Psyllid Documentation

Release v1.12.4

Project 8 Collaboration

Mar 24, 2020

Contents

Contents:

CHAPTER 1

Installation Guide

1.1 Requirements

1.1.1 Operating Systems

Psyllid can be installed on Linux and macOS systems. Some functionality (e.g. the Fast Packet Acquisition) is only available on Linux.

1.1.2 Dependencies

- CMake 3.1 or higher
- C++11 (gcc 4.9 or higher; or clang 3 or higher)
- Boost 1.48 or higher
- HDF5
- rabbitmqc

These can all be installed from package managers (recommended) or by source.

1.1.3 Submodules

- midge
- dripline-cpp
- monarch

1.2 Basic Installation

1. Ensure the dependencies are installed.
2. Clone:

```
> git clone https://github.com/project8/psyllid.git
```

or:

```
> git clone git@github.com:project8/psyllid.git
```

3. Updated submodules:

```
> git submodule update --init --recursive
```

4. Create a build directory and run CMake:

```
> cd psyllid
> mkdir build
> cd build
> cmake ..
```

5. Build and install (will install within the build directory):

```
> make install
```

6. Test the installation

    > bin/psyllid

With the dependencies installed clone into `https://github.com/project8/psyllid.git`. Make sure to also pull the submodules by navigating to the cloned directory and running:

```
git submodule update --init --recursive
```

1.3 Commonly Used Build Options

- `CMAKE_INSTALL_PREFIX`: the default is the build directory; change to wherever you want your libraries and binaries installed
- `CMAKE_BUILD_TYPE`: the default is `DEBUG`; set to `RELEASE` for the fastest performance and less verbosity
- `Psyllid_ENABLE_ITERATOR_TIMING`: the default is `OFF`; set to `ON` to get some diagnostics on how fast the nodes are processing data
- `Psyllid_ENABLE_TESTING`: the default is `OFF`; set to `ON` to build the test programs

CHAPTER 2

Getting started

2.1 Introduction

Psyllid is a data acquisition package developed by the Project 8 collaboration and it's purpose is to receive UDP packets via a port, unpack and analyze them in real-time and write data to egg-files using Monarch.

Currently, the Project 8 experiment uses a ROACH2 board to continuously digitize and packetize a mixed down RF-signal (for more details on the packet structure and content see RoachPackets). The ROACH2 streams UDP packets at a rate of 24kHz via three 10GbE connections to a server. In order to keep up with such a high rate of incoming packets, Psyllid is written multi-threaded and the packet processing and data analysis is done in parallel by various nodes. By running psyllid with different node configurations, psyllid can be used to serve different purposes. It can for example continuously write the entire incoming data to files or examine the packet content and decide based on some pre-defined criteria whether or not to write out the content.

Psyllid's internal structure basically consists of 2 layers: the *control* layer and the *daq* layer. The classes in the *control* layer control the *daq* layer and allow a user to interact with psyllid. Dripline commands are received by the *run-server* and are forwarded to their destination node-binding. Upon request, an instance of the *daq-control* class tells Midge to deactivate or activate the daq-nodes and to start or stop a run. Psyllid knows different states: deactivated, activating, activated, running, canceled, do_restart, done and error. Data is only being taken in *running* state.

For more information read the sections How psyllid works and DAQ Status.

A note on data rates and starting runs

Psyllid will only go to *running* state if told by the user via a *on-startup* command or by sending the *start-run* dripline request. When a writer node is configured psyllid will write data to files in this state. In *streaming-mode* the data rate is about 200MB/s. So be careful with run durations. The default run-duration is 1s. If you start a run without specifying a run duration or a filepath it will result in a 200MB file being written to the directory in which you ran the psyllid executable. In all other states no data is being taken/written. So as long as you don't tell psyllid to start-running you don't need to worry about accidentally taking data without noticing. And in case of emergency: *Control-C*

2.2 Running psyllid

For installation instructions go here

Running psyllid does not mean start a psyllid run. Psyllid starts in deactivated or activated status.

2.2.1 Running requirements

- rabbit broker
- broker authentication file

If you don't have a rabbitmq-broker available you can go to broker-in-docker and run docker-compose up. A broker will be started inside a docker container and the host ports are forwarded to this container. Copy the *project8_authentications.json* file to */your-home/.project8_authentications.json*. This way psyllid can connect to it and set up a queue from anywhere on the host machine.

For adding a dripline or dragonfly container, look for examples on how to do that in insectarium.

2.2.2 Start psyllid

The psyllid executable can be found in */path_to_build/bin/*. Running the executable `/path_to_build/bin/psyllid` will result in a psyllid instance being created. Once started, psyllid will initialize its control classes, connect to the broker and start the pre-configured daq nodes. If psyllid cannot connect to a broker, it will exit.

2.2.3 Permissions

Generally, psyllid can be run without sudo permissions. However, they are required for using the *packet-receiver-fpa*. This node can shortcut the port and is therefore faster at reading the incoming packets. If you don't have sudo permissions you can only use the *presets* using the *packet-receiver-socket* node instead which cannot keep up with the ROACH2 packet rate.

2.3 Configuration files

By using the command line option *-c* and specifying the path to a configuration file in YAML-format

```
bin/psyllid config=/path-to-config/config.yaml
```

psyllid will be set up according to whatever is specified in the file and overwrite its default settings. Here is how a configuration file for running psyllid in streaming mode could look like:

```
amqp:
    broker: localhost
    queue: psyllid

post-to-slack: false

daq:
    activate-at-startup: true
    n-files: 1
    max-file-size-mb: 500
```

(continues on next page)

(continued from previous page)

```
streams:
    ch0:
        preset: str-1ch-fpa

        device:
            n-channels: 1
            bit-depth: 8
            data-type-size: 1
            sample-size: 2
            record-size: 4096
            acq-rate: 100 # MHz
            v-offset: 0.0
            v-range: 0.5

        prf:
            length: 10
            port: 23530
            interface: eth1
            n-blocks: 64
            block-size: 4194304
            frame-size: 2048

        strw:
file-num: 0
```

In this example the broker is running on localhost and a queue will be set up with the name *psyllid*. If no path to the broker-authentication file is provided, psyllid will look for *~/.project8_authentications.json*. The *post-to-slack* option, allows psyllid to post some messages (like "psyllid is starting up" or "Run is commencing") to a slack channel. Under *daq*, some general settings are configured. For example the maximum file size is set to 500 MB which means that psyllid will continue to write to a new egg file once the size of the last egg file has reached this size. In the *streams* section one stream with the name *ch0* is configured. In this stream the nodes are connected according to the preset *str-1ch-fpa*. The configurations of the individual nodes from this stream must be specified within the *stream* block. Here the *prf* (packet-receiver-fpa) is the node that receives and distributes the incoming packets. It is configured with the interface name and the port to which the ROACH2 is streaming the packets to. The *length* in node configurations always refers to a buffer size. Generally, the buffers of nodes should be larger than the buffer sizes downstream, to prevent that the data processing gets blocked if one of the nodes is falling behind.

