,

LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Technical Note	**LIGO-T070000-00-Z**	2007/10/19

Coherent Waveburst Review Document

Michele Zanolin[1] (reviewer), Keith Riles[2] (reviewer), Brian O'Reilly[3] (reviewer), Sergey Klimenko[4] (author), Igor Yakushin[3] (author), Adam Mercer[4] (author)
[1]*Massachusetts Institute of Technology,* [2]*University of Michigan,*
[3]*LIGO-Livingston Observatory,* [4]*University of Florida*

Draft
Circulation restricted to LIGO I members

California Institute of Technology
LIGO Project, MS 18-34
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project, Room NW17-161
Cambridge, MA 02139
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

LIGO Hanford Observatory
Route 10, Mile Marker 2
Richland, WA 99352
Phone (509) 372-8106
Fax (509) 372-8137
E-mail: info@ligo.caltech.edu

LIGO Livingston Observatory
19100 LIGO Lane
Livingston, LA 70754
Phone (225) 686-3100
Fax (225) 686-7189
E-mail: info@ligo.caltech.edu

WWW: http://www.ligo.caltech.edu/

Contents

1 Review Process

In this report we document the review and tests of the Coherent WaveBurst alghorithm for the purpose of approving its use in searches for unmodelled bursts with the networks of gravitational waves detectors.
This document will also touch, at the end, the review of the command line version of the Coherent WB which is the coherent event display (CED) and the application of Coherent WB to high frequency searches.

The Review commettee has been meeting regurlarly on 2 hours calls every week from May 2006 (beside part of July and August) until February 2007. We focused on the following aspects:

1. Review of the constrained likelihood theory behind Coherent WB. This has been done studing the Coherent WB paper [1] and the Coherent WB note [2] . Particular effort has been put by the authors in the note in order to use the notations which are closer to the actual sofware implementation.
2. Review of the implementation of the alghorithm. Minor software bugs in the analysis code and job submission scripts have been found, fixed and documented.
3. We identified the relevant parts of the codes that needed to be involved in a walk throught because they were new. Some other parts that were identical to the inchoerent WB pipeline were not reviewed anew.
4. Based on the block structure of the code and the details of the theory we identified a set of tests to compare coherent and incoherent WB pipelines to specifically test and understand features of the Coherent WB implementation.
5. The documentation of this work is contained in this note and in e-log entries in the coherent network searches at:

   `http://www.lsc-group.phys.uwm.edu/bursts/investigations/s5index.html`

Many of the checks suggested by the reviewers had already been carried out by the authors. Others were promptly carried out at the reviewer's request.
In section 2 we describe the functions and scripts we choosed to involve in the walk through. In section 3 we describe the choice of the tests we felt necessary in order to feel comfortable in pubblishing S5 results. More tests can be envisioned (a list is presented at the end of this report); however, our approach has been to focus on those checks thought to be most important for the approvals of the ongoing analysis. In section 4 we discuss the Coherent ED and future applications of the Coherent WB.

2 Overview of the Coherent WaveBurst Algorithm

The coherent WaveBurst pipeline has essentially the same structure as the incoherent WaveBurst pipeline [6, 7] used in combination with the CorrPower algorithm [13].

Both pipelines share the same input, data conditioning and some of the output infrastructures, but the generation of burst triggers is different. In the incoherent pipeline the trigger generation is based on the time-frequency coincidence of the excess power between the detector data streams while in the coherent pipeline it is based on the network likelihood.

The coherent WaveBurst pipeline uses a likelihood method [1] for a coherent detection and reconstruction of burst signals. The likelihood analysis is performed in the wavelet (time-frequency) domain. The coherent WaveBurst algorithm uses several model independent constraints (so-called regulators). The regulators are selected depending on the objective of the burst analysis (un-triggered, triggered, all-sky, etc) and the network configuration. The details of the pipeline implementation are described in the coherent WaveBurst technical note [2].

The pipeline consists of two stages: a) the production stage, when the coherent burst triggers are generated for a network of GW detectors and b) the post-production stage, when additional selection cuts are applied to distinguish the GW candidates from the background events.

The layout of the production stage is shown in Figure 1. The pipeline reads data from several detectors, performs binary wavelet transformation and applyes data conditioning algorithms. This steps in the analysis are identical to the incoherent pipeline [8]. Similarly to the incoherent WaveBurst the coherent analysis is performed at several time-frequency resolutions.

For each resolution, first the likelihood time-frequency (LTF) maps are constructed. This is a coherent statistic which combines data from all detectors in the network for a given time-frequency location (LTF pixel). The construction of the LTF maps involves variation of the likelihood of each LTF pixel over all possible source coordinates selected for the analysis (typically 180 × 360 sky locations for the all-sky search). This step requires calculation of the time-delayed amplitudes in the wavelet domain, to take into account the time delays of the GW signal between the detectors for a given sky location. The time delays between the detector data streams in wavelet domain are introduced by means of the time-delay filters. The likelihood value of the LTF pixel is the estimation of the maximum possible SNR detected by the network at all selected sky locations. By setting a threshold on the LTF pixel likelihood, loudest pixels are selected for the analysis.

The selected LTF pixels are combined into clusters which represent the coherent burst triggers at a given wavelet resolution. The clustering algorithm used in the coherent Waveburst is identical to the one used in the incoherent pipeline to cluster pixels in the individual detectors. After the coherent triggers are reconstructed, the pipeline performes the constrained likelihood analysis for each trigger. During this step a complete coherent analysis is performed, including reconstruction of the GW polarizations and the source coordinates. Clusters reconstructed at different time-frequency resolutions are combined into *superclusters* if they overlap in time or frequency. The optimal TF resolution is defined by a cluster with the largest value of the likelihood. The parameters of this clusters, including coherent statistics such as likelihood, null energy, coherent energy, etc, are saved in a file for the post-production analysis.

During the post-production stage additional selection cuts may be applied to the reconstructed coherent triggers. The coherent Waveburst post-production stage deals with selection of "optimal" set of parameters and cuts to reject glitches (so-called consistency cuts). This selection cuts are entirely ad hoc and depend on the network configuration and the parameters of the search. For this reasons the Coherent WB pipeline is typically run at loose threshold and high output false alarm rate ($\sim$ 10 mHz), so the pipeline sensitivity is limited by the post-production selection cuts and there is no need to re-run the coherent algorithms to tune the pipeline. The selection cuts used in the coherent pipeline are:

- Cuts on the reconstructed energy $S_k = E_k - N_k$, where E_k is the total normalized energy in the output of k-th detector and N_k is its null stream (estimated energy of the noise). Several energy cuts are possible, depending how many detectors are used in the network and how strict is the energy cut. For example, by requiring that at least two detectors in the network have the reconstructed energy S_k greater than some threshold, the logic OR of all detector pairs can be selected. The most strict requirement is when all detectors in the network have S_k above the threshold. Although the coherent triggers are coincident in time by construction, this selection cut is the coincidence requirement between the detectors in the coherent analysis.

- Cut on the network likelihood C_{net} which is calculated as

  $$C_{net} = \frac{E_{coh}}{N + |E_{coh}|}, \tag{1}$$

  where E_{coh} is the total coherent energy detected by the network (one of the parameters calculated in the likelihood analysis) and N is the network null stream. In the coherent WaveBurst this cut may be used in combination with the excess power statistic (total detected SNR ρ):

  $$\rho_{eff} = \rho^{C}_{net}, \tag{2}$$

  where ρ_{eff} is the effective SNR.
  The review committee did not see any flaw in the conceptual design of the Coherent WB alghorithm.

3 CODE REVIEW

3.1 Data conditioning

Data conditioning: coherent WB uses the same data conditioning as the incoverent WB therefore we did not suggest any further tests. We also did not repeat the walkthrough described in the report of the review process of WB version 5.

3.2 Coherent waveburst implementation

The code is implemented in the Wavelet Analysis Tool package where the inchoerent pipeline is implemented as well. WAT is a C++ package.

