Villanova University, Villanova, PA, 19085

VET Instruction Manual

Michael C. Davis

July 13, 2023

The Veto Evaluation Tool (VET) measures the efficiency, deadtime, and use percentage of a data quality (DQ) flag against a set of events. A DQ flag indicates times when a specific problem is known to be occurring. A DQ veto is a time that is removed from the search and a flag is applied.

Efficiency is the fraction of the background events that were vetoes. Deadtime is the fraction of the analysis time that was removed. Use percentage is the fraction of auxiliary channel glitches used to veto. The ratio of efficiency over deadtime indicates the quality of the data versus how much time was removed. It is better to have as little time as possible removed from vetoes, so a useful veto results in a higher ratio of efficiency to deadtime (around eight to ten).

VET works on a single detector and a single event trigger generator (ETG) at a time. VET can be run for a flag to determine which has the best efficiency/deadtime ratio and to remove the majority of the poor data.

VET is run against the cWB results with both Hierarchical Veto (Hveto) and Used Percentage Veto (UPV) to identify unique glitch features. The final VET page provides tabs that display results for the entire day of Hveto and UPV as well as individual channel winners for that day.

1 Tutorial

The file `UPVandHvetoVET.py`, which can be viewed at https://github.com/mcdavis62/VET, contains every function that is required to execute VET against cWB results. It is necessary to enter commands first to allow VET to run properly. First, in the terminal window enter `kinit` and enter your albert.einstein password. Then, enter `ligo-proxy-init -k`

The `Wrapper()` function will execute VET on a single day or many days within the same month. No other libraries are required to execute VET. `Wrapper` requires the following input parameters:

- IFO - the detector, either H1 or L1 (string)

- year - the year (4 digit integer)
- month - the month (1 or 2 digit integer)
- day - the day (1 or 2 digit integer) or an array for many days in a month
- dest_dir - the destination directory, where you want you files to be created, default is your current directory, `'.'` (string)
- verbose - switch to display progress, default is `False` (boolean)

The following can be used to execute VET on the single day January 5^{th}, 2020 at the Hanford detector, H1, with `verbose` switched on:

```
from UPVandHvetoVET import *

Wrapper('H1', 2020, 1, 5, dest_dir='.', verbose = True)
```

The following can be used to execute VET on the days January 5^{th} through January 15^{th}, 2020 at the Livingston detector, L1, with `verbose` switched off:

```
from UPVandHvetoVET import *
import numpy as np

Wrapper('L1', 2020, 1, np.arange(5,16,1), dest_dir='.', verbose = False)
```

Once the VET has been completed, you will be able to view the resulting webpage by going to:

- **Caltech**: https://ldas-jobs.ligo.caltech.edu/~albert.einstein/[your directory]
- **LLO**: https://ldas-jobs.ligo-la.caltech.edu/~albert.einstein/[your directory]
- **LHO**: https://ldas-jobs.ligo-wa.caltech.edu/~albert.einstein/[your directory]

Clicking on the folder with the GPS time(s) you used will lead you to the VET page that is the same style as the summary pages. Click on the Hveto or UPV tabs at the top of the page to view the results.

2 Functions

This section will provide an overview of the functions included in the `UPVandHvetoVET.py` package and highlight possible links or directories that could change in the future. When path names are listed, directories labeled with brackets and all caps are the date chosen. Note: all of these functions can be run on one single day or on a month of days – the `Wrapper()` function does everything required to make this switch.

2.1 UPV_scraper

UPV_scraper() queries the UPV daily result files to make the following files needed to execute VET: `UPV.txt`, `UPV_merge.txt`, `UPV_segs.txt`, and `UPV_segs.txt`. These files are made for the entire day's worth of data as well as the individual channel winners. Therefore, each day will provide a set of these .txt files as well as additional sets for every channel winner of that day for UPV. UPV_scraper() determines the number of round winners on the day

and stores those channel names. The function also creates the dictionary required to hold the UPV flag files and UPV flag names used later.
UPV_scraper() gathers the UPV veto segments from the following path:

`/home/detchar/public_html/NewUPV/O3/results/O3-H-[GPS_TIME_START]_[GPS_TIME_END]/`.

The UPV results are also found at the following webpage:

https://ldas-jobs.ligo-la.caltech.edu/~detchar/NewUPV/O3/results/.

2.2 UPV_sort_on_col()

UPV_sort_on_col() sorts a data array based on the values in a single column. It creates the `UPV_sorted.txt` file containing the UPV veto times in ascending order. It provides a `UPV_sorted.txt` file for the entire day's data as well as the individual channels. `UPV_sorted.txt` is used with the make_flag_xml() function.