In theory, psyllid can be run with multiple streams, each of which could be set up with a different node configuration. In practice though, psyllid is mostly used with only one stream set up and in case data should be read from multiple ports, multiple instances of psyllid are run in parallel.

2.3.1 Node connections and presets

Which nodes will be set up and how they will be connected can be specified either by using a preset as in the example above or manually in the configuration file. This is how the same node configuration as above could be achieved manually:

```
preset:
    type: reader-stream
    nodes:
      - type: packet-receiver-fpa
        name: prf
      - type: tf-roach-receiver
        name: tfrr
```

(continues on next page)

(continued from previous page)

```
      - type: streaming-writer
        name: strw
      - type: term-freq-data
        name: term
    connections:
      - "prf.out_0:tfrr.in_0"
      - "tfrr.out_0:strw.in_0"
      - "tfrr.out_1:term.in_0"
```

The available presets can be found in node configurations.

If you want to use psyllid to process ROACH2 packets and write all the content to files use the *str-1ch-fpa* preset. If you want to take triggered data use *events-1ch-fpa*.

2.3.2 on-startup

A *on-startup* block can be added to the configuration file with a list of requests that psyllid will process and act upon right after starting-up. This is especially convenient when using psyllid to read data from a file instead of from a port (see *egg-reader* for more on this). In this case psyllid will only start the the *run-server* for receiving commands from a user after executing all the on-startup commands. Here is an example for a *on-startup* command block:

```
amqp:
    broker: rabbit_broker
    queue: psyllid
    make-connection: false

on-startup:
  - type: get
    rks: daq-status
    payload: {}
    sleep-for: 0
  - type: wait-for
    rks: daq-status
    payload: {}
    sleep-for: 1000
  - type: wait-for
    rks: daq-status
    payload: {}
    sleep-for: 100
```

In this example, the psyllid instance will not try to connect to the broker and as a result it will exit after processing the *on-startup* requests.

2.4 Interacting with psyllid

As mentioned a few times above, it is possible to send dripline requests via a rabbitmq broker to a running psyllid instance. There is a detailed list of which requests can be received and processed in Psyllid API.

Assuming you have dragonfly installed, here are some examples for how to interact with psyllid from the command line:

- If you have a psyllid instance running (and it was configured to have "psyllid" as queue name), you can for example send a request to ask what state psyllid is in by running:

```
dragonfly get psyllid.daq-status -b rabbit_broker
```

- Deactivate and activate psyllid:

```
dragonfly cmd psyllid.activate-daq -b rabbit_broker

dragonfly cmd psyllid.deactivate-daq -b rabbit_broker
```

- Make psyllid exit:

```
dragonfly cmd psyllid.quit-psyllid -b rabbit_broker
```

- To start a 500ms run:

```
dragonfly cmd psyllid.start-run duration=500 filename=a_test.egg -b rabbit_broker
```

- Node configurations can also be changed by sending the relevant request. If you are running psyllid using the *event_builder_1ch_fpa* preset you can set the snr trigger level of the frequency mask trigger node to 20 with:

```
dragonfly set psyllid.active-config.ch0.fmt.threshold-power-snr 20 -b rabbit_
↪broker
```

  Check the threshold setting with:

```
dragonfly get psyllid.active-config.ch0.fmt.threshold-power-snr -b rabbit_broker
```

- Changing the buffer size of a node at run time, will only re applied after psyllid reactivates (because buffers are initialized when nodes are activated).

```
dragonfly cmd psyllid.reactivate-daq -b rabbit_broker
```

2.5 Egg reader

Instead of reading data packets that were received via a port, psyllid has the option to read the content from an egg file. By using the *egg-reader* node and configuring this node with the path to the file that it should read from, psyllid will perfrom the exact same operations as it would in normal operation on the content of a file. By defining a list of start-up commands in the configuration file, psyllid will perform all of them and then exit once the processing of the file content was completed.

Here is an example for an egg-reader configuration:

```
e3r:
    length: 1000
    egg-path: /a_test_file.egg
    read-n-records: 0
```

CHAPTER 3

How Psyllid Works

3.1 Goals and Organization

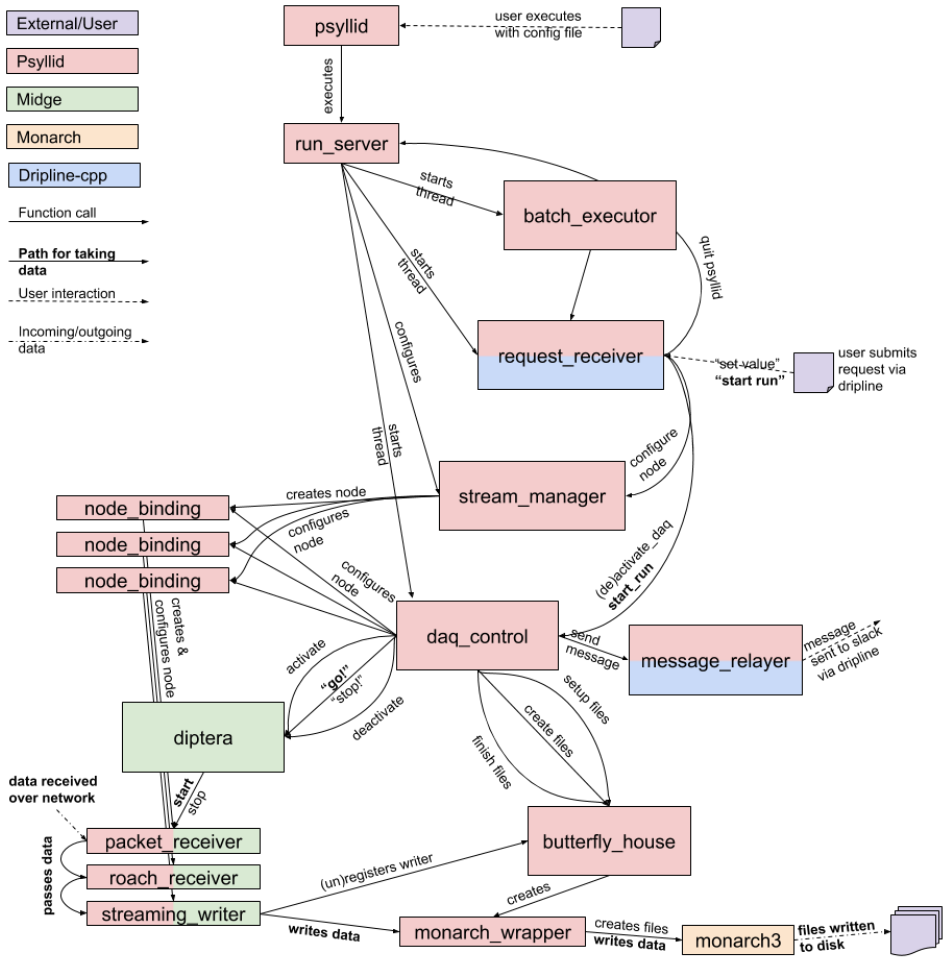
The main goals of the Psyllid architecture are to:

1. be module and configurable at runtime,
2. safely run components of the DAQ system in parallel, and
3. integrate into an existing Dripline controls system

The source directory of Psyllid includes the following subdirectories:

1. `applications` — executables
2. `control` — classes responsible for control of the DAQ system and the user interface
3. `data` — data classes
4. `daq` — classes responsible for the actual data acquisition
5. `test` — test programs
6. `utility` — utility classes

The basic setup of the classes and the interactions between them is shown here:



The main components and their responsibilities are:

- `run_server` — creates and configures the main Psyllid components, and starts their threads;
- `batch_executor` — submits pre-configured commands after the main Psyllid components have started up;
- `request_receiver` — provides the dripline interface for user requests;
- `stream_manager` — creates and configures nodes;
- `daq_control` — starts and stops runs, and peforms active configurtion operations;
- `diptera` — owns and operates the DAQ nodes;
- `message_relayer` — sends message to Slack via dripline;
- `butterfly_house` — provides centralized interactions with the Monarch3 library, and deadtime-less file creation.

