developer.skatelescope.org Documentation

Release 4.1.0

Marco Bartolini

Nov 29, 2022

Contents

This is an emulator for the Correlator Beamformer and its data sending capabilities. It reads data from a data source (typically a Measurement Set, but other sources will be added in the future, like correlator data dumps) and sends it over a network transport to a number of receivers, thus mimicking the CBF-SDP interface.

This is an extensible and configurable package that has been designed to support multiple communication protocols and provide a platform for testing consumers of CBF data payloads. It currently only implements the SPEAD transmission protocol, but other protocols can be added in the future.

CHAPTER 1

Installation

1.1 CAR

This program is distributed by python wheels and all dependencies should be automatically pulled. The only caveat is that some of the dependencies are found in the CAR server instead of PyPI, so we need to point `pip` to the CAR in order to them:

```
# Install directly via pip from the CAR repository
pip install --extra-index-url=https://artefact.skao.int/pypi-all/simple ska-sdp-cbf-
↪emulator
```

1.2 Poetry (Developer)

This project requires packaging using Poetry before being installation. Poetry can be installed to OS python environment using either of the following:

```
# System user install
curl -sSL https://install.python-poetry.org | python3 -

# Install latest version on PyPI
pip install -U poetry
```

For local development and packaging, Poetry can be used in several ways:

```
# Go into the top-level directory of this repository
cd ska-cbf-sdp-emulator

# (if using pyenv) Use pyenv for testing against a specific python version. It is
# recommended to regularly test versions used by continuous integration.
pyenv local 3.10.6

```

(continues on next page)

(continued from previous page)

```
# (if using poetry env) Use Poetry virtual environment to install and use
# project dependencies and package in isolation from python global packages.
poetry env use 3.10.6

# Regenerate the lock file for the active python environment (poetry commands
will always use a virtual environment if one is found).
poetry lock

# Development install to the currently active python environment. This will
# setup a .pth in site-packages that points to the development directory.
poetry install

# Test commands within the poetry virtual environment, e.g.
poetry run pytest

# Alternatively can use the poetry shell instead of `poetry run`.
poetry shell
pytest

# (if using poetry shell) Deactivate Poetry shell
exit

# (if using poetry env, optional) To remove/delete a Poetry environment run
# the following from the top-level directory
poetry env remove 3.10.6

# (if using pyenv, optional) Remove association with pyenv python version
pyenv local --unset
```

Once changes are made and tested, a python wheel may be generated using Poetry to the dist/ directory that can be installed via pip. Building wheel inside a poetry virtual environment will mark the wheel with the specific python version:

```
# Build a local development wheel
poetry build
pip install dist/*.whl
```

This is performed automatically when using pip:

```
# Use pip to install the project (note: -e editable mode does not work here)
pip install --extra-index-url=$PYPI_REPOSITORY_URL/simple .
```

CHAPTER 2

Sender

2.1 Configuration options

The following configuration categories/names are supported:

- `reader`: these are configuration options applied when reading the input Measurement Set.
- `start_chan`: the first channel for which data is read. Channels before this one are skipped. If `start_chan` is bigger than the actual number of channels in the input MS an error is raised.
- `num_chan`: number of channels for which data is read. If `num_chan` + `start_chan` are bigger than the actual number of channels in the input MS then `num_chan` is adjusted.
- `num_timestamps`: number of timestamps to be sent, defaults to `0` which is all of them.
- `num_repeats`: defaults to 1 - number of times the number of timestamps are sent. This will send the same data over and over which is less realistic but imposes less stress on the file-system. TIME values increment with each repetition.
- `transmission`: these are options that apply to the transmission method.
- `scan_id`: the scan_id to use for all payloads in transmission.
- `method`: the transmission method to use, defaults to `spead2`.
- `target_host`: the host where data will be sent to.
- `target_port_start`: the first port where data will be sent to.
- `endpoints`: the endpoints where data will be sent to (see below). If present, `target_host` and `target_port_start` are ignored.
- `channels_per_stream`: number of channels for which data will be sent in a single stream.
- `max_packet_size`: the maximum size of packets to build, used by `spead2`.
- `buffer_size`: the socket buffer size, used by `spead2`.
- `rate`: the maximum send data rate, in bytes/s. Used by `spead2`, defaults to 1 GB/s.

