David McKenna called `lofar_udp_guppi_raw` to convert our data to GUPPI raw format. Generally this CLI can be called by simply using `lofar_udp_guppi_raw -h`, to print the help options to the console for example. When using a singularity image this command becomes;
`singularity exec <path/to/image> lofar_udp_guppi_raw -h`
The above command will print a help message to the console which describes the various parameters that can be passed. The images are found in `/datax2/obs/singularity` on both machines, and it's generally wise to use the most recently updated image, which at the time of writing is `lofar-upm_0.7.0-2021-03-25-9c91c04c3cc1.simg`. The command line arguments needed are as follows:

- -i : The name of our input .zst files, with the port number replaced by %d.
- -o : The output directory and name of output .raw files. General format: `<source_name>.%04d.raw`
- -m : Number of packets to process in each read request, use 4096.

- -e : Split the file every N iterations, use 1.
- -a : fakeHeader text file for observation. Based on a template you will need to provide a fakeHeader text file containing metadata to the CLI. In general, as the same frequency window will be used for each observation only the source name paramater will need to be changed. You can find a template file in /home/charlesg/B2217+47/data/ on the Swedish node.
- -b : Range of subbands used. 0,411 for 110-190 MHz. The exact values can be calculated based on the frequency range.

Finally, we then arrive at our completed command, except for one more modification. Currently, we need to source a bash_profile before running the extractor, and thus our final command looks as follows.

```
singularity exec <path/to/image> bash -l -c
"source /root/.bash_profile; lofar_udp_guppi_raw
-i udp_SE607_1607%d.blc00.bl.pvt.2021-04-21T13:45:00.000.zst
-o /output/path/B2217+47.%04d.raw -m 4096 -e 1
-a ./fakeHeader_B2217 -b 0,411"
```

The extractor can take some time to process our data and so it is recommend it to run it from inside a tmux virtual terminal. The process for using these is very simple. You simply enter `tmux new -s <terminal name>`, which will create a new tmux instance. You then simply execute the command in this virtual terminal. You can detach from the terminal by pressing `ctrl+B` and `D` in succesion. Re-attach to the window by entering
`tmux attach -t <terminal name>`. You will know that the process has exited succesfully as it will print the following message to the terminal:

```
Data processing finished.  Cleaning up file and memory objects...
Reader cleanup performed successfully.
CLI memory cleaned up successfully.  Exiting.
```

2.3 Channelisation

When the extractor has finished you will find many (1-2000) .raw files in your chosen output directory. In order to form our filterbank files we use a software called *rawspec*, which is a CUDA based spectroscopy package for GUPPI RAW. This process is far simpler. Navigate to your chosen output directory and enter `rawspec -f 65536,8 -t 2,16 <source_name>` where `<source_name>` is the same as that used above. The first argument determines the length of the FFT's (Fast Fourier Transforms). In our case, 411 subbands become 411×65536 frequency channels in the case of the high frequency resolution product and 411×8 channels for the high time resolution production. This will produce two filterbanks (.fil extension):

- `<source_name>.rawspec.0000.fil` for the high spectral resolution product.
- `<source_name>.rawspec.0001.fil` for the high time resolution product.

You can check for yourself the time and frequency resolutions of these filterbanks using Sigproc. You should be able to use Sigproc executables following the additions to your `bashrc` file earlier.

You can use tools from Sigproc, a suite of software package used in pulsar research. It's most useful feature is the `header` command which prints the header of a specified filterbank file. It has some other tools which I recommend you check out. There also exists a Python3 wrapper for Sigproc which can be very handy, called Sigpyproc.

Once you have produced both data products at both the Irish and Swedish stations it is useful to take a brief moment to organise your directories. This is optional but I found myself often facing confusion because I hadn't named files/directories in an informative manner. Having clear and repeatable structures in your directory hierarchy will save you time in the long run.

3 SETI: Searching for aliens using turboSETI

We now move onto searching our data for narrow-band signals which could be of interest to SETI researchers. To do so, we use an analysis tool called turboSETI which searches filterbank files for narrow-band signals which are drifting in frequency. turboSETI, and it's dependencies, are installed in a Python3 virtual environment, found in `/home/charlesg/seti_py` at both stations. You can activate this environment by entering `source /home/charlesg/seti_py/bin/activate`. Once you have activated the environment you can use turboSETI. This is quite straight-forward, simply navigate to the directory containing your high spectral resolution filterbank, and turboSETI can be called from the command line. It takes a number of arguments;

- -M : Maximum drift rate for turboSETI to search to. Based on estimates by Sheikh et al. (2019) a maximum drift rate of $\pm$ 4 Hz/s is appropriate.
- -n : Number of coarse channels. This argument is required for turboSETI to work with our data. The number of coarse channels is simply the number of subbands before channelisation, 411 in the above example.
- -s : Minimum SNR threshold.