3.2 Data Acquisition

The core data-acquisition capabilities are built on the Midge framework. The capabilities of the system are implemented in "nodes," each of which has a particular responsibility (e.g. one node receives packets off the network;

another node determines if the data passes a trigger; another node writes data to disk). Nodes are connected to one another by circular buffers of data objects.

3.3 User Interface

3.3.1 On Startup

The user typically starts Psyllid with a YAML configuration file. Several example configuration files can be found in the `examples` directory. Most configuration files will include these top-level blocks:

1. `amqp` — AMQP broker information;

2. `batch-actions` — commands to be submitted automatically after the components of Psyllid have started up; 2. `daq` — Information for the DAQ controller, such as how many files to prepare, maximum file size, and whether to activate the DAQ on startup; 3. `streams` – Definitions of the streams to be used.

Configuration options can also be set/modified from the command line.

Example Command Line Usage

```
> bin/psyllid config=my_config.yaml amqp.broker=my.amqp.broker
```

Explanation:

- `bin/psyllid` is the Psyllid executable
- `config=my_config.yaml` specifies `my_config.yaml` as the file that will set most of the configuration values
- `amqp.broker=my.amqp.broker` modifies this value in the configuration:

```
amqp:
  broker: my.amqp.broker
```

3.3.2 During Execution

User interaction with Psyllid during execution is performed via the dripline protocol. Psyllid uses the `dripline-cpp` library to enable that interface. The `request_receiver` class (a `dripline::hub`) ties dripline commands to particular functions in the `stream_manager`, `daq_control`, and `run_server` classes.

See the *API* page for more information about the dripline API for Psyllid.

CHAPTER 4

API

Users interact with Psyllid using with Dripline-standard requests.

The API documentation below is organized into requests that are part of the Dripline standard, and those that are Psyllid-specific. They are further divided by request type: RUN, GET, SET, and CMD.

The routing key should consist of the server queue name, followed by the Routing Key Specifier (RKS): *[queue].[RKS]*.

For each request type, the API includes the allowed RKS, Payload options, and contents of the Reply Payload.

<batch-command-key> All keys listed in the *batch-commands* node of the configuration are bound as command names and may be called. These add one or more commands to the batch_executor queue; the return code indicates only that the actions were queued, nothing about their execution status. The command `hard-abort` is defined in the default server_config to execute `stop-run`.

4.1 Dripline Requests

4.1.1 OP_GET

`is-locked`

Returns whether or not the server is locked out.

Reply Payload

- `is-locked=[true/false (bool)]`

4.1.2 OP_CMD

`lock`

Requests that the server lockout be enabled. Nothing is done if already locked.

Reply Payload

- `key=[UUID (string)]` – lockout key required for any lockable requests as long as the lock remains enabled
- `tag=[lockout tag (object)]` – information about the client that requested that the lock be enabled

`unlock`

Requests that the server lockout be disabled.

Payload

- `force=[true (bool)]` – *(optional)* Disables the lockout without a key.

`ping`

Check that the server receives requests and sends replies. No other action is taken.

`set_condition`

Cause the server to move to some defined state as quickly as possible; Psyllid implements conditions 10 and 12, both of which stop any ongoing run. _Note:_ set_condition is expected to be a broadcast command (routing key target is *broadcast* not <*psyllid-queue*>).

Payload

- `values=[condition (int)]` – the integer condition

4.2 Psyllid Requests

4.2.1 OP_RUN

The *run* message type is used to start a run.

All *run* requests are lockable.

Details

There are no Routing Key Specifiers for *run* requests.

Payload

- `filename: [filename (string)]` – *(optional)* Filename for `file_number` 0.
- `filenames: [array of filenames (string)]` – *(optional)* Filenames for all files specified. Overrides `filename`.
- `description: [description (string)]` – *(optional)* Text description for `file_number` 0; saved in the file header.
- `descriptions: [array of descriptions (string)]` – *(optional)* Text descriptions for all files specified. Overrides `description`.
- `duration: [ms (unsigned int)]` – *(optional)* Duration of the run in ms.

4.2.2 OP_GET

The *get* message is used to request information from the server.

No *get* requests are lockable.

`daq-status`

Returns the current acquisition configuration.

Reply Payload

- `status: [status (string)]` – human-readable status message
- `status-value: [status code (unsigned int)]` – machine-redable status message

`node-config.[stream].[node]`

Returns the configuration of the node requested.

Reply Payload

- `[Full node configuration]`

`node-config.[stream].[node].[parameter]`

Returns the configuration value requested from the node requested.

Reply Payload

- `[parameter name]: [value]` – Parameter name and value

`active-config.[stream].[node]`

Returns the configuration of the active DAQ node requested.

Reply Payload

- `[Full node configuration]`

`active-config.[stream].[node].[parameter]`

Returns the configuration value requested from the active DAQ node requested. Please note that this action will not necessarily return the value in use (e.g. if a parameter that is only used once during initialization has been changed since then), and is not necessarily thread-safe.

Reply Payload

- `[parameter name]: [value]` – Parameter name and value

`stream-list`

Returns a list of all streams in the psyllid instance

Reply Payload

- `streams: [[stream_name (string)]]` – array of names of the streams

`node-list.[stream]`

Returns a list of all the nodes in the indicated stream

Reply Payload

- `nodes: [[node_name (string)]]` – array of names of the nodes

`filename.[file_number (optional)]`

Returns the filename that will be written to by writters registered to `file_number`. Default for `file_number` is 0.

Reply Payload

- `values: [[filename (string)]]` – Filename as the first element of the `values` array

`description.[file_number (optional)]`

Returns the description that will be written to the file header for file corresponding to `file_number`. Default for `file_number` is 0.

Reply Payload

- `values: [[description (string)]]` – Description as the first element of the `values` array

`duration`

Returns the run duration (in ms).

Reply Payload

- `values: [[duration (unsigned int)]]` – Duration in ms as the first element of the `values` array

`use-monarch`

Returns the use-monarch flag.

Reply Payload

- `values: [[flag (bool)]]` – Use-monarch flag as the first element of the `values` array

4.2.3 OP_SET

The *set* message type is used to set a value to a parameter in the configuration.

