HERA Monitor and Control Subsystem Definition

HERA Team

March 14, 2019

1 Introduction

HERA is an international experiment to detect and characterize the Epoch of Reionization (EOR). The telescope is located at the South African SKA site in the Karoo Astronomy Reserve. This note summarizes Monitor and Control (M&C) subsystem for HERA.

Monitor and Control has two main tasks: configuration management and real time metadata logging. Configuration management involves tracking all the physical parts of the telescope and how they are all connected. Real time metadata logging tracks the performance, settings and communications of the subsystems.

The M&C system is built around a database with a well documented table schema and a python software layer to provide a simple developer framework. It also includes various online daemons for monitoring things, and both a front end web-based user interface and a command-line interface to support analysis code.

The organization of this document is as follows: the high-level M&C requirements and the design specifications are in laid out in section 2, the configuration management is described in section 3, the database tables are detailed in section 4 and future plans are sketched out in section 5.

2 Requirements and Specifications

2.1 Requirements

These were developed and established in early 2016 in discussions among the subsystem leads.

1. Ability to fully reconstruct the historical state of the system.
2. All interactions between subsystems must go through or be logged by M&C.
    a. Both subsystems in an interaction are responsible for logging communications to M&C.
    b. Subsystems in an interaction are responsible for logging communications to M&C.
3. Operational metadata (e.g. temperatures, correlator bit occupancies) must be logged to M&C.
4. High availability (M&C must not limit uptime of telescope).
5. M&C is a provider of information about observations to end-users and must be available to them

2.2 Design Specification

These were developed and established in 2016 based on the requirements.

1. SQL database
    a. DB Design principle: every logical sub group has a group of tables. One adds tables to do more things. E.g. different versions of subsystems add new tables. Operations reference which tables they use.
    b. This document (and appendices) will contain all table definitions.

c. Use careful dB design to avoid duplicated data, make table links/data relationships clear, use many-to-one and many-to-many links.

d. Transactions must be used to ensure DB integrity.

e. Must be mirrored in some fashion to observer locations.

2. At least one SW interface layer will be provided.

   a. Its not required to interact with M&C.

   b. Must support relational db (i.e. multiple column primary and foreign keys) and transactions.

3. Hardware

   a. LOM capabilities

   b. Multi-teraByte mirrored disk RAID

   c. Backup machine available on site

3 Configuration Management

The HERA array/part configuration management database is a set of five tables within the larger hera_mc database, which is maintained on-site in the Karoo. The tables are detailed in the Appendix, but they are:

psql table	**in python file**	**with class name**
geo_location	hera_mc/geo_location.py	GeoLocation
station_type	hera_mc/geo_location.py	StationType
parts	hera_mc/part_connect.py	Parts
part_info	hera_mc/part_connect.py	PartInfo
connections	hera_mc/part_connect.py	Connections
cm_version	hera_mc/	
dubitable	hera_mc	

Software is contained in the repository https://github.com/HERA-Team/hera_mc.

The databases are structured primarily around *parts* and *connections*. *Parts* are meant to be single items that, in theory at least, are a thing than can be replaced as a unit. *Connections* define *ports* on a given parts and connect two ports together. Parts are all upper case and ports are all lower case.

All parts and connections are timed in that they have a start and stop time of operation. If stop is `None`, then it is active (it is given a date in the relatively far future). There are currently two special parts (one at each end of the signal chain) that are "geo_located" parts and have entries in the geo_location table: **station** and **node**.

Parts are hooked together via connections of their ports. The signal chain hook-up is given below and shown in Fig. 1.

- **Station**: geo_located position. See prefixes in table station_type (*e.g.* `HH` for herahex)
- **Antenna**: This is also the correlator number which now matches station number. **A**{`#`}
- **Feed**: element feed (Vivaldi). **FDV**{`#`}
- **FEM**: front-end module. **FEM**{`#`}
- **Cable-RFoF**: cable RF-over-fiber FEM-to-PAM. Number matches station/antenna. **CRF**{`#`}
- **PAM**: post-amp module in node. **PAM**{`#`}
- **SNAP**: digitizer/channelizer in node. **SNP**{`#`}
- **Node**: field-deployed node as a part **N**{`#`}
- **Node**: geo_located position for node as a station **ND**{`#`}