The final argument is then the name of the filterbank. turboSETI can also take some time to run, so using tmux is recommended. As turboSETI runs it prints the SNR and drift rate of each fit it finds to the terminal, and will print the total search time when it has finished. When turboSETI has concluded, there will be three new files in your directory. turboSETI first creates a copy of the filterbank in H5 format, with a .h5 extension, and then searches this file. It tabulates the results in a .dat file and produces a .log file which is useful for diagnostics. The .dat file is most important, it details the frequency, drift rate and SNR of each potential "hit" (narrow-band signal) turboSETI has found. It should look something like Figure 2. For each hit, you can produce a plot by calculating the central frequency of the hit and plotting a frequency window corresponding to how far the signal drifts over the length of the observation. This can be done using some of the code in my GitHub repository. Such a plot should look something like Figure 1.

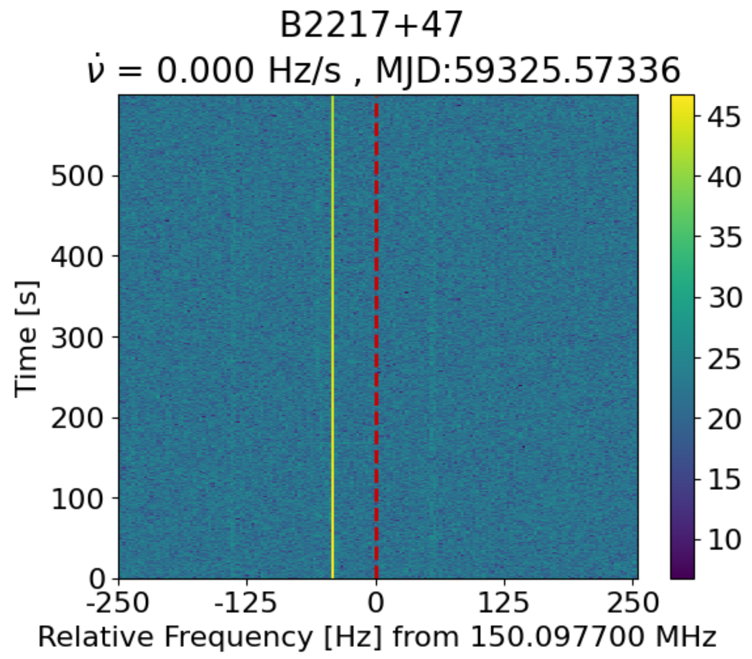
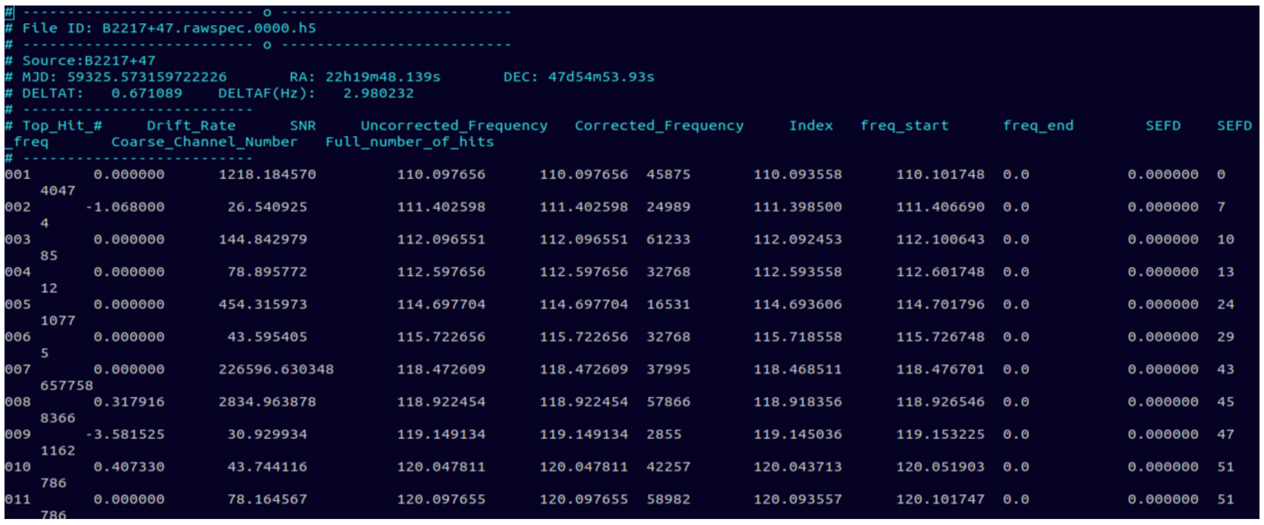