2.3 query_UPV_flag()

query_UPV_flag() queries the UPV daily result files and makes the path to the `UPV_sorted.txt` file. query_UPV_flag() creates the path from the following directory:

`/home/detchar/public_html/NewUPV/O3/results/O3-H-[GPS_TIME_START]_[GPS_TIME_END]/[IFO]:GDS-CALIB_STRAIN`

2.4 hveto()

hveto() queries the Hveto daily result files to make the following files needed to execute VET: `hveto.txt`, `hveto_merge.txt`, `hveto_segs.txt`, and `hveto_segs.txt`. These files are made for the entire day's worth of data as well as the individual channel winners. Therefore, each day will provide a set of these .txt files as well as additional sets for every channel winner of that day for Hveto. hveto() determines the number of round winners on the day and stores those channel names. The function also creates the dictionary required to hold the Hveto flag files and Hveto flag names used later.
hveto() gathers the Hveto veto segments from the following path:

`/home/detchar/public_html/hveto/day/[yyyymmdd]/latest/segments/`

hveto() loads the summary statistics .txt file from the following path:

`/home/detchar/public_html/hveto/day/[yyyymmdd]/latest/summary-stats.txt`

hveto() collects the veto files for each individual channel round (for a round N) from the following path:

`/home/detchar/public_html/hveto/day/[yyyymmdd]/latest/segments/[IFO]-HVETO_VETO_SEGS_ROUND_[N]-[GPS_TIME_START]-86400.txt`

2.5 query_hveto_flag()

query_UPV_flag() queries the Hveto daily result files and makes the path to the `hveto_sorted.txt` file. query_hveto_flag() creates the path from the following directory:

`/home/detchar/public_html/hveto/day/[yyyymmdd]/latest/segments`

2.6 hveto_sort_on_col()

hveto_sort_on_col() sorts a data array based on the values in a single column. It creates the `hveto_sorted.txt` file containing the Hveto veto times in ascending order. It provides a `hveto_sorted.txt` file for the entire day's data as well as the individual channels. `hveto_sorted.txt` is used with the make_flag_xml() function.

2.7 make_flag_xml()

make_flag_xml() makes candidate flag xml files. Within the Wrapper() function, it is run twice – one for UPV and one for Hveto. It makes flag names (with the `:1` format) and the flag file (with the `.xml` format).

2.8 query_cwb_events()

query_cwb_events() queries the cWB EVENTS file from a CIT cluster. It creates the `cwb.lcf` and the the `EVENTS.txt` files. The path that query_cwb_events() will retrieve the `EVENTS.txt` from will change depending on the observing run. This function will be the one that will cause the most issues with incorrect directories.
query_cwb_events() gets the `EVENTS.txt` file from the following path:

for O3a: `/home/waveburst/public_html/online/O3_LH_BurstLF_ONLINE/POSTPRODUCTION/FOM_daily_[GPS_TIME_START]_[GPS_TIME_END]/plotbin2_cut/data/EVENTS.txt`

for O3b: /home/waveburst/public_html/online/O3b_LH_BurstLF_ONLINE/POSTPRODUCTION/FOM_daily_[GPS_TIME_START]_[GPS_TIME_END]/plotbin2_cut/data/EVENTS.txt

The `EVENTS.txt` file needs to be retrieved from Caltech, so if VET is being executed from another location, the following command must be used with query_cwb_events():

`scp ldas-pcdev1.ligo.caltech.edu:  [PATH]`

The key fingerprint for that server can change and this may cause the EVENTS file to not be transferred. To prevent this, enter the following path and enter "yes" to the question *Are you sure you want to continue connecting (yes/no)?*:

scp ldas-pcdev1.ligo.caltech.edu:/home/waveburst/public_html/online/O3b_LH_BurstLF_ONLINE/POSTPRODUCTION/FOM_daily_[GPS_TIME_START]_[GPS_TIME_END]/plotbin2_cut/data/EVENTS.txt [DESTINATION_DIRECTORY]

2.9 Write_config()

Write_config() writes a file to perform a VET run with desired parameters, without using DQSEGDB. It creates the `cwb_daily.ini` file, the VET configuration file that needs to be specified.

2.10 make_excecutable()

make_excecutable() makes the executable file (`run.sh`) that is used to execute VET.

3 References

- VET Tutorial: https://wiki.ligo.org/DetChar/VetoEvaluationTool
- VET Example (O3): https://wiki.ligo.org/DetChar/VetoEvaluationToolEx