The generation of the Coherent WB triggers it involves generating
a) delay filter in wavelet domain
b) likelihood TF plots
c) generation of coherent triggers
d) reconstruction of coherent events
e) time-shift analysis.
The Choerent WB is implemented in the following C++ classes:

1. skymap.hh was used to store two dimensional antenna patterns, time delays, network likelihood etc. For the usage of this function the review process focused on the correct calculation of the antenna patterns and that the angular resolution was sufficient enough for the upper frequency involved in the search.
2. netcluster.hh we reviewed this structure that contain the coherent triggers and the methods for the TF pattern recognition.
3. detector.hh we went over the definition of the class detector, which includes: (a) detector locations and orientations (b) antenna patterns (c) time delay skymap (c) time delay filter
4. network.hh is a class that Coherent WB uses define a network. It includes: (a) list of detectors and network parameters (b) construction of LTF and clusters (coherence3()). $coherence3(a_E, a_c)$ is one of the important methods of Coherent WB and needed to be revied carfeully. In fact a_E, the threshold on the average amplitude a_p of the LTF pixels, is the main handle to control the pipeline output rate, while a_c is used by corrcut (that we also reviewed) to remove loud glitches that are however grossly inconsistent between different IFOs (c) likelihood of clusters (likelihood3()).likelihood3(mode, core, aC) is the core of the implementation of the likelihood method, where mode is used to decide the flavour of the regulators to be used, 'core' distinguishes between using only core pixels or core plus halo, and a_c is a selection on properties of the core pixels.
5. wavearray.hh data containers used in the analysis.
6. wseries.hh data containers used in the analysis.

3.3 Walk through of .cc files

The headers we went over explicitly are listed in the previous section. Here we add some comments on the .cc files involved in the walkthrough. The first set of scripts we decided to review line by line are the WAT data containers (wavearray and WSeries). Even if they are the same as for incoherent WB it was a useful/necessary refreshment.

We also covered explicitly: skymap.cc
netcluster.cc
detector.cc
network.cc Here we focused onto the functions:
- coherence3() - coherent selection of TF pixels
- likelihood3() - likelihood reconstruction for clusters (likelihood3('l',true,1.67) - likelihood for core (loud) pixels in the cluster; likelihood3('l',false,1.67) - likelihood for all pixels in the cluster; likelihood3('L',false,1.67) - final likelihood calculation)
- setNDM() - calculation of the network likelihood matrix
likelihood is a quadratic form Ł = sum$_i j(Lij) = sum_i j(Cij*)$, where Lij is the likelihood matrix. Diagonal elements of Lij represent incoherent energy terms, off-diagonal are coherent energy terms.
- corrcut() - rejection of events based on the Parson's network cross-correlation. netpixel.cc

Some auxiliary tools involved already software already reviewed before:
Detector coordinates: coherent WB , used by GravEn
antenna patterns: GravEn/NetSim/cWB see for reference MDC validation
detector time delays: Zanolin/GravEn/cWB see for reference MDC validation

The main issues that were discovered and fixed are; ADD THE LIST HERE.

4 Running cWB jobs on LIGO clusters

The pipeline is implemented as a root script run in root shell, with output triggers into root files

```
root -l -b -q parameters.C net.C
```

5 Postproduction scripts

The post-production analysis is greatly simplified in the coherent WB pipeline. All trigger parameters and run specific information such as live time and list of software injections is generated at run time and saved in the root files. Therefore current postproduction scripts are used simply to produce cWB figures of merit: rates, reconstructed parameters, efficiency, etc. The scripts involved were:

```
a) initialization of root files and selection cuts:  netInit.C
b) simulation figures of merit :  sg20plot.C
c) rates :  netplot.C
```

6 List of Coherent WB tests

We suggest the following tests in order to validate the most critical parts of the coherent WB pipeline. **list all of them and the findings**

1. tests of the detector objects (THIS TEST WILL NOT CHANGE WITH THE CWB MODIFICATIONS):
   a) calculation of antenna patterns and time delays: we have already performed this test by comparison with the GravEn and Michele's antenna pattern code [9]
   b) validation of the delay filters: given a time series `x(t)`, the authors produce a wavelet series `w(t,f)`. Then the delay filter with delay T is applied to `w(t,f) -> w(t,f,T)`. The inverse wavelet transformation is applied to w(t,f,T) producing y(t+T). The time series y(t+T) is delayed by -T in time domain and compared with x(t). The description of this test was incorporated in the coherent WB note in section 3.3
2. test consistency of the likelihood algorithm (THIS TEST MIGHT CHANGE MILDLY WITH THE CWB MODIFICATIONS)): We have a builtin test (in the `network::getNDM()` method) which check reconstructed energy balance. For example, it holds that 2*L = E-N, where L is the likelihood, E is the total energy and N is the null stream. L and N are calculated independently and they can be checked with the relation above. NOTE: was network.cc updated in CVS ?
3. reconstruction of burst parameters (THIS TEST MIGHT CHANGE MILDLY WITH THE CWB MODIFICATIONS): standard figures of merit showing reconstructed parameters such as central time, frequency, hrss, etc are produced by post-production scripts and coherent display PUT HERE A CONNECTION WITH THE PLOT AT THE END OF THE DOCUMENT.
4. comparison of coherent and incoherent algorithms: a) extensive comparison of coherent and incoherent algorithms have been done with the S4 and S5 data. See section 6.2 for a detailed trigger by trigger comparison on SG235Q9 injections (The authors also did it for some time shifted triggers comparing reconstructed parameters which were quite consistent).

6.1 Reconstruction of High Q injections with low frequency

(THIS TEST MIGHT CHANGE MILDLY WITH THE CWB MODIFICATIONS) Incoerent version of WB was missing some low frequency - large Q injections. The authors showed that the problem is no longer present in the Coherent one.

Explicitly we asked the authors to investigate the effect of amplitude cut in incoherent WB for low frequency signals and perform a comparison with coherent WB (see figures 2,3,4,5). Efficiencies were analyzed with and without amplitude cut for 70 Hz Q=9,100, in the incoherent version of WB

What happens with the incoherent version of WB and the 70Hz sineGaussians is that 70Hz is very close to the lower frequency boundary of the search. Part of the energy of a signal is lost and this part is different for different detectors due to differences in their sensitivity curves. The greater Q is, the more energy of a signal is concentrated near the low frequency boundary and can be cut off. Detected and missed events for incoherent and coherent waveburst Below the results are given for SG70Q9 waveforms injected in S5a with the strain=1.8e-19 (the following parameters are provided in the list of detected events: time in L1, frequency in L1, hrss in L1,H1,H2):

```
injected 2943
detected by incoherent WB with AC 2810
detected by incoherent WB without AC 2921
detected by coherent WB 2919
```

For incoherent WaveBurst threshold 2.1 was used (no CorrPower cuts). For coherent WaveBurst threshold 4.5 was used (corresponding to the final background rate).

6.2 Comparison of coherent and incoherent triggers for SG235Q9

(THIS TEST MIGHT CHANGE MILDLY WITH THE CWB MODIFICATIONS)
* The set of coherent triggers is at the end of the whole coherent waveburst pipeline (before category 2-4 DQ or vetoes), background rate is 2e-8 Hz.
* The set of incoherent triggers is after incoherent waveburst and amplitude cut (before CorrPower, category 2-4 DQ or vetoes), backgroud rate is 2e-5 Hz.
* The total number of injections for each strain is 2952.