All *set* requests are lockable.

`node-config.[stream].[node]`

Configures one or more parameters within a node. Takes effect next time the DAQ is activated.

Payload

- `[node configuration (dictionary)]` – Parameters to set in the node

Reply Payload

- `[the parameters that were set (dictionary)]` – Parameter name:value pairs that were set

`node-config.[stream].[node].[parameter]`

Configure a single parameter in a node. Takes effect next time the DAQ is activated.

Payload

- `values: [[value]]` – Parameter value to be set as the first element of the `values` array.

`active-config.[stream].[node]`

Configures one or more parameters within an active DAQ node. Takes effect immediately.

Payload

- `[node configuration (dictionary)]` – Parameters to set in the node

Reply Payload

- `[the parameters that were set (dictionary)]` – Parameter name:value pairs that were set

`active-config.[stream].[node].[parameter]`

Configure a single parameter in an active DAQ node. Takes effect immediately. Please note that this action will not necessarily be useful for all node parameters (e.g. if a parameter is used once during initialization), and is not necessarily thread-safe.

Payload

- `values: [[value]]` – Parameter value to be set as the first element of the `values` array.

`description.[file_number (optional)]`

Sets the description that will be written to the file header for the file corresponding to `file_number`. Default for `file_number` is 0. Takes effect for the next run.

Payload

- `values: [[description (string)]]` – Description

Reply Payload

- `[the parameter that was set as a dictionary]` – Parameter name:value pair that was set

`duration`

Sets the run duration in ms. Takes effect for the next run.

Payload

- `values: [[duration (unsigned int)]]` – Duration in ms

`use-monarch`

Sets the use-monarch flag. Takes effect for the next run.

Payload

- `values: [[flag (bool)]]` – Flag value (true, false, 0, 1)

4.2.4 OP_CMD

The *cmd* message type is used to run a variety of different command instructions.

All *command* requests are lockable.

`add-stream`

Adds a stream to the DAQ configuration. Takes effect next time the DAQ is activated.

Payload

- `name: [stream name (string)]` – Unique name for the stream.
- `config: [stream configuration (dictionary)]` – Configuration for the stream

`remove-stream`

Remove a stream from the DAQ configuration. Takes effect next time the DAQ is activated.

Payload

- `values: [[stream name (string)]]` – Name of the stream to remove as the first element of the `values` array

`run-daq-cmd.[stream].[node].[cmd]`

Instruct an active DAQ node to execute a particular command. Please note that this action is not necessarily thread-safe.

Payload

- `[command arguments (dictionary)]` – Any arguments needed for the execution of the command.

Reply Payload

- `[the command configuration given to the node (dictionary)]` – Repeating what the node was told to do

`stop-run`

Stop a run that's currently going on.

`start-run`

Same as the OP_RUN command above.

`activate-daq`

Put the DAQ in its actiavated state to be ready to take data. Psyllid must be in its deactivated state before this call.

`reactivate-daq`

Deactivate, then reactivate the DAQ; it will end in its activated state, ready to take data. Psyllid must be in its activated state before this call.

`deactivate-daq`

Put in its deactivated state, in which it is not immediately ready to take data. Psyllid must be in its activated state before this call.

`quit-psyllid`

Instruct the Psyllid executable to exit.

CHAPTER 5

Node Configurations

5.1 Nodes

5.1.1 Producers

packet_receiver_fpa

A producer to receive UDP packets via the fast-packet-acquisition interface and write them as raw blocks of memory. Works in Linux only. Parameter setting is not thread-safe. Executing is thread-safe.

- Type: `packet-receiver-fpa`
- Configuration
  - "length": uint – The size of the output buffer
  - "max-packet-size": uint – Maximum number of bytes to be read for each packet; larger packets will be truncated
  - "port": uint – UDP port to listen on for packets
  - "interface": string – Name of the network interface to listen on for packets
  - "timeout-sec": uint – Timeout (in seconds) while listening for incoming packets; listening for packets repeats after timeout
  - "n-blocks": uint – Number of blocks in the mmap ring buffer
  - "block-size": uint – Number of packets per block in the mmap ring buffer
  - "frame-size": uint – Number of blocks per frame in the mmap ring buffer
- Output
  - 0: `memory_block`

packet_receiver_socket

A producer to receive UDP packets via the standard socket interface and write them as raw blocks of memory. Parameter setting is not thread-safe. Executing is thread-safe.

- Type: `packet-receiver-socket`
- Configuration
  - "length": uint – The size of the output buffer
  - "max-packet-size": uint – Maximum number of bytes to be read for each packet; larger packets will be truncated
  - "port": uint – UDP port to listen on for packets
  - "ip": string – IP port to listen on for packets; must be in IPV4 numbers-and-dots notation (e.g. 127.0.0.1)
  - "timeout-sec": uint – Timeout (in seconds) while listening for incoming packets; listening for packets repeats after timeout
- Output
  - 0: `memory_block`
- `udp_receiver` (`udp-receiver`)
  - Output 0: `time_data`
  - Output 1: `freq_data`

egg3_reader

Egg file reader based on the monarch3 library

- Type: `egg3-reader`
- Configuration
  - "egg_path": string – resolvable path tot he egg file from which to read time series data
- Output * 0: `time_data`

5.1.2 Transformers

event_builder

Keeps track of the state of the packet sequence (is-triggered, or not).

When going from *untriggered* to *triggered*, adds some number of pretrigger packets. Includes a skip tolerance for untriggered packets between two triggered packets.

In untriggered state, the *flag* and the *high_threshold* variables of the trigger flags are checked. Only if both are true the event builder switches state. If *f_n_triggers* > *1* the next state is collecting_triggers, otherwise triggered. In collecting_triggers state the event builder counts the incomming trigger flags with *flag* = *true*. Only if this *count* == *f_n_triggers* the state is switched to triggered.

Events are built by switching some untriggered packets to triggered packets according to the pretrigger and skip-tolerance parameters. Continuous sequences of triggered packets constitute events.

Parameter setting is not thread-safe. Executing is thread-safe.

The cofigurable value *time-lengt* in the `tf_roach_receiver` must be set to a value greater than *pretrigger* and *skip-tolerance* (+5 is advised). Otherwise the time domain buffer gets filled and blocks further packet processing.

- Type: `event-builder`
- Configuration
  - "length": uint – The size of the output buffer
  - "pretrigger": uint – Number of packets to include in the event before the first triggered packet
  - "skip-tolerance": uint – Number of untriggered packets to include in the event between two triggered packets
  - "n-triggers": uint – Number of trigger flags with flag == true required before switching to triggered state
- Input
  - 0: `trigger_flag`
- Output
  - 0: `trigger_flag`

`frequency_mask_trigger`

The FMT has two modes of operation: updating the mask, and triggering.

When switched to the "updating" mode, the existing mask is erased, and the subsequent spectra that are passed to the FMT are used to calculate the new mask. The number of spectra used for the mask is configurable. Those spectra are summed together as they arrive. Once the appropriate number of spectra have been used, the average value is calculated, and the mask is multiplied by the threshold SNR (assuming the data are amplitude values, not power).

