

# Proposed Burst pipeline test plan

Alan Weinstein, 9/17/01; Sam Finn, 9/28/01.

This is more of a development plan than a test plan, since there is so much more development to do. The testing of existing features should proceed in parallel with, and drive, the development. Some of these tests can proceed in parallel, but most cannot, because they necessarily build on the success of previous tests. At the moment, these tests are in descriptive form; detailed descriptions come later.

Ideally, a stable and dedicated LDAS installation (like MIT) should be available to the burst group for this work.

## What we are working towards (strawman scenario):

- **Burst Filtering**: Each (of 3) IFO data streams run through pipeline independently, in their own dedicated LDAS installation, populating their own LDAS MetaDB with burst event triggers (*burst\_sngl* table).
  - At first, this process will be "offline", using archived engineering data. It will then run "on-line", keeping up with the data-taking.
  - In principle, all that is needed is the (a) GW channel, and the IFO response function. This can be extracted from full frames staged to disk, or from prepared very-reduced datasets (VRDS's). We will definitely want VRDS's eventually, to allow collaborators who don't have TB's of disk space go through the data at their home institutions.
  - the noise PSD is calculated "on the fly" by the datacondAPI.
  - Something like a tcl script can query the MetaDB to find in-lock segments, make sure the frames are staged to disk, and submit LDAS jobs, one after the other, to process segments (eg, 512 second's worth per job). This automated process should keep track of output, processor timing, etc; identify crashes and initiate recovery; provide summary statistics in human-readable form. It should be capable of operating "DC", pausing appropriately if/when it gets ahead of data taking, and providing email alarms if things go awry.
  - Burst event filters should, at this stage, operate with a relatively low threshold, tuned to produce an event rate which the MetaDB can handle (eg, 0.1 Hz or less).
  - Burst event MetaDB entries should give as complete information as possible: GPS time, duration, bandwidth, S/N, and estimated hrms.
- **Veto triggers**: Separately, raw or RDS frames containing many channels are fed into DMT filters, generating triggers in the MetaDB which can be used to veto events (reducing fake rate at the expense of livetime).
  - At this stage, many potential veto triggers can be formed; their efficacy can be evaluated in a later step.

- Here again, tcl scripts should automate the feeding of data into the DMT filters; keep track of output, processor timing, etc; identify crashes and initiate recovery; provide summary statistics in human-readable form.
- **Statistical and Coincidence analysis**: Offline (later, nearly on-line), event trigger and veto trigger MetaDB entries are pulled from MetaDB's of all IFO LDAS installations, to compile statistics and look for coincidences.
  - Single-IFO *sngl\_burst* trigger clusters are merged into single events, making use of duration information (separately for each DSO algorithm, at least until we understand how/if to combine/OR them).
  - Single-IFO (fake) event rate is evaluated as a function of S/N threshold (separately for each DSO algorithm).
  - Single-IFO efficiency for embedded signals is evaluated as a function of  $h_{\text{rms}}$  for fixed S/N thresholds, or S/N threshold for fixed  $h_{\text{rms}}$  (separately for each DSO algorithm).
  - Each source of DMT trigger veto is evaluated for efficacy: reduction in fake rate vs reduction in livetime; as a function of any threshold cut that trigger may possess. Efficacious vetoes (those that reduce the fake rate significantly with small loss of livetime) are applied, and the fake rate re-measured for each DSO algorithm.
  - Finally, un-vetoed event lists are compared between different IFOs, coincident events found, and the coincident fake rate evaluated.
  - **“Orthogonality”**: The events, fake and otherwise, that are generated by the different DSOs are compared to determine the relative effectiveness of the different searches to events of different types. E.g., does one DSO catch everything caught by another DSO? Do two different DSOs see mostly different things? How independent are the tests associated with each DSO on injected bursts of different kinds, or on false alarms?
    - We need here a set of burst signals, of different character, that in some sense span the space of signals we believe we should be sensible to. These would be injected into the data to determine the efficiency of different searches (as a function of threshold and thus false rate) to sources of this type.
    - This set should include, at a minimum, representative samples from the ZM catalog, Gaussian modulated sine waves, exponentially damped sinusoids (i.e., ring-downs), and white-noise bursts with a 1 KHz bandwidth. The bandwidth-duration product should be limited to be less than TBD (10? 20? 50? 100?) and scale free (i.e., distributed in bandwidth-duration space in a manner independent of bandwidth, duration, or characteristic frequency).
- **Cross-correlation**: Raw data streams from 2 or more IFOs in 512-sec intervals centered around a coincidence are cross-correlated, and additional cuts applied if the signals are not consistent with one another.