Given that the columns are: strain, coherent, incoherent, intersection, coherent-incoherent, incoherent-coherent. we have:

```
1.88e-22 46 7 6 40 1
2.65e-22 318 118 108 210 10
3.75e-22 888 502 487 401 15
5.27e-22 1428 1036 1029 399 7
7.45e-22 1826 1545 1534 292 11
1.05e-21 2119 1900 1894 225 6
1.49e-21 2377 2200 2193 184 7
2.1e-21 2561 2396 2389 172 7
2.98e-21 2691 2560 2555 136 5
4.2e-21 2771 2685 2683 88 2
5.92e-21 2824 2756 2749 75 7
```

```
8.37e-21 2869 2814 2804 65 10
1.18e-20 2896 2860 2855 41 5
1.67e-20 2905 2892 2884 21 8
2.36e-20 2917 2915 2906 11 9
3.33e-20 2919 2927 2913 6 14
4.68e-20 2920 2933 2914 6 19
6.6e-20 2924 2939 2921 3 18
9.3e-20 2926 2941 2923 3 18
1.31e-19 2931 2943 2928 3 15
1.84e-19 2937 2945 2935 2 10
```

* The agreement between coherent and incoherent triggers is quite good for loud injections. There are slightly more incoherent triggers but we do not know how many of them would be cut by CorrPower.
o One thing that at first looked surprising is the fact that all the 10 loud triggers found by the incoherent waveburst but missed by the coherent one have strong signal in L1 and very weak signal in H1 and H2.
The first hypothesis was that it is the cut on the pairwise energy used in the coherent waveburst that is responsible for missing these triggers. So the autors redid the calculations without that cut. However, it turned out that it would give only one extra loud trigger detected by the coherent waveburst.
The second hypothesis was that it is the cut on the network correlation that is responsible for this effect. So we relaxed that cut as well. That reduced the number of triggers missed by the coherent waveburst and detected by the incoherent waveburst from 10 to 2. Therefore the explanation is: when a signal is weak in both H1 and H2, the network correlation is weak as well since there is not a single pair of detectors with strong correlation (while L1 has a good signal, H1 and H2 have only noise). It also probably means that the CorrPower Gamma for such injections would be small as well since Gamma is an average of pairwise correlations. So there is a good chance that the complete incoherent waveburst + CorrPower pipeline would miss those injections as well since they are unlikely to pass waveform consistency tests. On the other hand, if a signal is strong in H1 and H2 but weak in L1, the network correlation and Gamma would be much higher due to the pairwise H1-H2 correlation coefficient value. That explains the observed asymmetry between sites for hrss of triggers missed by the coherent waveburst.
* For weak signals coherent waveburst is obviously more sensitive despite of the fact that the triggers come after waveform consistency test.

6.3 Study of missed loud injections

(THIS TEST MIGHT CHANGE MILDLY WITH THE CWB MODIFICATIONS)

Keith and Brian suggested to look at loud injections missed by the coherent pipeline. We expect that all of them should be associated with unfortunate polarization angle, which results in very different L1 and H1 responses and therefore, such injections are rejected by the constraint used in the coherent WB. Igor performed a test with loud SG235Q9 injections.

```
strain detected injected injected-detected
1.84e-19 2937 2953 16
```

The columns are:

```
71  816506161.627  816506161.624  194.1  118.7
    76.7  5.9e-20  1.1e-21  0.2  0.2  -0.32  0.0062
116  816671665.952  816671665.960  319.6  82.5
    59.1  4.9e-20  1e-21  0.39  0.061  -0.27  -0.0055
249  817261954.106  817261954.109  222.6  156.
  0  11.8  1.5e-20  5.3e-23  0.074  0.16  -0.08  -0.00028
331  817524267.614  817524267.614  233.7  95.9
    6.4  4.8e-20  2.6e-22  0.46  0.37  0.26  0.0014
377  817649861.020  817649861.018  280.7  21.5
    133.6  3.3e-20  7.9e-22  0.63  0.76  0.18  0.0043
403  817743371.029  817743371.032  228.8  131.
  6  101.1  1.6e-20  1.6e-22  0.025  0.00055  -0.085  0.00086
505  818231659.034  818231659.041  266.6  150.
  4  221.8  1.6e-20  1.9e-22  0.16  0.2  0.085  -0.001
584  818472768.303  818472768.300  137.7  133.
  2  21.0  5.8e-20  1.1e-21  0.62  0.46  0.31  -0.0062
1181  820414369.072  820414369.069  292.0  7.6
    58.1  3.3e-20  7.7e-23  0.42  0.6  -0.18  0.00042
1698  822195872.399  822195872.401  334.4  34.
  2  87.5  5.3e-20  7.3e-22  0.41  0.43  -0.29  0.004
1999  823519256.709  823519256.703  185.1  91.
  0  346.8  6.3e-20  1.5e-22  0.2  0.31  0.34  -0.00082
2119  824898867.672  824898867.676  40.2  128.
  2  274.9  1.6e-20  4.5e-22  0.58  0.91  -0.089  0.0024
2464  826619353.860  826619353.858  244.9  46.
  3  102.1  1.5e-21  1.2e-22  0.84  0.99  -0.0083  -0.00066
2627  827280572.592  827280572.594  214.3  134
 .1  142.5  2.7e-20  6.3e-22  0.022  0.031  0.14  -0.0034
2871  827919465.944  827919465.935  108.9  70.
  5  342.9  4.9e-20  2.3e-22  0.45  0.24  0.27  -0.0012
2894  827983868.253  827983868.258  321.2  56.
  5  339.3  6.2e-20  5.9e-22  0.58  0.34  0.33  0.0032
```

and show indeed that ...

7 Suggestions for future Coherent WB implementations

7.1 Effects of the down sampling frequency

In the future, if the upper limit of the search is 2000 the down sampling should be larger than 4096 to prevent from loss of efficienciy near the Nyquist frequency.

8 How to Run a WaveBurst Job

9 CVS

The current version of waveburst (used for the final analysis of the first year of S5) is tagged **wat-4_5_4** in the directory **Analysis/WaveBurst/S5/coherent** of **gds** CVS:

```
http://ldas-sw.ligo.caltech.edu/cgi-bin/cvsweb.cgi/Analysis/?cvsroot=GDS
```

10 Segments

The scripts and data files for S5 are located in CVS in **segments** directory (counting from the above path). **cat1** and **cat2** subdirectories contain category 1 and 2 segments for single detectors and a script to combine them into various detector networks (**combinations.py**).

Category 1 segments are divided into job segments using **jobs.sh** script that calls **break.pl** for each detector network segment list. Job segments can be between 300 and 600 seconds and overlap to take into account 8 second offset needed by WaveBurst. That also means that 8 second at the begining and end of each science segment is not used in the analysis.
The end result of the above segment processing are two files for each network:

```
cat1/S5year1-cat1-${network}_jobs.txt
cat2/S5year1-cat2-${network}.txt
```

where network can be, for example, H1H2L1, H1H2, H1L1, H2L1. WaveBurst jobs run on the first segment list while the second segment list is given as an input in the jobs configuration file to further downselect triggers during run time.

Notice: for S5 full year `S5year1-cat1-H1H2L1_jobs.txt` had to be modified manually: job 5714 had no H1 and H2 h(t) for the last 8 seconds, job 5715 had no H1 and H2 h(t) for the first 8 seconds.

11 Input files for a single job

The examples below are given for H1H2L1 network. Let us first consider a setup for production jobs in the directory `offline/PRODUCTION/S5_H1H2L1_run29/` First one needs to modify **pre.pl** file to specify input frames types, input and output directories. All the files generated at this stage go to the input directory (which are input to WaveBurst jobs) while the output directory is where the ROOT files from each job are stored.
For example

```
#!/usr/bin/perl

$start=$ARGV[0];
$stop=$ARGV[1];
$pid=$ARGV[2];

$INPUT_DIR="/archive/home/igor/INPUT_S5_H1H2L1_run29";
$OUTPUT_DIR="/archive/home/igor/OUTPUT_S5_H1H2L1_run29";
$OUTPUT_LABEL="S5_H1H2L1_run29";
$INPUT_LABEL="S5_H1H2L1_run29";
$incrementD=300;

‘mkdir -p $INPUT_DIR‘;
‘mkdir -p $OUTPUT_DIR‘;

‘echo "$pid  ${INPUT_DIR} ${INPUT_LABEL} ${OUTPUT_DIR} ${OUTPUT_LABEL}" > ${INPUT_DIR}/${pid}_${INPUT_LABEL}.in‘;

$cmd="./framelist.pl $start $stop H1_RDS_C03_L2 H 8 0 ${INPUT_DIR}/lho1.lst.$pid $incrementD";
print ‘$cmd‘;
$cmd="./framelist.pl $start $stop H2_RDS_C03_L2 H 8 0 ${INPUT_DIR}/lho2.lst.$pid $incrementD";
print ‘$cmd‘;
$cmd="./framelist.pl $start $stop L1_RDS_C03_L2 L 8 0 ${INPUT_DIR}/llo.lst.$pid $incrementD";
print ‘$cmd‘;
```