In triggering mode, each arriving spectrum is compared to the mask bin-by-bin. If a bin crosses the threshold, the spectrum passes the trigger and the bin-by-bin comparison is stopped.

It is possible to set a second threshold (*threshold-power-snr-high*). In this case a second mask is calculated for this threshold and the incoming spectra are compared to both masks. The output trigger flag has an additional variable *high_threshold* which is set true if the higher threshold led to a trigger. This variable is always set to true if only one trigger level is used.

The mask can be written to a JSON file via the *write_mask()* function. The format for the file is:

{ "timestamp": "[timestamp]", "n-packets": [number of packets averaged], "mask": [value_0, value_1, ...] }

Parameter setting is not thread-safe. Executing (including switching modes) is thread-safe.

- Type: `frequency-mask-trigger`
- Configuration
  - "length": uint – The size of the output data buffer
  - "n-packets-for-mask": uint – The number of spectra used to calculate the trigger mask
  - "threshold-ampl-snr": float – The threshold SNR, given as an amplitude SNR
  - "threshold-power-snr": float – The threshold SNR, given as a power SNR
  - "threshold-power-snr-high": float – A second SNR threshold, given as power SNR
  - "threshold-dB": float – The threshold SNR, given as a dB factor
  - "trigger-mode": string – The trigger mode, can be set to "single-level-trigger" or "two-level-trigger"

  - "n-spline-points": uint – The number of points to have in the spline fit for the trigger mask
- Available DAQ commands
  - "update-mask" (no args) – Switch the execution mode to updating the trigger mask
  - "apply-trigger" (no args) – Switch the execution mode to applying the trigger
  - "write-mask" ("filename" string) – Write the mask in JSON format to the given file
- Input
  - 0: `freq_data`
- Output
  - 0: `trigger_flag`

`frequency_transform`

Compute fourier transform of time dataa

- Type: `frequency-transform`
- Configuration
  - "time-length": uint – The size of the output time-data buffer
  - "freq-length": uint – The size of the output frequency-data buffer
  - "fft-size": unsigned – The length of the fft input/output array (each element is 2-component)
  - "start-paused": bool – Whether to start execution paused and wait for an unpause command
  - "transform-flag": string – FFTW flag to indicate how much optimization of the fftw_plan is desired
  - "use-wisdom": bool – whether to use a plan from a wisdom file and save the plan to that file
  - "wisdom-filename": string – if "use-wisdom" is true, resolvable path to the wisdom file
- Input
  - 0: `time_data`
- Output
  - 0: `time_data`
  - 1: `freq_data`

`tf_roach_receiver`

Splits raw combined time-frequency stream into time and frequency streams. Parameter setting is not thread-safe. Executing is thread-safe.

- Type: `tf-roach-receiver`
- Configuration
  - "time-length": uint – The size of the output time-data buffer
  - "freq-length": uint – The size of the output frequency-data buffer
  - "udp-buffer-size": uint – The number of bytes in the UDP memory buffer for a single packet; generally this shouldn't be changed and is specified by the ROACH configuration

  - "time-sync-tol": uint – (currently unused) Tolerance for time synchronization between the ROACH and the server (seconds)
  - "start-paused": bool – Whether to start execution paused and wait for an unpause command
  - "force-time-first": bool – If true, when starting ignore f packets until the first t packet is received
- Input
  - 0: `memory_block`
- Output
  - 0: `time_data`
  - 1: `freq_data`

5.1.3 Consumers

`triggered_writer`

Writes triggered data to an egg file. Parameter setting is not thread-safe. Executing is thread-safe.

- Type: `triggered-writer`
- Configuration
  - "file-size-limit-mb": uint – Not used currently
  - "device": node – digitizer parameters
    - "bit-depth": uint – bit depth of each sample
    - "data-type-size": uint – number of bytes in each sample (or component of a sample for sample-size > 1)
    - "sample-size": uint – number of components in each sample (1 for real sampling; 2 for IQ sampling)
    - "record-size": uint – number of samples in each record
    - "acq-rate": uint – acquisition rate in MHz
    - "v-offset": double – voltage offset for ADC calibration
    - "v-range": double – voltage range for ADC calibration
  - "center-freq": double – the center frequency of the data being digitized
  - "freq-range": double – the frequency window (bandwidth) of the data being digitized
- Input
  - 0: `time_data`
  - 1: `trigger_flag`

`roach_freq_monitor`

Checks for missing frequency packets

- Type: `roach-freq-monitor`
- Configuration (none)

- Input
  - 0: `freq_data`

roach_time_monitor

Checks for missing time packets

- Type: `roach-time-monitor`
- Configuration (none)
- Input
  - 0: `time_data`

streaming_writer

Writes streamed data to an egg file. Parameter setting is not thread-safe. Executing is thread-safe.

- Type: `streaming-writer`
- Configuration
  - "file-size-limit-mb": uint – Not used currently
  - "device": node – digitizer parameters
    - "bit-depth": uint – bit depth of each sample
    - "data-type-size": uint – number of bytes in each sample (or component of a sample for sample-size > 1)
    - "sample-size": uint – number of components in each sample (1 for real sampling; 2 for IQ sampling)
    - "record-size": uint – number of samples in each record
    - "acq-rate": uint – acquisition rate in MHz
    - "v-offset": double – voltage offset for ADC calibration
    - "v-range": double – voltage range for ADC calibration
  - "center-freq": double – the center frequency of the data being digitized
  - "freq-range": double – the frequency window (bandwidth) of the data being digitized
- Input
  - 0: `time_data`

streaming_frequency_writer

Writes streamed frequency data to an egg file

- Type: `streaming-frequency-writer`
- Configuration
  - "file-size-limit-mb": uint – Not used currently
  - "device": node – digitizer parameters
    - "bit-depth": uint – bit depth of each sample

        - "data-type-size": uint – number of bytes in each sample (or component of a sample for sample-size > 1)
        - "sample-size": uint – number of components in each sample (1 for real sampling; 2 for IQ sampling)
        - "record-size": uint – number of samples in each record
        - "acq-rate": uint – acquisition rate in MHz
        - "v-offset": double – voltage offset for ADC calibration
        - "v-range": double – voltage range for ADC calibration
    - "center-freq": double – the center frequency of the data being digitized
    - "freq-range": double – the frequency window (bandwidth) of the data being digitized
- Input
    - 0: `freq_data`

`terminator_freq`

Does nothing with frequency data

- Type: `terminator-freq`
- Configuration (none)
- Input
    - 0: `freq_data`

`terminator_time`

Does nothing with time data

- Type: `terminator-time`
- Configuration (none)
- Input
    - 0: `time_data`

5.2 Stream Presets

- `streaming_1ch` (`str-1ch`)
    - Nodes
        - `packet-receiver-socket` (`prs`)
        - `tf-roach-receiver` (`tfrr`)
        - `streaming-writer` (`strw`)
        - `term-freq-data` (`term`)
    - Connections