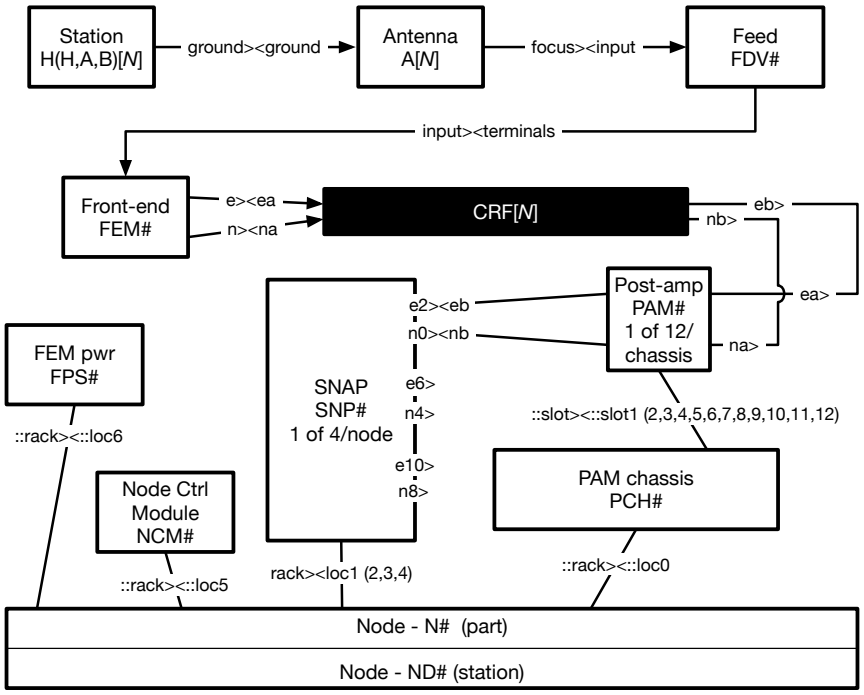

Figure 1: Block diagram of hookup. Filled boxes are cables, blue objects are in the receiverator and green objects are in the container. The line labels indicate the port names/connections.

3.1 Package Modules

This section provides a high-level overview of the python configuration management package modules within hera_mc that are called from scripts or used in interactive sessions.

geo_location.py	Defines `station_type` and `geo_location`. Provides update utility for `geo_location`. Contains `is_in_geo_location` and `is_in_connections`.
geo_handling.py	Contains myriad defs that handle various geo functionalities.
part_connect.py	Defines `parts_paper`, `part_info`, and `connections`. Provides update utilities for `parts_paper` and `connections`. Contains `__get_part_revisions`.
cm_table_info.py	Has mapping and order of tables and classes for initialization.
cm_handling.py	Defines the `Handling` class to handle various configuration management functionalities.
cm_transfer.py	Contains myriad defs that help package and initialize the cm tables.
cm_part_revisions.py	Contains myriad defs that deal with finding revision numbers etc.
cm_hookup.py	Defines the `Hookup` class to help determine and show the full part hookup.
cm_dataview.py	Contains various methods to display cm information.
cm_utils.py	Contains various defs called by other modules.

3.2 Scripts

This section provides a high-level overview of the high-level scripts: geo.py and parts.py. Note that info you type `info` after them, it presents information on them. The structure is

`geo.py` *action* `--arguments`

action is optional with a default.

3.2.1 geo.py

Has various plotting/printing options for station information. Available actions are (note you need the first 3 letters only):

- `geo` [default]: plots and/or lists (`-g`) antenna locations. Plots have the "background" behind them.
- `cofa`: plots and/or lists the position of the center of the array.
- `since`: plots and/or lists the antennas built since the provided date.
- `info`: provides information on the script.

`geo.py -g -l HH67` will show the "background" (selectable via –station-types or -t) of active (selectable via –show-state) antennas with HH67 highlighted. It will provide an overview printout as well.