The above **pre.pl** file specifies that that we use `H1_RDS_C03_L2` frame type for H1, `H2_RDS_C03_L2` frame type for H2, `L1_RDS_C03_L2` for L1 and the list of frames for each job (numbered after the segment number) is stored in `$INPUT_DIR` in files `lho1.lst.${job number}`, `lho2.lst.${job number}`, `llo.lst.${job number}`
Let us take look inside on of the frame list files, for example, **lho1.lst.100**

```
4
128
816362832
816363169
16384
/data/node130/frames/S5/strain-L2/LHO/H-H1_RDS_C03_L2-8163/H-H1_RDS_C03_L2-816362779-128.gwf
/data/node130/frames/S5/strain-L2/LHO/H-H1_RDS_C03_L2-8163/H-H1_RDS_C03_L2-816362907-128.gwf
/data/node130/frames/S5/strain-L2/LHO/H-H1_RDS_C03_L2-8163/H-H1_RDS_C03_L2-816363035-128.gwf
/data/node130/frames/S5/strain-L2/LHO/H-H1_RDS_C03_L2-8163/H-H1_RDS_C03_L2-816363163-128.gwf
```

Ignore lines 2,5 which are not used anymore. The first line specifies the number of frame files that cover the segment given on lines 3-4. The frame files are listed starting from line 6.
The name of the “*.in” configuration file for a job is `${INPUT_DIR}/${pid}_${INPUT_LABEL}.in‘` and it can be used to run a single job (more details below) as

```
cat ${in} | ./net.sh
```

The “*.in” file consists of one line and its structure is obvious from **pre.pl**.

`$OUTPUT_LABEL` is used to derive the name of the output ROOT file (that also includes start time, duration, job id and, in case of simulations, a factor by which constant strain injections are multiplied). The input and output labels are typically chosen to be the same.

Usually, nothing else besides the above parameters need to be changed in **pre.pl**. To generate input files for each job, run

```
./generateIn.pl ${path}/S5year1-cat1-H1H2L1_jobs.txt | tee log
```

generateIn.pl calls **pre.pl** that calls **frameslist.pl** that uses LSCdataFind to produce the frame lists. For 27372 H1H2L1 segments, it would take about a day to generate all the input files.

If one runs a simulation, one must also generate input files for MDC frames. Couple lines are added to **pre.pl** for an MDC channel while the lines generating frame lists for ifo channels can be commented. Input/output labels, output directory need to be changed while input directory should remain the same (not to regenerate frame lists for ifo channels). Also MDC frame type and the list of ifo channels in MDC should be specified. For example, for HSG1_S5 simulations:

```
#!/usr/bin/perl

$start=$ARGV[0];
$stop=$ARGV[1];
$pid=$ARGV[2];

$INPUT_DIR="/archive/home/igor/INPUT_S5_H1H2L1_run29";
$OUTPUT_DIR="/archive/home/igor/OUTPUT_HSG1_S5_H1H2L1_run29";

$OUTPUT_LABEL="HSG1_S5_H1H2L1_run29";
$INPUT_LABEL="HSG1_S5_H1H2L1_run29";
$MDC_FRAMETYPE="HSG1_S5";
$MDC_IFOS="G,H,L,T,V";
$incrementD=300;

`mkdir -p $INPUT_DIR`;
`mkdir -p $OUTPUT_DIR`;

`echo "$pid  ${INPUT_DIR} ${INPUT_LABEL} ${OUTPUT_DIR} ${OUTPUT_LABEL}" > ${INPUT_DIR}/${pid}_${INPUT_LABEL}.in`;

#$cmd="./framelist.pl $start $stop H1_RDS_C03_L2 H 8 1 ${INPUT_DIR}/lho1.lst.$pid $incrementD";
#print `$cmd`;
#$cmd="./framelist.pl $start $stop H2_RDS_C03_L2 H 8 1 ${INPUT_DIR}/lho2.lst.$pid $incrementD";
#print `$cmd`;
#$cmd="./framelist.pl $start $stop L1_RDS_C03_L2 L 8 1 ${INPUT_DIR}/llo.lst.$pid $incrementD";
#print `$cmd`;

$start-=8;$stop+=8;
print `./framelist.pl $start $stop  ${MDC_FRAMETYPE} ${MDC_IFOS} 0 0 ${INPUT_DIR}/${INPUT_LABEL}.lst.$pid $incrementD`;
```

12 Submit file

Submit file tells Condor where to get stdin and executable, where to put stdout, stderr and log files.

```
universe = vanilla
getenv = true
priority = $(PRI)
executable = net.sh
input = /archive/home/igor/INPUT_S5_H1H2L1_run29/$(PID)_S5_H1H2L1_run29.in
output = /usr1/igor/$(PID)_S5_H1H2L1_run29.out
error = /usr1/igor/$(PID)_S5_H1H2L1_run29.err
log = /usr1/igor/waveburst/S5_H1H2L1_run29.log
notification = never
rank=memory
requirements=disk > 100000
queue
```

For example, the above `S5_H1H2L1_run25.sub` file tells Condor to run in a vanilla universe, use the same environment as a UNIX interactive session, execute **net.sh** which takes its stdin from `/archive/home/igor/INPUT_S5_H1H2L1_run29/$(PID)_S5_H1H2L1_run29.in` where PID is job id. stdout and stdin are put on a corresponding node into /usr1/igor. The Condor log entries are appended to `/usr1/igor/waveburst/S5_H1H2L1_run29.log`. The jobs prefer to run on virtual machines with maximum amount of memory available per Condor job.

13 DAG file

The dag file is very simple and each job there is independent. To generate it, edit generateDAG.pl to specify the correct submit file and the number of jobs. After that run

```
./generateDAG.pl > S5_H1H2L1_run29.dag
```

14 Parameter file

`par${network}.C` specifies waveburst parameters. Among the ones you might need to change are:

```
  int lags     = 101;      // number of lags including zero lag
  double step  = 3.125;    // time interval between lags [sec]
```

For simulations the number of lags would be 1.

shifts define a constant time shift between data streams in addition to the lags that are applied to the last ifo with the step given above:

```
                           //  L1  H1  H2  G1 constant time shifts
  double shift[3] =         {0., 0., 0.};
```

Simulation parameters define whether we are doing simulation or production and if simulation how many and what factors should be applied to injection stream:

```
  int simulation = 0;      // 1 for simulation, 0 for production
  int   nfactor = 1;       // number of strain factors
  double factors[] ={1.0}; // array of strain factors
```

data_dir defines the location of binary filter files on the nodes:

```
  char data_dir[]  = "/usr1/igor/waveburst/data64";
```

nodedir defines where the ROOT files are originally generated on the nodes (they are copied later from nodes to the output directory):

```
  char nodedir[] = "/usr1/igor";
```

Category 2 segment list is specified here:

```
  char finalSegmentList[]="S5year1-cat2-H1H2L1.txt";
```

For simulations we also need to provide the list of injections in the format used by MDC framework to put the injections that are inside category 2 segments into ROOT files so that one can easily compare what is injected with what is detected. Leave it blank for production triggers:

```
  char injectionList[]="";
```

The prefix for files with frame lists is given here for each channel:

```
  char fileNamesRaw[3][50]={"llo.lst.","lho1.lst.","lho2.lst."};
```

Names for gravitational wave and MDC channels:

```
  char channelNamesRaw[3][50]={"L1:LSC-STRAIN","H1:LSC-STRAIN","H2:LSC-STRAIN"};

  char channelNamesMDC[3][50]={"L1:GW-H","H1:GW-H","H2:GW-H"};
```

15 Running a single job

Edit net.sh file for the correct path, parameter and waveburst file names:

```
#!/bin/bash
dir=/archive/home/igor/S5/coherent/offline/PRODUCTION/S5_H1H2L1_run29
${ROOTSYS}/bin/root -b -q -l ${dir}/parL1H1H2.C ${dir}/net3.C
exit 0
```

Now you are ready to run a single job or a DAG. To run a single job with id PID

```
cat /archive/home/igor/INPUT_S5_H1H2L1_run29/$(PID)_S5_H1H2L1_run29.in | ./net.sh
```

16 Running DAG

To run the whole DAG:

```
condor_submit_dag S5_H1H2L1_run29.dag
```

One can monitor jobs with

```
condor_userprio -all
```

or

```
condor_q igor
```

with various options (`man condor_q`).