        * prs.out_0:tfrr.in_0
        * tfrr.out_0:strw.in_0
        * tfrr.out_1:term.in_0
- streaming_1ch_fpa (str-1ch-fpa)
    - Nodes
        * packet-receiver-socket (prf)
        * tf-roach-receiver (tfrr)
        * streaming-writer (strw)
        * term-freq-data (term)
    - Connections
        * prf.out_0:tfrr.in_0
        * tfrr.out_0:strw.in_0
        * tfrr.out_1:term.in_0
- fmask_trigger_1ch (fmask-1ch)
    - Nodes
        * packet-receiver-socket (prs)
        * tf-roach-receiver (tfrr)
        * frequency-mask-trigger (fmt)
        * triggered-writer (trw)
    - Connections
        * prs.out_0:tfrr.in_0
        * tfrr.out_0:trw.in_0
        * tfrr.out_1:fmt.in_0
        * fmt.out_0:trw.in_1
- fmask_trigger_1ch_fpa (fmask-1ch-fpa)
    - Nodes
        * packet-receiver-fpa (prf)
        * tf-roach-receiver (tfrr)
        * frequency-mask-trigger (fmt)
        * triggered-writer (trw)
    - Connections
        * prf.out_0:tfrr.in_0
        * tfrr.out_0:trw.in_0
        * tfrr.out_1:fmt.in_0
        * fmt.out_0:trw.in_1
- event_builder_1ch (events-1ch)

  - Nodes
    * packet-receiver-socket (prs)
    * tf-roach-receiver (tfrr)
    * frequency-mask-trigger (fmt)
    * event-builder (eb)
    * triggered-writer (trw)
  - Connections
    * prs.out_0:tfrr.in_0
    * tfrr.out_0:trw.in_0
    * tfrr.out_1:fmt.in_0
    * fmt.out_0:eb.in_0
    * eb.out_0:trw.in_1
- event_builder_1ch_fpa (events-1ch-fpa)
  - Nodes
    * packet-receiver-fpa (prf)
    * tf-roach-receiver (tfrr)
    * frequency-mask-trigger (fmt)
    * event-builder (eb)
    * triggered-writer (trw)
  - Connections
    * prf.out_0:tfrr.in_0
    * tfrr.out_0:trw.in_0
    * tfrr.out_1:fmt.in_0
    * fmt.out_0:eb.in_0
    * eb.out_0:trw.in_1

CHAPTER 6

DAQ Status

The status can be queried by doing a GET of `daq-status`. The payload will include the status name as a string with key `status`, and the value with key `status-value`.

Status	Value	Description
Deactivated	0	Psyllid is in the idle state and needs to be activated to run
Activating	2	DAQ spinning up
Activated	4	DAQ is ready to start a run
Running	5	A run is ongoing
Deactivating	6	DAQ is spinning down
Canceled	8	Psyllid was canceled and is probably shutting down
Do Restart	9	A non-fatal error occurred; reactivate Psyllid to run again
Done	10	Psyllid operations are complete; restart Psyllid to run again
Error	200	An error occurred; restart Psyllid to run again

CHAPTER 7

Roach Packets

The ROACH2 for Project 8 sends UDP packets over a 10 GBe connection.

As configured for Phase 2 of Project 8, time-domain and frequency-domain packets alternate for each channel.

These notes are based on an email from André to Noah (1/21/16).

7.1 Structure

7.1.1 Definition

```
#define PAYLOAD_SIZE 8192 // 1KB
#define PACKET_SIZE sizeof(packet_t)
typedef struct packet {
      // first 64bit word
      uint32_t unix_time;
      uint32_t pkt_in_batch:20;
      uint32_t digital_id:6;
      uint32_t if_id:6;
      // second 64bit word
      uint32_t user_data_1;
      uint32_t user_data_0;
      // third 64bit word
      uint64_t reserved_0;
      // fourth 64bit word
      uint64_t reserved_1:63;
      uint64_t freq_not_time:1;
      // payload
      char data[PAYLOAD_SIZE];
} packet_t;
```

7.1.2 Fields

unix_time The usual unix time meaning (seconds since 1 Jan 1970). At start-up the computer configures the "zero time" unix time, and from there the roach just keeps an internal counter.

pkt_in_batch This is just a counter that increments with each pair of time/frequency packets sent out and wraps every 16 seconds (that is the shortest time that is both an integer number of seconds and integer number of packets). It counts to 390625 before wrapping to zero (390626 * (1 time + 1 freq packet) = 781252 packets in total every 16 seconds). A frequency and time packet that have the same counter value contains information on the same underlying data.

digital_id This identifies the digital channel (one of three). In the bitcode and control software I call the channels 'a', 'b', and 'c', which will map to digital_id 0, 1, and 3, respectively.

if_id Identifies the IF input on the roach, either 0 or 1, and corresponds to a label on the front panel. Currently only if0 used.

user_data_1 This is a software configurable register within the roach reserved for whatever use you can think of. It is simply copied into the header of each packet.

user_data_0 Ditto user_data_1

reserved_0 This is a hard-coded 64-bit header inside the bitcode. Can also be put to use, or perhaps made software configurable.

reserved_1 Ditto reserved_0, except it is only 63-bit.

freq_not_time If 1, then the packet contains frequency-domain data, if 0 it contains time-domain data.

7.1.3 Size

32 bytes header + 8192 bytes payload = 8224 bytes in total.

Note: network interfaces should be setup to handle MTU of this size - for high-speed network interfaces the default is usually much smaller. In any case, I've done this for the computer we acquired to go with the roach.)

7.1.4 Byte Reordering

You may need to do some byte-reordering depending on the machine you're working with. Each 64-bit word inside the packet is sent as big-endian, whereas the machine attached to the roach uses little-endian. So I had to do this (quick-and-dirty fix that I used for testing, there may be more elegant solutions):

```
typedef struct raw_packet {
  uint64_t _h0;
  uint64_t _h1;
  uint64_t _h2;
  uint64_t _h3;
  ...
} raw_packet_t;

packet_t *ntoh_packet(raw_packet_t *pkt) {
  pkt->_h0 = be64toh(pkt->_h0);
  pkt->_h1 = be64toh(pkt->_h1);
  pkt->_h2 = be64toh(pkt->_h2);
  pkt->_h3 = be64toh(pkt->_h3);
  ...
  return (packet_t *)pkt;
}
```

7.2 Interpreting the Data

There are two kinds of packets: time-domain and frequency-domain.

7.2.1 Frequency Domain

There are 4096 x (8-bit real, 8-bit imaginary) spectrum samples within a packet.

Since the input signal is real-valued, the spectrum is symmetric and only the positive half of the spectrum is provided. So each packet contains the full spectral-information for each 8192-point FFT window. (There is a technical detail here, that is by the time the signal gets to the FFT it is already IQ-sampled and contains only information in the positive half-spectrum, but that doesn't affect anything.)

The samples are in canonical order, so the first sample is DC and the last sample is `(100 MHz - channel width)`.

Sample rate at input to FFT is 200 Msps.

7.2.2 Time Domain

There are 4096 x (8-bit real, 8-bit imaginary) time-domain samples within a packet.