Figure 1: Plot of a narrow-band hit detected by the turboSETI algorithm. For a hit which isn't drifting, a 500 Hz frequency window is standard. The dotted red line is overplotted for signals where the drift rate is difficult to see.



```
# -------------------------- o --------------------------
# File ID: B2217+47.rawspec.0000.h5
# -------------------------- o --------------------------
# Source:B2217+47
# MJD: 59325.573159722226       RA: 22h19m48.139s       DEC: 47d54m53.93s
# DELTAT:   0.671089    DELTAF(Hz):   2.980232
# --------------------------
# Top_Hit_#     Drift_Rate      SNR     Uncorrected_Frequency   Corrected_Frequency     Index   freq_start      freq_end        SEFD    SEFD
_freq       Coarse_Channel_Number   Full_number_of_hits
# --------------------------
001       0.000000      1218.184570         110.097656      110.097656  45875       110.093558      110.101748  0.0            0.000000  0
    4047
002      -1.068000       26.540925          111.402598      111.402598  24989       111.398500      111.406690  0.0            0.000000  7
    4
003       0.000000      144.842979          112.096551      112.096551  61233       112.092453      112.100643  0.0            0.000000  10
    85
004       0.000000       78.895772          112.597656      112.597656  32768       112.593558      112.601748  0.0            0.000000  13
    12
005       0.000000      454.315973          114.697704      114.697704  16531       114.693606      114.701796  0.0            0.000000  24
    1077
006       0.000000       43.595405          115.722656      115.722656  32768       115.718558      115.726748  0.0            0.000000  29
    5
007       0.000000      226596.630348       118.472609      118.472609  37995       118.468511      118.476701  0.0            0.000000  43
    657758
008       0.317916      2834.963878         118.922454      118.922454  57866       118.918356      118.926546  0.0            0.000000  45
    8366
009      -3.581525       30.929934          119.149134      119.149134  2855        119.145036      119.153225  0.0            0.000000  47
    1162
010       0.407330       43.744116          120.047811      120.047811  42257       120.043713      120.051903  0.0            0.000000  51
    786
011       0.000000       78.164567          120.097655      120.097655  58982       120.093557      120.101747  0.0            0.000000  51
    786
```

Figure 2: Example of a .dat file output from a narrow-band search using turboSETI.

Once you have done this for each station, we can begin to reject hits by comparing the results from Ireland and Sweden. For this, you will need to transfer one .dat file to another station, I recommend the Swedish station. You can do this using `rsync`. Once you have both .dat files in one directory you use a simple script to compare them. In essence, we are looking for hits which appear at the same frequency and drift rate at both stations. You can use my script `dat_comp.py` as a basis.

4 SPANDAK

For each observation a high-time resolution filterbank is also produced. These filterbanks will be searched using Heimdall, a GPU accelerated transient detection pipeline. We will also use a wrapper for Heimdall called SPANDAK which was written by Vishal Gajjar [1]. Using this wrapper is very straightforward, but requires some setup. Both Heimdall and SPANDAK, and their dependencies, have been installed on the compute nodes at both stations. Every time you wish to use SPANDAK you will need to enter the following commands;

- `source /usr/local/pulsar64/pulsar.bash`
- `source  /.bashrc`
- `SPANDAK -fil file.fil -lodm 0 -hidm 10000`

(Of course you only need to enter the first two lines once, you can then run SPANDAK as many times as needed). The first command gives you access to some of the dependencies SPANDAK requires, however it overwrites some changes your `bashrc` makes and so you will need to source this again. The SPANDAK command given above will look for transients at dispersion measures from 0 to 10,000. SPANDAK will produce a .png file with some plots for each potential FRB candidate. They should look something like Figure 3 and are useful for further analysis.