## Desired Outcomes:

- For each DSO
  - Fake rate as function of threshold
  - Efficiency vs fake rate for a set of “standard candle” burst sources
  - Each on “real data” as well as simulated data
- For each DMT trigger
  - Live-time vs. threshold on real data as well as (normally distributed) simulated data
- For subsets of DSOs and DMT trigger
  - Fraction of DSO events also identified by DMT triggers (on PEM or non-strain detector channels) when applied to simulated data or engineering data without any signals present (want a high fraction here).
  - Fraction of DMT triggers (on PEM or non-strain detector channels) that also appear as DSO events when applied to simulated data or engineering data without any signals present (want a high fraction here). An interesting figure of merit might be the product of these two statistics, which would be high for a good DMT trigger and low for a poor one. There is an optimization problem that can be solved here, since this product is a function of the DSO and DMT thresholds.
  - Fraction of all DSO triggers, not excluded by DMT triggers, that result in two-DSO, three-DSO, more-DSO coincidences. This is interesting for injected signals of different kinds.
- DSO Calibration
  - For each DSO, relationship between DSO “S/N” and burst normalization ( $h_{\text{RMS}}$ ) for bursts of different types.
  - Signature of bursts of different types in different DSOs (bursts of different types will appear differently in the same DSO: e.g., will have different appearances in the time-frequency plane in the tfcluster analysis. Can use this to identify not only that there has been a burst, but to classify the burst type and identify, for that type of burst, what the overall amplitude is.)

## Tests of LDAS pipeline:

Say we want to go through an entire E6 run, one IFO, with one LDAS installation. We'd need the entire dataset staged to disk (either RDS or Very-RDS), either all-at-once or in a "ring-buffer" keeping up with the LDAS jobs. We'll need some tcl or perl script to check the MetaDB to select N-second stretches of only good in-lock data; submit LDAS jobs, do bookkeeping on the jobs, check and parse output, etc.

- Run lots of jobs through lots of engineering data. (The LDAS crew should make large RDS samples available on disk for this).
- The pipeline should not crash.
- Check and keep records of job performance (cpu secs, number of beowulf cpu's, disk load, IO load, MetaDB load, etc).

## Tests of DatacondAPI:

The DatacondAPI must perform the following tasks for each DSO algorithm:

- resample raw GW data
- optionally fully whiten the GW data. Remove lines? At what level? By what mechanism can it determine the parameters of the line noise at that time?
- pass to the WrapperAPI a useful response function ADC/h (as a spectrum table? A pole/zero model?) which includes the effect of any extra whitening. This requires input from the detector team, promptly after data taking.
- Accumulate and pass to the WrapperAPI a noise PSD
- What else?

Test that for all DSOs, the DatacondAPI is supplying these things in a useable manner, and that they make sense at some level. Eg, report the mean and sigma of the GW data handed to the WrapperAPI. Report the  $h_{rms}$  noise in the 200-300 Hz region by applying the response function. (although each DSO may be sensitive in different frequency regions and may thus want to check the noise level for itself, a standard check across DSOs is a useful monitor of the data quality at the time of an LDAS job).

This presumably requires a bit of extra code (in LAL), to be called within each DSO, to test that the data passed by the DatacondAPI is sensible.

## Tests of the DSO search algorithm:

The 3 existing burst DSOs (power, tfcluster, slope) all work, but they likely can benefit from adopting the best of the other approaches. They should endeavor to use common (LAL) code for common calculations:

- To parse the input data passed to it from DatacondAPI, and perform checks (see above) to ensure that the data is of sufficient quality.
- To use the IFO response function to calculate the band-limited  $h_{rms}$  for the triggered signal.
- To construct the event trigger MetaDB entry.

Tests using Eng. Run data:

- Test the correctness of the MetaDB trigger entries:
  - how many entries/sec, in which tables?
  - What do the entries look like, are the numbers sensible?
- What is the optimal resampling rate (2048?)? Should not depend on DSO, only on signal spectrum and noise PSD.
- What is the optimal time interval for each LDAS job? (512 sec,  $2^{23}$  samples at 16384?) Too short, and fake rate, even efficiency, could suffer due to edge

effects. Too short or too long, job time will suffer. Plot average job time vs ifo time.

- CPU time required? Optimal number of Beowulf nodes?
- What parameters need optimizing? Optimize by minimizing fake rate while still triggering efficiently on sufficiently large injected burst signals.
- Evaluate the fake rate vs threshold.
- Evaluate the efficiency vs  $h_{\text{rms}}$  of some standard injected signals (eg, ZM catalog) for some threshold
- Evaluate the accuracy of the computed  $h_{\text{rms}}$  compared with injected signal.

### **Tests of DMT vetoes:**

We will have a list of DMT triggers that are candidates for event vetos. Each will veto for a certain specified duration. For each, we measure their rate, histogram their duration, and determine the resulting livetime fraction, were they to be employed to veto GW signals.