One can also look into stdout and stderr files on the nodes. `nodeN:/usr1` is mounted on ldas-grid or ldas-pcdev1 as `/data/nodeN`.

17 Merging files

Each job generates its own ROOT file (in the first year of S5 for H1H2L1 network there are 27372 jobs). For simulations multiply it by the number of factors (23). It is impossible to do postproduction with so many files. Fortunately one can easily merge them in ROOT.
First, construct a list of ROOT files to merge. For example:

```
mkdir /archive/home/igor/archive/home/igor/OUTPUT_HSG1_S5_H1H2L1_run29.merged
cd /archive/home/igor/archive/home/igor/OUTPUT_HSG1_S5_H1H2L1_run29.merged
nohup perl -e '@a=glob("/archive/home/igor/archive/home/igor/OUTPUT_HSG1_S5_H1H2L1_run29/*.root");print join("\n",@a),"\n"' > list 2>&1 &
```

Next copy **rootlogon.C**, **mergeMDC.C** (or **mergePROD.C** for production triggers), **merge.sh** into `/archive/home/igor/archive/home/igor/OUTPUT_HSG1_S5_H1H2L1_run29.merged`. Second edit **merge.sh** file:

```
echo list HWNB2_S5_H1H2L1_run29.root | root -l -b -q mergeMDC.C
echo "Finished"
```

The only thing that is modified is the suffix for the merged ROOT file names passed to **mergeMDC.C** via stdin. Next execute it

```
nohup ./merge.sh > log 2>&1 &
```

It might take couple hours for production full year of S5 triggers or most of a day for simulations triggers. For production triggers the following merged files are generated:

- `w_S5_H1H2L1_run29.root` contains **waveburst** tree;
- if data exceeds 2G, it would be divided into several files: `w_S5_H1H2L1_run29_1.root`, `w_S5_H1H2L1_run29_2.root`, ...
- `lt_S5_H1H2L1_run29.root` contains **liveTime** tree that is used to figure out livetime;
- `n_S5_H1H2L1_run29.root` describes noise;
- `v_S5_H1H2L1_run29.root` describes noise variability.

Typically we just use **waveburst** and **liveTime** trees in postproduction.

18 Working with ROOT files

Here is the simplest way to explore a ROOT file `w_S5_H1H2L1_run29.root`:

```
  root -l
  root [0] TFile f("OUTPUT_S5_H1H2L1_run29.merged.all/w_S5_H1H2L1_run29.root")
  root [1] f.ls()
    TFile**         OUTPUT_S5_H1H2L1_run29.merged.all/w_S5_H1H2L1_run29.root        chain files
    TFile*         OUTPUT_S5_H1H2L1_run29.merged.all/w_S5_H1H2L1_run29.root        chain files
      KEY: TTree    waveburst;3     waveburst
      KEY: TTree    waveburst;2     waveburst
  root [2] waveburst->Draw("frequency[0]")
  root [3] waveburst->Print()
  ...
  root [4] TBrowser *a=new TBrowser()
```

Print() command lists the leaves in the tree. Creating a **TBrowser** object opens a GUI ROOT file manager that allows to plot simple graphs with a click of a mouse.

The way we work with ROOT files is as follows. When ROOT session starts, it executes commands in **rootlogon.C** file in the current directory. In our **rootlogon.C** file we load **wavelet.so** library and various macros, in particular, **netInit.C**.

```
{

  TString rdir="/archive/home/igor/S5/coherent/wat/wat-4.5.4";
  TString ldir="/archive/home/igor/SOFT1/root/lib";
  TString aux="/archive/home/igor/S5/coherent/aux";

  printf("ROOT initialization starting.\n");
  printf("Loading Wavelet Analysis Tool library.\n");

  gSystem->Load(rdir+"/lib/wavelet.so");

  gSystem->Load(ldir+"/libFrame.so");
  gSystem->Load(ldir+"/libFrameROOT.so");
  gSystem->Load(ldir+"/libFrvROOT.so");

  printf("Loading WAT macros.\n");

  gROOT->LoadMacro(aux+"/readframes7b.C");
  gROOT->LoadMacro(rdir+"/macro/wbsingle.C");
  gROOT->LoadMacro(rdir+"/macro/variability.C");
  gROOT->LoadMacro(rdir+"/macro/xcsample.C");
  gROOT->LoadMacro(rdir+"/macro/wavenoise.C");
  gROOT->LoadMacro(rdir+"/macro/netevent.C");
  gROOT->LoadMacro(rdir+"/macro/injection.C");
  gROOT->LoadMacro(rdir+"/macro/livetime.C");
  gROOT->LoadMacro(rdir+"/macro/WTSpectrum.C");
  gROOT->LoadMacro("logNfit.C");
  gROOT->LoadMacro("Draw.C");
  gROOT->Macro("netInit.C");
  gROOT->LoadMacro("type2freq.C");
}
```

netInit.C loads ROOT files and defines postproduction cuts:

```
{
  //netInit.C for production triggers

  const int ni=3;
  char ifos[ni][3]={"L1","H1","H2"};

  TChain net("waveburst");
  TChain lt("liveTime");

  //Load ROOT files into TChain

  net.Add("w_S5_H1H2L1_run29.root");
  net.Add("w_S5_H1H2L1_run29_1.root");
  lt.Add("lt_S5_H1H2L1_run29.root");
  netevent  w(&net,ni);

  TString nonzero="lag[0]>0";
  char buffer[2048];
  sprintf(buffer,"sqrt(2*ecor)-sqrt((null[0]+null[1]+null[2]))");
  TString selection=TString(buffer);
  TString halfcut=selection;
  TString halfcut1=nonzero + " && " + halfcut;

  //Frequency dependent cut

  TString halfcut2=nonzero+" && ((frequency[0]<200 && "+selection+">6.2) || (frequency[0]>=200 && "+selection+">4.96))";
}
```

Now if one executes `root -l` in the directory with the above **rootlogon.C** and **netInit.C** files, one explore the remaining triggers:

```
waveburst->Draw("frequency[0]",halfcut2)
//threshold vs frequency
waveburst->Draw(selection+":frequency[0]",halfcut2,"colz")
//threshold vs LHO time
waveburst->Draw(selection+":time[1]",halfcut2,"colz")
```

To get rate vs threshold table, run the following from the production directory:

```
root -l -b -q rate_parameters.C rate.C
```

You might want to edit the list of thresholds to use.

To get a standard set of FOMs on production triggers, run from the production directory:

```
root -l summary_parameters.C plotSummary.C
```

Depending on the desired rate, you might want to adjust the first parameter **th** in **summary_parameters.C**.

To dump triggers into an ASCII file, run

```
root -l -b -q dump_parameters.C dump.C
```

You might want to adjust selection cut and output file name in **dump_parameters.C**

To get efficiency curves and injection reconstruction FOMs, run the following from the simulation directory:

```
root -l parameters.C mdc.C
```

References

[1] S.Klimenko, S.Mohanty, M.Rakhmanov, G.Mitselmakher,
Phys. Rev. D **72**, 122002 (2005).

[2] S.Klimenko, I.Yakushin, A. Mercer, and G.Mitselmakher,
Coherent WB note, T060282-00-Z

[3] Coherent WB and Links (Klimenko and Yakushin):

[4] S. Klimenko and G. Mitselmakher, Wavelet method for GW burst detection
Class. Quantum Grav. **21**, S1819 (2004).