3.2.2 parts.py

Has various printing options for part information (part info, connections, hookup, types, etc). Available actions are (note you only need the first 2 letters only):

- `hookup` [default]: lists hookup info for parts (as a csv list with no spaces) after -p
- `part_info`: provides a summary of given part(s)
- `conn_info`: provides a summary table of parts connected to given part(s)
- `rev_info`: provides a summary of revisions to given part(s)
- `types`: prints a summary table of part types
- `check_rev`: checks whether a given revision of a part exists
- `overlap_check`: checks that the given part doesn't have overlapping revisions.
- `info`: provides information on the script

`parts.py -p ri5` will provide a table of all hookups to receiverator 5. `parts.py -p ri5 --hookup-cols "station,cable-post-amp(in),f-engine"` will just show those columns.

3.3 Use Cases

Three use cases are identified:

1. In-the-field
2. Remote user
3. Bulk database

3.3.1 In-the-field updates

This is the "operational" mode, where inputs to the on-site database are done via scripts (command-line and/or web), which update the database directly. These scripts will be documented under the repository software.

3.3.2 Remote user

Remote use is the case when a remote user initializes and uses the tables outside of the main database. This assumes that postgresql etc is configured on the remote computer. Note that this mode supports both offline use (essentially steps 3-5 below, which are in bold) and development (probably all steps). The process is as follows:

1. Generate ascii files of the tables in the main database (in the Karoo) (`cm_pack.py [--tables table1,table2,...)`
2. Push to GitHub
3. **Pull down to remote computer**
4. **Install (`python setup.py install`)**
5. **Initialize the updated ones (`cm_init.py`)**
6. If you are developing, make a base version before you start (`cm_pack.py --base`)

Note that generally you should leave off the `--tables` option, since you should use all the tables so as to not break foreign keys (it is left there for "expert" users). Steps 3-5 are bolded, since if you just want to pull down the latest and use it assuming things are up-to-date they are all that are needed (i.e., leave the "packaging to the experts").

This has another option of `--base`, so that you can set up a base copy to which you can revert if need be by including `--base` on both commands (step 6).

3.3.3 Main database updates

This is effectively the opposite of the remote user case, bringing an updated set of table(s) back into the main database. The process is as follows:

1. Generate ascii files of the desired tables from the remote database: (`cm_pack.py --maindb $key [--tables table1,table2,...]`)
2. Push to GitHub
3. Pull down to site computer
4. Install
5. Update existing tables (`cm_init.py --maindb $key [--tables table1,table2,]`)

Note that `$key` is a user supplied string to just make sure you meant to do the main database update. It can be anything, you just need to remember it when you reload it on the main database computer qmaster. As per above, you should generally leave off the `--tables` option so as to do them all.

4 Table Definitions

4.1 Observations . . . 7
4.1.1 hera_obs . . . 7
4.2 Common tables . . . 7
4.2.1 server_status (template) . . . 7
4.2.2 subsystem_errors . . . 7
4.3 RTP Tables . . . 8
4.3.1 rtp_server_status . . . 8
4.3.2 rtp_status . . . 8
4.3.3 rtp_process_events . . . 8
4.3.4 rtp_process_record . . . 8
4.3.5 rtp_task_resource_record . . . 8
4.4 Librarian Tables . . . 9
4.4.1 lib_server_status . . . 9
4.4.2 lib_status . . . 9
4.4.3 lib_raid_status . . . 9
4.4.4 lib_raid_errors . . . 9
4.4.5 lib_remote_status . . . 10
4.4.6 lib_files . . . 10
4.5 Correlator Tables . . . 10
4.5.1 correlator_config_file . . . 10
4.5.2 correlator_config_status . . . 10
4.5.3 correlator_control_state . . . 10
4.5.4 correlator_control_command . . . 11
4.5.5 correlator_take_data_arguments . . . 11
4.5.6 correlator_config_command . . . 11
4.5.7 node_sensor . . . 11
4.5.8 node_power_status . . . 12
4.5.9 node_power_command . . . 12
4.5.10 roach_temperature (deprecated) . . . 12
4.6 QA Info Tables . . . 12
4.6.1 metric_list . . . 13
4.6.2 ant_metrics . . . 13
4.6.3 array_metrics . . . 13
4.7 Site Info Tables . . . 13
4.7.1 weather_data . . . 13
4.8 Configuration Management Tables . . . 13
4.8.1 geo_location :: geo_location.GeoLocation . . . 14
4.8.2 geo_location :: station_type.StationType . . . 14
4.8.3 part_connect :: parts_paper.Parts . . . 14
4.8.4 part_connect :: part_info.PartInfo . . . 14

The formatting of the tables is as follows:

- **Bold font** = primary key
- *Italics* = foreign_key.
- * = NotNull entries

4.1 Observations

4.1.1 hera_obs

This is the primary observation definition table. It is written to by the correlator.