The time-domain data is IQ data of the positive half-spectrum of the signal going into the FFT, sampled at 100 Msps. In other words, the 200 Msps signal going into the FFT is tapped off, filtered and downconverted to shift information from 0 MHz to +100 MHz down to -50 MHz to +50 MHz, and then complex-sampled at 100 Msps. The samples are in correct order, so first sample first and last sample last. If you grab a time-domain and frequency-domain packet that have the same serial number (same `unix_time` and same `pkt_in_batch`), then you should essentially get the same spectrum. (This is not strictly true, the time-domain data passes through a second filter + downconversion, and then there's quantization in the roach FFT, but to first order this is true.)

CHAPTER 8

Validation Log

8.1 Guidelines

- All new features incorporated into a tagged release should have their validation documented. * Document the new feature. * Perform tests to validate the new feature. * If the feature is slated for incorporation into official DAQ processing, perform tests to show that the official configuration works and benefits from this feature. * Indicate in this log where to find documentation of the new feature. * Indicate in this log what tests were performed, and where to find a writeup of the results.
- Fixes to existing features should also be validated. * Perform tests to show that the fix solves the problem that had been indicated. * Perform tests to shwo that the fix does not cause other problems. * Indicate in this log what tests were performed and how you know the problem was fixed.

8.2 Template

8.2.1 Version:

Release Date:

New Features:

- **Feature 1**
  - Details
- **Feature 2**
  - Details

Fixes:

- **Fix 1**
  - Details
- **Fix 2**
  - Details

8.3 Log

8.3.1 Version: 1.12.3

Release Date: October 31, 2019

Fixes:

- **travis build configuration now triggers on branches ending with "/build"**
  - this convention is needed to be able to test CI without actually creating a merge or pushing to a main branch
- **build details modified for more natural use in debian and consistency**
  - travis builds on debian now install to /usr/local
  - setup.sh file is populated outside of the conditional blocks for each os (for lines that are common)
  - setup.sh on debian now has sufficient variables (though they are also not needed per the new location above)

8.3.2 Version: 1.12.2

Release Date: August 27, 2019

Fixes:

- Updated Docker base image to p8compute_dependencies v0.9.0.

8.3.3 Version: 1.12.1

Release Date: July 25, 2019

Fixes:

- Version setting in .travis.yml

8.3.4 Version: 1.12.0

Release Date: July 25, 2019

New Features:

- Unified Docker build, with automation through Travis CI

8.3.5 Version: 1.11.1

Release Date: July 21, 2019

Fixes:

- Updated Dripline to v1.11.0
- If butterfly house is being canceled, but streams do not cancel, then does a global cancel.

8.3.6 Version: 1.11.0

Release Date: June 28, 2019

New Features:

- Added the data-producer node

Fixes:

- Updated Dripline to v1.10.1
- Updated Midge to v3.7.3
- Define Psyllid-specific dripline return codes in Psyllid

8.3.7 Version: 1.10.3

Release Date: May 30, 2019

Fixes:

- Handle error conditions while recording data

8.3.8 Version: 1.10.2

Release Date: May 26, 2019

Fixes:

- Fixed return codes in error conditions

8.3.9 Version: 1.10.1

Release Date: May 23, 2019

Fixes:

- Fixed missing payloads.
- Dripline-cpp updated to v1.9.2

8.3.10 Version: 1.10.0

Release Date: May 22, 2019

New Features:

- **All submodules updated**
  - Dripline-cpp v1.9.1 (now under driplineorg)
  - Midge v3.7.1
  - Monarch v3.5.8
  - Scarab v2.4.7
- **New CL syntax**
  - Now using the new Scarab CLI framework
  - Standard CL argument format
- Using the updated Dripline-cpp interface
- Using the new Midge template metaprogramming

Fixes:

- Fixed infinite loop for the startup corner case where *activate-on-startup* is used but no streams are defined.

8.3.11 Version: 1.9.3

Release Date: February 17, 2019

Fixes:

- Fixed missing ampersand in the FMT binding

8.3.12 Version: 1.9.2

Release Date: January 10, 2019

Fixes:

- Fixed setting of the run description via dripline

8.3.13 Version: 1.8.3

Release Date: August 27, 2018

Fixes:

- **Midge update to v3.6.3**
  - Missing include fixed
  - Validation: Build now works on Ubuntu system where it failed before

8.3.14 Version 1.8.0

Release Date: July 27, 2018

New Features:

- **ids in skip_buffer are written as true when event_builder switches from skipping to untriggered**
  - as before, if the capacity of the skip_buffer is greater than the capacity of the pretrigger_buffer only ids that don't fit into pretrigger_buffer are written out as true
  - if the capacity of the skip_buffer is smaller than the capacity of the pretrigger_buffer all ids in the skip_buffer are written out as true
  - tested by running psyllid with the egg3-reader and checking the logging output. No crash occured and the looging output showed that the correct number of ids were written.
- **implementing support for both set_condition and batch actions:**
  - server_config now defines condition 10 and 12, both call the cmd 'hard-abort'
  - server_config now defines a top-level node 'batch-commands' with an entry for 'hard-abort' which calls 'stop-run'
  - request_receiver stores the above map (configurable in config file as top-level node 'set-conditions'); responds to set-condition commands by calling the mapped rks as an OP_CMD with empty message body * this had a bug which is now fixed, it checked for the new name but populated by the old one
  - batch_executor stores the batch-commands map (each entry in the node is an array of commands following the same syntax as those run when the system starts
  - batch_executor's constructor binds request-receiver commands for each key in the above map to do_batch_cmd_request, which adds the configured array of actions to the batch queue. This is called as *agent cmd <queue>.<key>*.
  - batch_executor's execute() method now has an infinite loop option which always tries to empty a concurrent_queue of actions (there are now utility methods plus the above which can populate that queue.
  - run_server's thread execution logic changed to account for the above changes to batch_executor's execute()
  - the 'batch-actions' top-level node name is changed to 'on-startup' to be more clear
  - tested by running psyllid in insectarium and confirming execution of stop run both on *cmd broadcast.set_condition 0* and *cmd psyllid_queue.hard-abort*.
- **updating scarab dependency to version v2.1.1**

  - tested by running psyllid in insectarium in batch mode

- **adding condition_variable notice from daq_control to indicate to request_receiver and batch_executor when the nodes are**

  - tested by having batch executor use on-start commands that need to talk to nodes (this previously resulted in crashing)

Fixes:

- corrected compiler warnings related to use of '%u' vs '%lu' for long unsigned ints in testing
- modified tk_spline (external) spline::set_boundary to be inline (it was triggering gcc warnings because it is unused)

8.3.15 Version 1.7.1:

Release Date: July 11, 2018

New Features

None

Fixes

- **Modified the Frequency Transform node to re-order FFTW output into ascending frequency order (should match Roach**

  - Tested by making psyllid record and write a frequency mask from frequency data that it produced by reading and fourier-transforming the time series from an egg file. The content of the array is now ordered correctly. This was verified by comparing the mask to the gain variation calculated by Katydid.