[5] S. Klimenko I.Yakushin, and G. Mitselmakher*et al*, Preliminary S2 waveburst results
Class. Quantum Grav. **21**, S1685 (2004).

[6] S.Klimenko, I.Yakushin and G.Mitselmakher,
WaveBurst, LIGO note T040040-00-Z

[7] S.Klimenko, I.Yakushin and G.Mitselmakher,
WaveBurst, version 5, LIGO note T050222-00-Z

[8] S.Klimenko, I.Yakushin and G.Mitselmakher,
WaveBurst, S5 version, LIGO note T060112-00-Z

[9] `https://gravity.psu.edu/~s4/sims/BurstMDC/Validation/BurstMDC_Validation.html`

[10] `{ H.Siemens }`

`http://www.lsc-group.phys.uwm.edu/\~siemens/ht.html`

[11] S.Klimenko and I.Yakushin, Waveburst CVS,

`"http://www.lsc-group.phys.uwm.edu/cgi-bin/Analysis/Waveburst"`

[12] S.Klimenko and I.Yakushin, Wavelet analysis tool,

`"http://www.lsc-group.phys.uwm.edu/cgi-bin/Analysis/Waveburst/S4/wat",`

[13] L. Cadonati, Coherent waveform consistency test for LIGO burst candidates.
Class. Quantum Grav. **21**, S1695 (2004).

[14] S. Chatterji, L. Blackburn, G. Martin, E. Katsavounidis,
Multiresolution techniques for the detection of gravitational-wave bursts.
Class. Quantum Grav. **21**, S1809 (2004).

[15] B. Vidakovic, Statistical modeling by wavelets, 1999.

[16] W.Press et al., Numerical recipes, 1992

[17] R.Adhikary et al.,
Calibration of the LIGO detectors for the first LIGO scientific run, LIGO note T030097-00-D

[18] P.Sutton and K.Schlaufman, SenseMonitor,

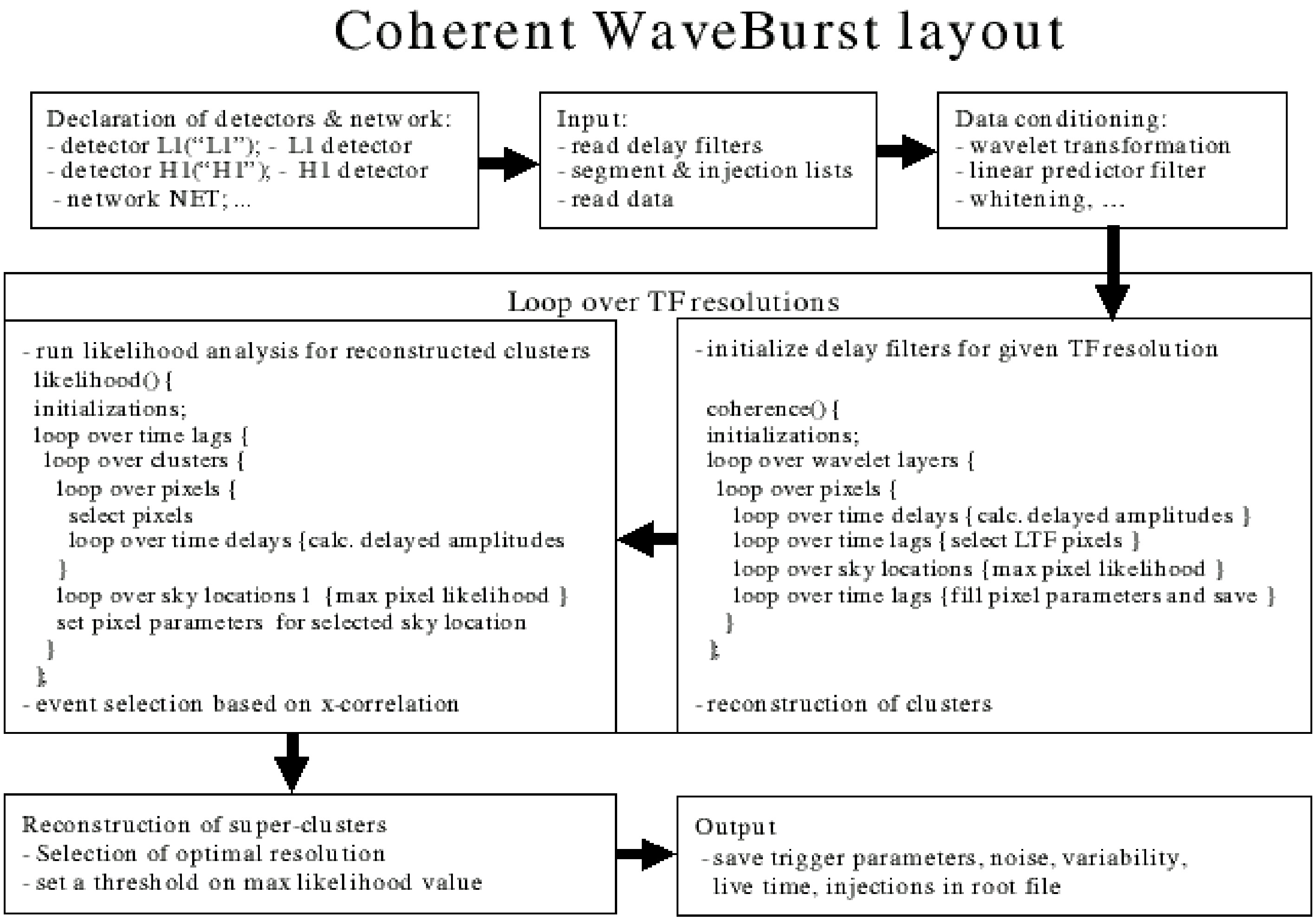
`http://blue.ligo-wa.caltech.edu/gds//dmt/Monitors/SenseMonitor`

[19] All LIGO coordinates are verbatim from LIGO-P000006-D-E:
Rev. Sci. Instrum., Vol. 72, No. 7, July 2001

[20] Anderson, Brady, Creighton, and Flanagan, PRD 63 042003, 2001

[21]
`http://www.lsc-group.phys.uwm.edu/cgi-bin/bag-enote.pl?nb=burs4simulations&actio`

[22] L. Wen and B. F. Schutz,Coherent network detection of gravitational waves: the redundancy veto,
Class. Quantum Grav. **22**, S1321 (2005).


Coherent WaveBurst layout
Declaration of detectors & network:
- detector L1("L1"); - L1 detector
- detector H1("H1"); - H1 detector
- network NET; ...
Input:
- read delay filters
- segment & injection lists
- read data
Data conditioning:
- wavelet transformation
- linear predictor filter
- whitening, ...
Loop over TF resolutions
Reconstruction of super-clusters
- Selection of optimal resolution
- set a threshold on max likelihood value
Output
- save trigger parameters, noise, variability,
live time, injections in root file

```
-initialize delay filters for given TFresolution

 coherence() {
 initializations;
 loop over wavelet layers {
  loop over pixels {
    loop over time delays {calc. delayed amplitudes }
    loop over time lags {select LTF pixels }
    loop over sky locations {max pixel likelihood }
    loop over time lags {fill pixel parameters and save }
   }
 }

-reconstruction of clusters
```

```
- run likelihood analysis for reconstructed clusters
 likelihood() {
 initializations;
 loop over time lags {
  loop over clusters {
   loop over pixels {
     select pixels
     loop over time delays {calc. delayed amplitudes
    }
   loop over sky locations 1 {max pixel likelihood }
   set pixel parameters for selected sky location
  }
 }
- event selection based on x-correlation
```

Figure 1: Coherent WB layout.