Column	Type	Description
obsid	long integer	start time in floor(GPS) seconds. GPS start adjusted to be within 1 second of LST to lock observations to LST for the night
starttime	double	start time in gps seconds. The start time to full accuracy of the beginning of integration of first visibility
stoptime	double	stop time in gps seconds. The stop time to full accuracy of the end of integration of last visibility
jd_start	double	start time in JD. Calculated from starttime, provides a quick way to filter on JD times.
lst_start_hr	double	decimal hours from start of sidereal day. Calculated from start-time, provides a quick search for matching LSTs

4.2 Common tables

4.2.1 server_status (template)

Common table structure for server status info. **Note: There is no table named server_status. This is the structure used for several subsystem tables named <subsystem>_server_status**.

Column	Type	Description
hostname	string	name of server
mc_time	long	time report received by M&C in floor(gps seconds)
ip_address	string	IP address of server (how should we handle multiples?)
mc_system_timediff	float	difference between M&C time and time report sent by server in seconds
num_cores	integer	number of cores on server
cpu_load_pct	float	CPU load percent = total load / num_cores, 5 min average
uptime_days	float	server uptime in days
memory_used_pct	float	percent of memory used, 5 min average
memory_size_gb	float	amount of memory on server in GB
disk_space_pct	float	percent of disk used
disk_size_gb	float	amount of disk space on server in GB
network_bandwidth_mps	float	Network bandwidth in MB/s, 5 min average. Can be null

4.2.2 subsystem_errors

Subsystem errors/issues

Column	Type	Description
id	long	auto-incrementing error id
time	long	error report time in floor(gps seconds)
subsystem	string	name subsystem with error (e.g. 'librarian', 'rtp')
mc_time	long	time report received by M&C in floor(gps seconds)
severity	int	integer indicating severity level, 1 is most severe
log	text	TBD on format, either a message or a file with the log

4.3 RTP Tables

4.3.1 rtp_server_status

RTP version of the server_status table, see 4.2.1.

4.3.2 rtp_status

High level RTP status

Column	Type	Description
time	long	status time in floor(gps seconds)
status	string	status string, options TBD (might become an enum)
event_min_elapsed	float	minutes elapsed since last event
num_processes	integer	Number of processes running
restart_hours_elapsed	float	hours elapsed since last restart

4.3.3 rtp_process_events

RTP Processing events (per obsid)

Column	Type	Description
time	long	event time in floor(gps seconds)
obsid	long integer	observation identifier, foreign key into hera_obs table
event	string	one of: queued, started, finished, error

4.3.4 rtp_process_record

RTP record of processed obsids (entry added when processing finished)

Column	Type	Description
time	long	record time in floor(gps seconds)
obsid	long integer	observation identifier, foreign key into hera_obs table
pipeline_list	text	concatenated list of tasks
git_version	string	git version of RTP code
git_hash	string	git hash of RTP code

4.3.5 rtp_task_resource_record

RTP record of start and stop times for a task (e.g., omnical) for an obsid, as well as CPU and memory used (if available)

column	type	description
obsid	long integer	observation identifier, foreign key into hera_obs table
task_name	string	name of specific task (e.g., `OMNICAL`)
start_time	long	start time of task in floor(gps seconds)
stop_time	long	stop time of task in floor(gps seconds)
max_mem	float	maximum memory, in MB, consumed by the task; nullable column
avg_cpu_load	float	average CPU load, in number of CPUs, for task (e.g., 2.00 means 2 CPUs used); nullable column

4.4 Librarian Tables

4.4.1 lib_server_status

Librarian version of the server_status table, see 4.2.1.