- For each, we measure the resulting reduction in fake rate (for each DSO algorithm) and reduction in efficiency.
- From this, we can determine which DMT triggers are most effective as event vetos.
- The MetaDB must be populated; must run DMT filters through a large chunk of Eng data.
- Then, this analysis can be performed on MetaDB entries only, eg, in ROOT.

### **Tests of Coincidence (Network) analysis:**

Much effort must go into studying correlations between different interferometers, especially the rate of coincident (but fake) burst triggers. This requires coincident data taking (eg, E6 data). There, it is to be assumed that any coincidences are either random or due to terrestrial-induced correlations, because the poor sensitivity of the IFOs means that real signals are unlikely. As the IFO sensitivities improve, this will no longer be a valid assumption, and analysis mechanisms must be in place to cross-correlate the data streams.

- Develop time-delay coincidence analysis to distinguish random coincidences from those due to real terrestrial-induced correlations.
- Study ways to reduce the fake coincidences (at the event trigger level) by, eg, requiring consistent  $h_{\text{rms}}$ , making use of DMT triggers that were not used for single IFO vetos but which are appropriate for multi-IFO coincidences, etc.
- Develop automated mechanisms for bringing 512-sec (for example) snippets of data from multiple IFOs around a coincident event, compute their cross-correlation, and use this info to suppress fake coincidences.

- Finally, setting a tolerable false coincidence rate of, eg, 1 per decade, go back and re-adjust single-IFO fake rates (by adjusting the S/N thresholds or other parameters) accordingly.

### Short-term plan:

- Questionnaires to DSO authors (see below)
- Back up, then zero, the mit\_test database.
- Prepare as much as one day of (readable; RDS?) engineering run frames.
- Prepare to run DMT on these data, populate DB with triggers that could be used for veto.
- Run pipelines (all 3 DSOs), just a few test jobs to check easy things (like the things checked during the 9/7/01 MDC):
  - Is the code getting the data from the DatacondAPI?
  - Does the Datacond data make sense?
  - Are database entries being created, and are they getting into the DB?
  - Do the DB entries make sense?
  - Exercise all 3 DSOs with zero, white and colored noise, without and with big injected signals. Are the signals found?
- Run pipelines (all 3 DSOs) on lots of data, measure fake rate.
- Run on data with injected signals; are they found?
- Run on data with injected signals of different amplitudes; does snr scale linearly?
- Etc...

### Questionnaire for DSO authors:

- Does the code run smoothly on the Beowulf without crashing?
- How long does it take to search through N seconds of data? How much CPU time required? Optimal number of Beowulf nodes?
- What does your code need from the DatacondAPI? Now, and potentially in the future?
- Does the code check the data from Datacond for correctness (not just the existence of the correct data structures, but also that the data structures make sense: eg, mean and std of GW channel, noise amplitude at 200-300 Hz, useful form for the response function). Do you have code to do these checks routinely and quickly, and can it be available to the other DSOs as a standard check?
- Does the code construct trigger entries for the MetaDB? Does it fill all the quantities with meaningful information? Can this process be standardized for all DSOs with shared code?
  - GPS time: how “accurate” is the burst start time?
  - Duration
  - central\_freq

- bandwidth
- amplitude – how is this defined in your code? Shall we create a “standard” calculation, shared by all DSOs, taking as input the estimated duration and bandwidth, and outputting “band-limited h\_rms”?
- snr
- confidence – how is this defined in your code?
- What is the optimal resampling rate (2048?)? Should not depend on DSO, only on signal spectrum and noise PSD.
- What is the optimal time interval for each LDAS job? (512 sec, 2<sup>23</sup> samples at 16384?) Too short, and fake rate, even efficiency, will suffer. Too short or too long, job time will suffer. Plot average job time vs ifo time.
- What parameters need optimizing? Optimize by minimizing fake rate while still triggering efficiently on sufficiently large injected burst signals.
- What development work do you foresee in the near future and longer-term? What machine resources do you need? What human resources do you need?
- What other issues should the Bursts working group address?

## **Request to LDAS:**

We request that LDAS management and programmers supply tools to accomplish the following:

- Something like a tcl script to query the MetaDB to find in-lock segments, make sure the frames are staged to disk, and submit LDAS jobs, one after the other, to process segments (eg, 512 second's worth per job).
- This automated process should keep track of output, real and processor timing, beowulf resources used, etc; identify crashes and initiate recovery; summarize output files and MetaDB table entries; provide any other relevant summary statistics, in human-readable form (like a web page or log file; emails are cumbersome because they have to be processed, merged, etc).
- It should be capable of operating "DC", pausing appropriately if/when it gets ahead of data taking, and providing email alarms if things go awry.