[1]`https://github.com/gajjarv/PulsarSearch`

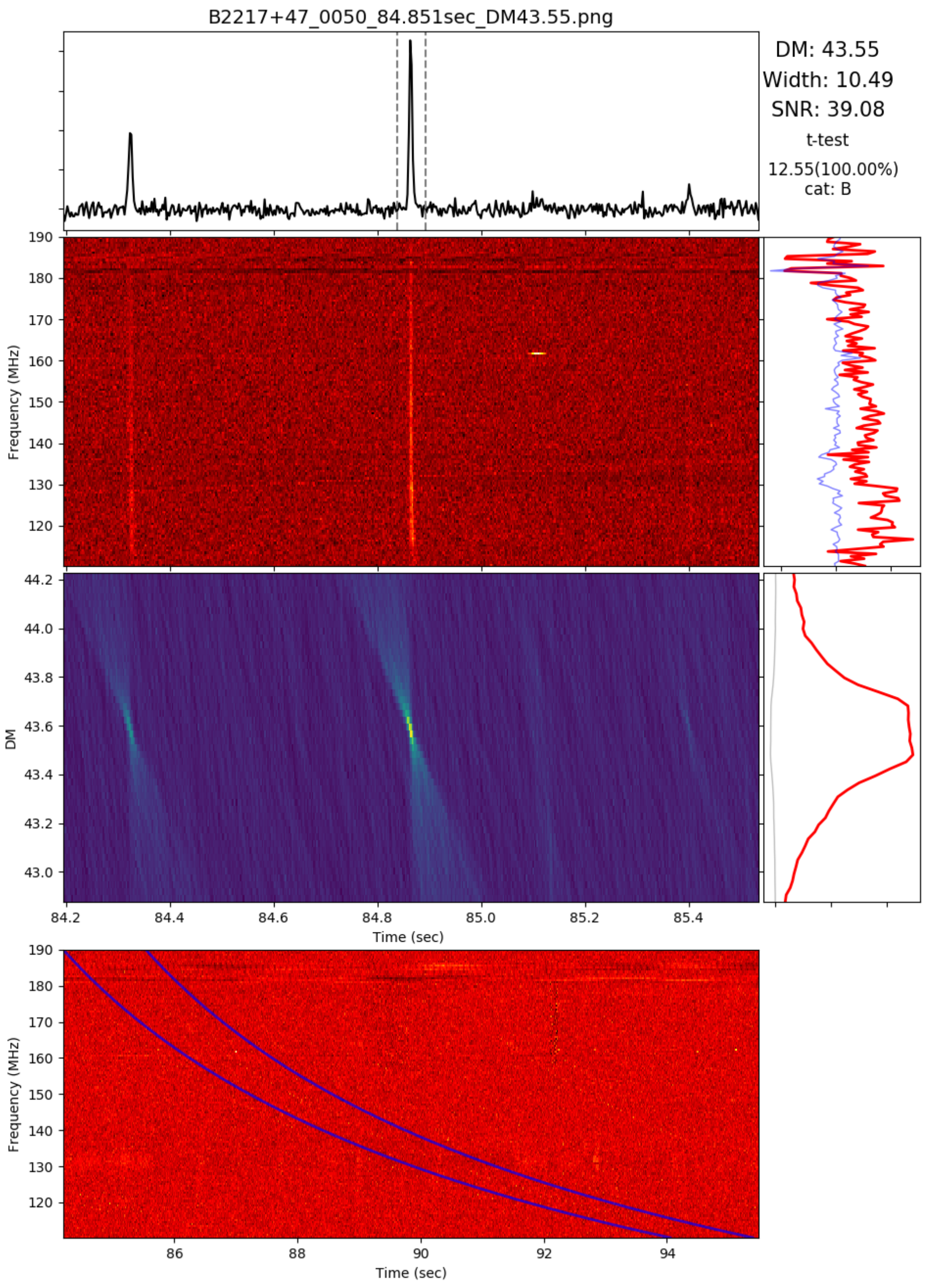


Figure 3: Output image of a pulse detected by the SPANDAK pipeline. Note, this image was produced using data from the pulsar B2217+47. The top panel shows a timeseries and gives some basic info about the nature of the pulse. The second panel shows a dynamic spectrum which has been dedispersed, and the panel below this shows a dispersion measure vs time spectrum. This plot shows the SNR of the pulse when dedispersed at different dispersion measures. The final panel displays a dynamic spectrum, note the quadratic shape of the pulse.