4.4.2 lib_status

High level Librarian status

Column	Type	Description
time	long	status time in floor(gps seconds)
num_files	long	total number of files in librarian
data_volume_gb	float	total data volume in gigabytes
free_space_gb	float	available space in gigabytes
upload_min_elapsed	float	minutes elapsed since last file upload
num_processes	integer	number of running background tasks
git_version	string	git version of Librarian code
git_hash	string	git hash of Librarian code

4.4.3 lib_raid_status

RAID controller status

Column	Type	Description
time	long	status time in floor(gps seconds)
hostname	string	name of RAID server
num_disks	int	number of disks in RAID server
info	text	TBD – various info from megaraid controller, may be several columns

4.4.4 lib_raid_errors

RAID controller errors/issues

Column	Type	Description
id	long	auto-incrementing error id
time	long	error report time in floor(gps seconds)
hostname	string	name of RAID server with error
disk	string	name of disk with error
log	text	TBD on format, either a message or a file with the log

4.4.5 lib_remote_status

Network bandwidth/health to all remote librarians

Column	Type	Description
time	long	status time in floor(gps seconds)
remote_name	string	name of remote librarian
ping_time	float	ping time in seconds
num_file_uploads	int	number of files uploaded in last 15 minutes
bandwidth_mps	float	bandwidth to remote in Mb/s, 15 minute average

4.4.6 lib_files

File creation log

Column	Type	Description
filename	string	name of file created
obsid	long integer	observation identifier, foreign key into hera_obs table. Can be null.
time	long	file creation time in floor(gps seconds)
size_gb	float	file size in gigabytes

4.5 Correlator Tables

The correlator tables are not all defined yet. Notes on future plans are in section 5.1.

4.5.1 correlator_config_file

List of correlator config files, which specify detailed correlator settings. All files in this table are in the Librarian.

Column	Type	Description
config_hash	string	unique hash for the config
filename	string	name of the config file in the Librarian

4.5.2 correlator_config_status

Config status of the correlator, i.e. which config file is being used by the correlator.

Column	Type	Description
time	long	time of the config status in floor(gps seconds)
config_hash	string	hash for the config in use, foreign key into correlator_config_file table.

4.5.3 correlator_control_state

State of control knobs in correlator.

Column	Type	Description
time	long	time of the control state in floor(gps seconds)
state_type	string	type of control state, one of: 'taking_data', 'phase_switching', 'noise_diode'.
state	boolean	indicator of whether the state_type is true or false

4.5.4 correlator_control_command

Commands issued to the correlator. If the command is 'take_data' or 'update_config', there will be a matching row in the 'correlator_take_data_arguments' table or the 'correlator_config_command' table respectively with the values of the parameters in those commands.

Column	Type	Description
time	long	time the command was sent in floor(gps seconds)
command	string	command sent, one of: 'take_data', 'stop_taking_data', 'phase_switching_on', 'phase_switching_off', 'noise_diode_on', 'noise_diode_off', 'update_config'.

4.5.5 correlator_take_data_arguments

Records the arguments passed to the correlator 'take_data' command.

Column	Type	Description
time	long	time the command was sent in floor(gps seconds), foreign key into correlator_control_command table
command	string	command sent, always 'take_data', foreign key into correlator_control_command table.
starttime_sec	long	time to start taking data in floor(gps seconds)
starttime_ms	integer	milliseconds to add to starttime_sec to set correlator start time
duration	float	duration to take data for in seconds. After this time, the correlator will stop recording
acclen_spectra	integer	accumulation length in spectra
integration_time	float	accumulation length in seconds, converted from acclen_spectra (the conversion is non-trivial and depends on the correlator settings)
tag	string	tag which will end up in data files as a header entry, one of: 'engineering', 'science'.

4.5.6 correlator_config_command

Records the config passed to the correlator 'update_config' command.

Column	Type	Description
time	long	time the command was sent in floor(gps seconds), foreign key into correlator_control_command table
command	string	command sent, always 'update_config', foreign key into correlator_control_command table.
config_hash	string	hash for the config to use, foreign key into correlator_config_file table.