- `time_interval`:: the period of time to wait between sending data for successive data dumps. Positive values are used as-is. A value of `0` means to use the time differences in the `TIME` column of the Measurement Set. Negative values mean to don't wait, sending data as fast as possible.
- `transport_protocol`: the network transport protocol used by spead2. Supported values are `udp` and `tcp`, defaults to `udp`.
- `delay_start_of_stream_heaps`: the number of data heaps to send on each stream before sending the corresponding start-of-stream (SOS) heaps. *0* (default) means don't delay the sending of the SOS heaps, *< 0* means never send the SOS heaps. Note that non-zero values emulate out-of-order transmission for the SOS heaps.

2.2 Endpoint specification

There are two ways to specify the target endpoints where data will be sent to. Note that in both cases the number of streams that are set up equals `number_channels / transmission.channels_per_stream`, where `number_channels` depends on `reader.num_chan` and the input Measurement Set.

- Using `transmission.target_host` and `transmission.target_port`. When using these options then all streams will be sent to `target_host`, and successive streams will be sent to successive ports starting at `target_port`.
- Using `transmission.endpoints`. This option is a single string of comma-separated endpoint specifications. Each endpoint takes the form of `host:ports`, where `ports` is either a single number, or a range like `start-end`. For example, `127.0.0.1:8000,127.0.0.1:8001` and `127.0.0.1:8000-8001` are equivalent.

If the list of endpoints to use is less than the number of streams an error is raised. If it's larger, then the first endpoints are used, and the rest are silently ignored.

CHAPTER 3

Running

An **emu-send** program should be available after installing the package. This program takes a Measurement Set and transmits it over the network using the preferred transmission method.

measurement_set
: The measurement set to read data from

-eb execution_block_id
: An execution block id to monitor for scans

-c config
: A configration file to read options from

-o option
: Additional configuration options in the form of category.name=value

-q quiet
: Additional parameter to silence info logging from standard output

3.1 Tango device wrapper

A Tango device wrapping the emulator sender is available under CBF-SDP Emulator TANGO Devices. The purpose of this Tango device is to be used as a simulation of the real CBF, making it possible to run a full end-to-end SKA system that exercises the visibility data flow.

3.2 In SDP

In the context of the SDP Integration the emulator is deployed as a Helm chart to exercise the visibility receive workflow.

CHAPTER 4

API documentation

This section describes the main entry points for the emulator API. While most users will be using the **emu-send** program, the sender code can be embedded directly into arbitrary python programs, like in the case of the CBF-SDP Emulator TANGO Devices.

4.1 cbf_sdp.packetiser module

Primary send functions for ska-sdp-cbf-emulator

cbf_sdp.packetiser.**packetise**(*config: configparser.ConfigParser, ms: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7f11ccd88d00>, pathlib.Path, str]*)

Reads data off a Measurement Set and transmits it using the transmitter specified in the configuration.

Uses the vis_reader get data from the measurement set then gives it to the transmitter for packaging and transmission. This code is transmission protocol agnostic.

4.2 cbf_sdp.transmitters.spead2_transmitters module

Implementation for the SPEAD2 network transport

This module contains the logic to take ICD Payloads and transmit them using the SPEAD protocol.

class cbf_sdp.transmitters.spead2_transmitters.**Spead2SenderPayload**(*num_baselines=None*, *num_channels=None*)

SPEAD2 payload following the CSP-SDP interface document

cbf_sdp.transmitters.spead2_transmitters.**parse_endpoints**(*endpoints_spec*)

Parse endpoint specifications.

Each endpoint is a colon-separated host and port pair, and multiple endpoints are separated by commas. A port can be a single number or a range specified as "start-end", both inclusive.

class cbf_sdp.transmitters.spead2_transmitters.**transmitter**(*config*)
SPEAD2 transmitter

This class uses the spead2 library to transmit visibilities over multiple spead2 streams. Each visiblity set given to this class' *send* method is broken down by channel range (depending on the configuration parameters), and each channel range is sent through a different stream.

close()
Sends the end-of-stream message

prepare(*num_baselines*, *num_channels*)
Create the sending SPEAD streams

send(*scan_id: int*, *ts: int*, *ts_fraction: int*, *vis: <sphinx.ext.autodoc.importer._MockObject object at 0x7f11ccc9cb80>*)
Send a visibility set through all SPEAD2 streams

Parameters

- **int** – the scan id
- **ts** – the integer part of the visibilities' timestamp
- **ts_fraction** – the fractional part of the visibilities' timestamp
- **vis** – the visibilities

Python Module Index

C

Index

Symbols

C

E

M

P

S

T