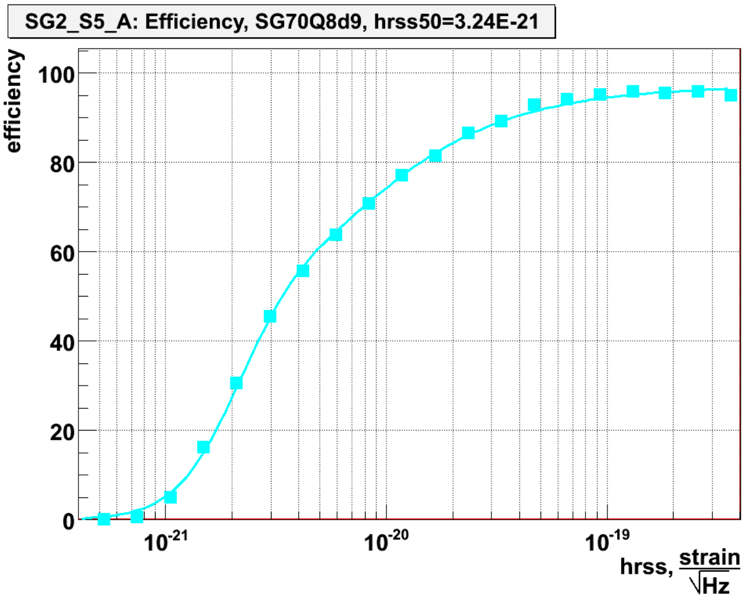


Figure 2: Efficiency SG70Q8d9 with A.C. .

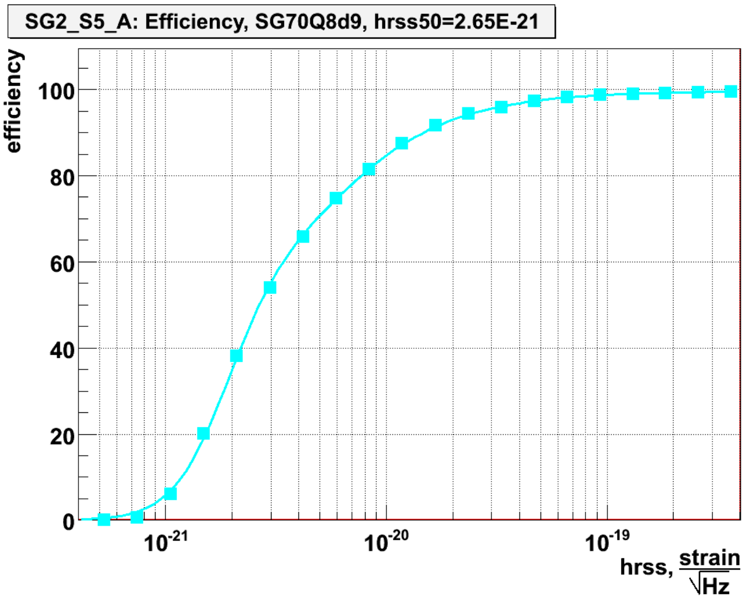


Figure 3: Efficiency SG70Q8d9 Coherent WB (without A.C.) .

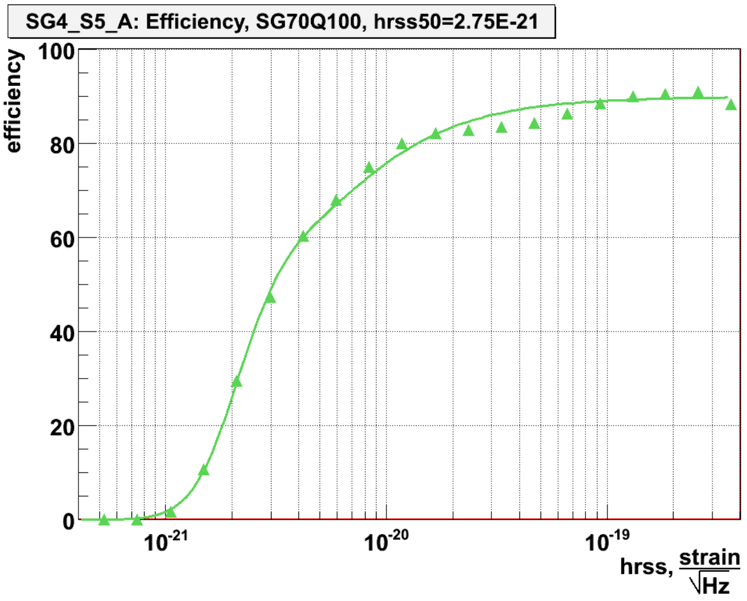


Figure 4: Efficiency SG70Q8d9 with A.C. .

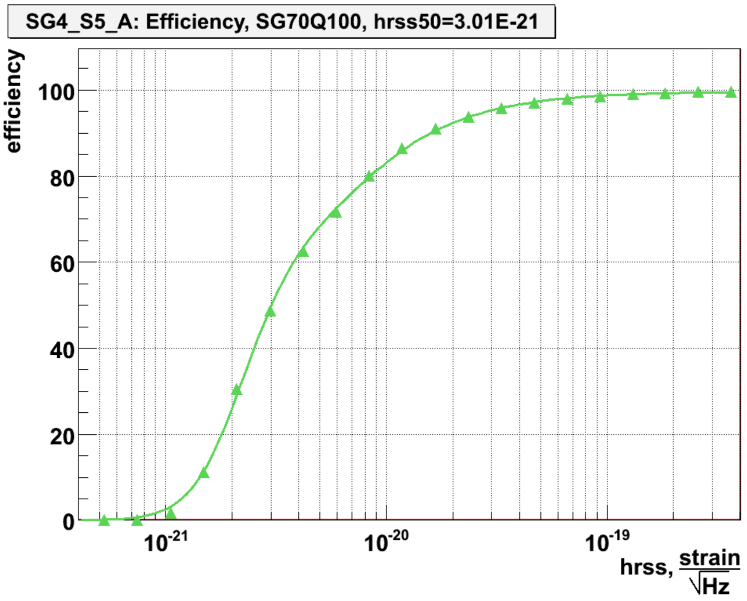


Figure 5: Efficiency SG70Q8d9 Coherent WB (without A.C.) .

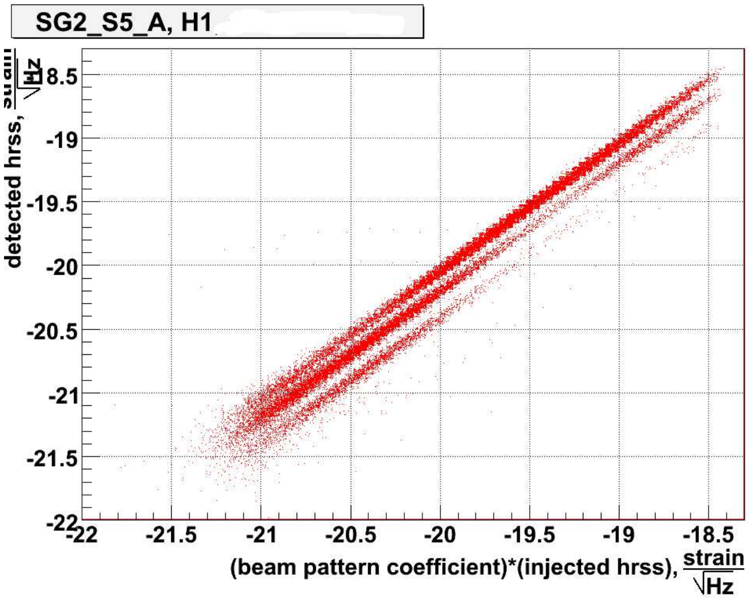


Figure 6: reconstructed hrss.