8.3.16 Version 1.7.0:

Release Date: June 27, 2018

New Features:

- **stream_manager methods for OP_GET of stream and node lists**
  - methods added to stream_manager, with extra get bindings in run_server
  - **tested by getting each from a running psyllid instance in insectarium and confirming:**
    * get stream-list: returns streams
    * get node-list: returns error (need to specify a stream)
    * get node-list.ch0: returns nodes

8.3.17 Version: 1.6.0

Release Date: May 25, 2018

New Features:

- midge updated to v3.5.4 (updates scarab to v1.6.1)
- **server_config now only sets the default authentication file path after checking that the path exists**
  - tested via docker batch execution with and without the auth file present; detection and setting appears to work fine
- **frequency mask trigger**
  - **updated to allow the mask and summed power arrays to be configured, either directly in the configuration file, o**
    - tested in file value arrays by setting in a file and calling write mask to ensure the values are in the output file
    - tested from-file by modifying the above output file (so that the values differ), configuring with it as input, and the writing a new output to compare
  - **added support for specifying thresholds to be measured in units of sigma of the noise, in addition to power (in dI**
    - building a mask now must accumulate variance data as well as power data
    - tested by checking sigma mask matches data-mean + sigma_threshold * sqrt(data-variance)
    - mask file contains data-mean, data-variance, mask and mask2 if present
  - **in two-level trigger-mode a second mask is created and stored; two masks can also be read in**
    - mask sizes are compared after reading
    - tested via batch mode that fmt throws error and psyllid deactivates after reading in a mask from a file if sizes mismatch
    - mask sizes are compared to incoming data array when run is started
    - tested via batch mode that a missing mask or mismatching mask sizes results in an error when run is started; psyllid exits
- **egg3-reader: support for "repeat-egg" boolean configuration option, if true, restarts reading the file from the first record**
  - tested via batch mode, using two sequential start-run commands with duration set to 0 and the egg reader configured to read 100000 records (file has ~120k records). The second run repeated the egg file (debug prints showed it re-reading earlier record IDs) and prints of the output pkt_id showed that they continued to increase as expected.
- **batch_executor: check return code of each action and exit if >= 100 (ie if an error occurred)**
  - tested with valid config file and one with a syntax error to cause error, both behave as expected (ie the latter causes a crash).

8.3.18 Version: 1.5.0

Release Date: May 8, 2018

New Features:

- batch_executor receives the reply message's payload and return code; each action happens after the prior one returns (which may not be the conclusion of the action, just like any dripline request)
- **frequency mask trigger**
  - **updated to also output the summed power data in addition to the spline fit used to define the frequency mask. Th**
    - tested using the egg reader and confirming qualitatively that the mask follows the shape of the accumulated power (after normalizing by the number of accumulated points and the mask's offset)
- Dripline-cpp updated to v1.6.0
- CMake option added to allow disabling the FPA on linux builds (useful for batch mode execution without root access).
- midge updated to v3.5.3 (updates scarab to v1.6.0)
- **server_config now only sets the default authentication file path after checking that the path exists**
  - tested via docker batch execution with and without the auth file present; detection and setting appears to work fine

8.3.19 Version: 1.4.0

Release Date: April 23, 2018

New Features:

- **Egg reader**
  - producer node which reads an existing egg file and produces a stream of time_data
  - is a flow controlling node (ie should start paused, is started by dripline commands)
  - intended use case is for reading previously streamed data and testing different trigger configurations
  - has been tested by reading an egg file and producing output files of reasonable size; content of output has not yet been validated
  - validation by using in conjunction with streaming writer and M3Info; printed record content from input file match output file.
  - documentation in doxygen output and node_configuration.rst
- **Frequency transform**
  - transform node which accepts a time_data stream and produces the same time_data stream and a corresponding freq_data stream
  - intention is that the frequency data match what would be in a ROACH2 frequency packet (as opposed to being the "best possible" FFT of the data, though hopefully those are similar)
  - supports a frequency-only output mode (for building a frequency mask)
  - has been tested only to show that both output streams can be passed to downstream nodes, content validity has not be tested
  - tested by qualitatively looking at a plot of the frequency magnitudes of frequency output file, and also the fft of the original input time data, they looked very similar (up to a normalization factor)

  - documentation in doxygen output and node_configuration.rst

- **Streaming frequency writer**
  - consumer node which is a direct copy of the streaming_writer node, with time_data replaced with freq_data (ie, it abuses the egg format and puts frequency data into what should be a time record)
  - intended for use only in testing nodes (see above), if a useful feature, the egg format needs to be extended to support it properly and this node modified correspondingly
  - documentation in doxygen output and node_configuration.rst
  - tested as part of the Frequency transoform test above
- **tf_roach_receiver optionally always starts on a t packet**
  - prior behavior was to start with the next packet received when unpaused; this feature adds a config option which will discard frequency data until the first time data is received (thus ensuring, in principle, that the output is always a matched pair)
  - documentation in doxygen output and node_configuration.rst
- **batch_executor control class**
  - allows a list of actions to be provided within the master configuration, which specifies a sequence of actions to execute at startup
  - control system modified to allow batch-only mode if the amqp configuration has *make-connection: false*, which will exit after completing batch commands
  - NOTE: currently does not do anything other than print return codes from commands; would be nice to upgrade to check those codes and crash if a command fails
  - tested using a configuration file which configures and uses a frequency mask trigger and event builder
- Dripline-cpp updated to v1.5.0

8.3.20 Version: 1.3.1

Release Date: January 30, 2018

Fixes:

- Documentation system update

8.3.21 Version: 1.3.0

Release Date: January 11, 2018

New Features:

- **Option to use monarch or not in daq_control**
  - Includes dripline get and set functions under the RKS *use-monarch*.
  - API documentation has been updated.
  - If the option is *false* and during a run a writer attempts to write to a Monarch file, Psyllid will crash.
  - Validated by demonstrating that no file is written if the option is *false* (no incoming data; standard streaming 1-channel socket config).

- **Auto-building documentation system added**
  - Creates a website on readthedocs.org
  - Uses previous documentation content

Fixes:

- **Pretrigger implementation in event_builder**
  - boost::circular buffer used to implement the pretrigger buffer instead of std::deque.
  - Validated using the ROACH simulator.
- **Stream-closing on node exit**
  - Writers perform a final attempt to close a stream when they exit.
  - Validated by inserting code to purposefully crash a node.

8.3.22 Version: 1.2.3

Release Date: August 28, 2017

New Features:

- **Validation log**
  - This file, documentation/validation_log.md, was added to record changes to Psyllid as they're made.
  - No validation is needed as this is not a functional change.

Fixes:

- **Propagate missing header values to subsequent files**
  - Previously-missing information included voltage offset and range, DAC gain, and frequency min and range.
  - Validated by with a run producing multiple files using the roach_simulator.
- **Prevent invalid duration setting**
  - Setting the duration to 0 caused undefined behavior. This could occur if the value of the duration setting in a dripline request was not an unsigned integer.
  - Now the duration is extracted and checked for validity. So far it just checks that it's not 0.
  - This was validated by by attempting to set the duration to 0. It failed, which was a successful test.

Full Doxygen API Reference