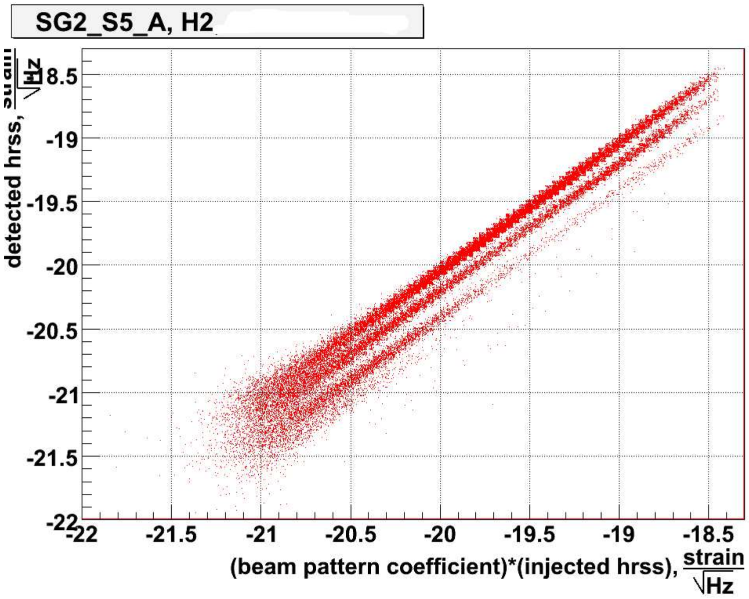


Figure 7: reconstructed hrss. .

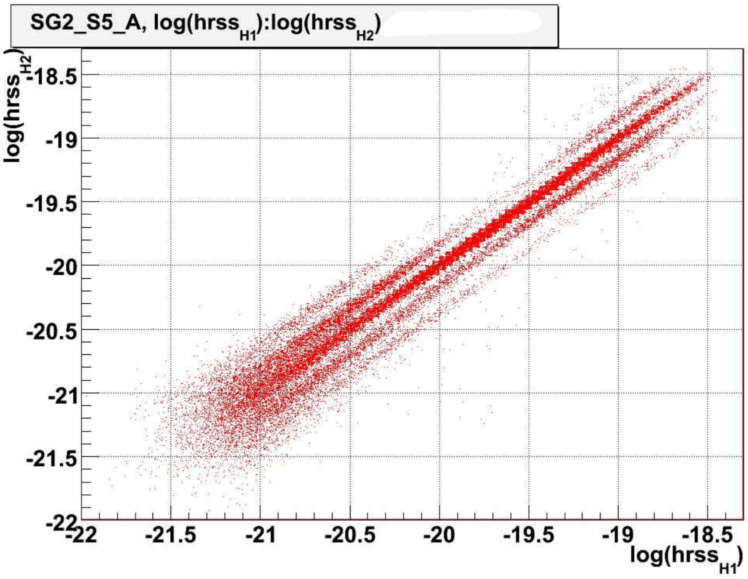


Figure 8: reconstructed hrss..

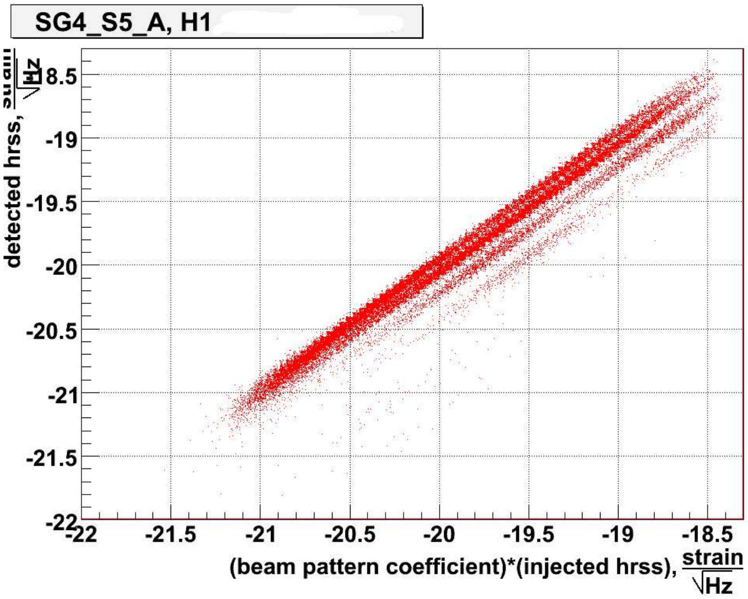


Figure 9: reconstructed hrss. .

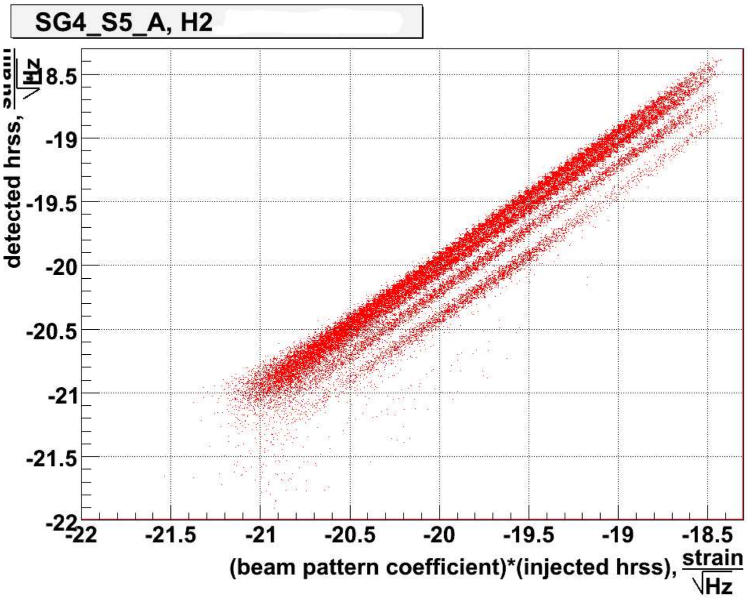


Figure 10: reconstructed hrss. .

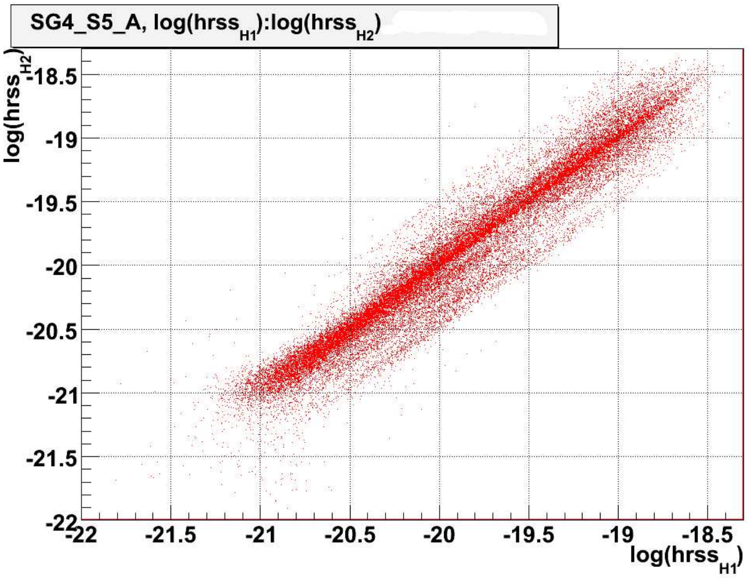


Figure 11: reconstructed hrss. .

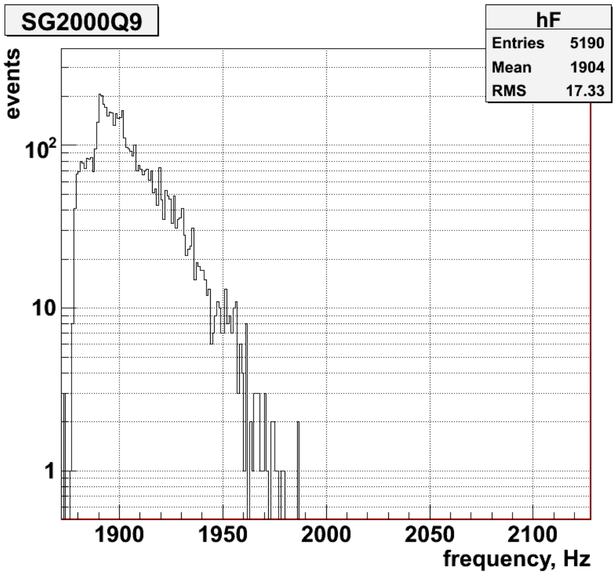


Figure 12: reconstructed frequency for SG2000Q9. The slope effect is a consequence of the downsampling to 4096Hz and consequently having the Nyquist frequency equal to the higher frequency of the search