4.5.7 node_sensor

Node temperature and humidity sensor readings

Column	Type	Description
time	long	measurement time in floor(gps seconds)
node	int	integer identifying the node
top_sensor_temp	float	temperature of top sensor reported by node in degrees C
middle_sensor_temp	float	temperature of middle sensor reported by node in degrees C
bottom_sensor_temp	float	temperature of bottom sensor reported by node in degrees C
humidity_sensor_temp	float	temperature of humidity sensor reported by node in degrees C
humidity	float	percent humidity measurement reported by node

4.5.8 node_power_status

Power status for SNAPs, FEMs and PAMs (monitored by nodes)

Column	Type	Description
time	long	measurement time in floor(gps seconds)
node	int	integer identifying the node
snap_relay_powered	bool	power status of the snap relay, True = powered
snap0_powered	bool	power status of the SNAP 0 board, True = powered
snap1_powered	bool	power status of the SNAP 1 board, True = powered
snap2_powered	bool	power status of the SNAP 2 board, True = powered
snap3_powered	bool	power status of the SNAP 3 board, True = powered
fem_powered	bool	power status of the FEM, True = powered
pam_powered	bool	power status of the PAM, True = powered

4.5.9 node_power_command

Commands issued to change the power status for SNAPs, FEMs and PAMs (via the nodes).

Column	Type	Description
time	long	time the command was sent in floor(gps seconds)
node	int	integer identifying the node commanded
part	string	part commanded, one of" 'snap_relay', 'snap0', 'snap1', 'snap2', 'snap3', 'pam', 'fem'.
command	string	command sent, 'on' or 'off'.

4.5.10 roach_temperature (deprecated)

Roach (correlator fpga board) temperatures (deprecated 8/2018)

Column	Type	Description
time	long	measurement time in floor(gps seconds)
roach	string	name of roach (correlator fpga board)
ambient_temp	float	ambient temperature reported by the roach in degrees C
inlet_temp	float	inlet temperature reported by the roach in degrees C
oulet_temp	float	oulet temperature reported by the roach in degrees C
fpga_temp	float	fpga temperature reported by the roach in degrees C
ppc_temp	float	ppc temperature reported by the roach in degrees C

4.6 QA Info Tables

The QA tables are not all defined yet. Notes on future plans are in section 5.2.

4.6.1 metric_list

List of metrics used in antenna or array metrics.

Column	Type	Description
metric	string	name of metric
desc	string	description of metric

4.6.2 ant_metrics

Antenna metrics, by polarization and obsid.

Column	Type	Description
obsid	long integer	observation identifier, foreign key into hera_obs table.
ant	integer	antenna number (≥ 0)
pol	string	polarization, 'x' or 'y'
metric	string	name of metric, foreign key into metric_list table.
mc_time	long integer	time report received by M&C in floor(gps seconds)
val	double	value of metric

4.6.3 array_metrics

Array metrics, by obsid.

Column	Type	Description
obsid	long integer	observation identifier, foreign key into hera_obs table.
metric	string	name of metric, foreign key into metric_list table.
mc_time	long integer	time report received by M&C in floor(gps seconds)
val	double	value of metric

4.7 Site Info Tables

The Site Info tables are not all defined yet.. Notes on future plans are in section 5.3.

4.7.1 weather_data

Weather data from KAT sensors

Column	Type	Description
time	long	status time in floor(gps seconds)
variable	string	name of weather variable (e.g. wind_speed, wind_direction, temperature)
value	float	value of the variable at this time

4.8 Configuration Management Tables

As described in section 3, there are five tables in the configuration management section of the database: (1) geo_location, (2) station_meta, (3) parts_paper, (4) part_info, (5) connections. The following tables summarize them with the following key:

4.8.1 geo_location :: geo_location.GeoLocation

Column	Type	Description
station_name*	character varying(64)	Name of position - never changes
*station_type_name**	character varying(64)	Type of station
datum	character varying(64)	UTM datum
tile	character varying(64)	UTM tile
northing	double precision	UTM coordinate
easting	double precision	UTM coordinate
elevation	double precision	Elevation
created_gpstime*	BigInt	GPS second of creation.

4.8.2 geo_location :: station_type.StationType

Column	Type	Description
station_type_name*	character varying(64)	Station type name
prefix*	character varying(64)	1-2 letter prefix for part station_name
description	character varying(64)	Short description
plot_marker	character varying(64)	Type of matplotlib marker

4.8.3 part_connect :: parts_paper.Parts

Column	Type	Description
hpn*	character varying(64)	HERA part number
hpn_rev*	character varying(32)	HPN revision letter (A-Z)
hptype*	character varying(64)	HPN part type category
manufacturer_number	character varying(64)	Unique serial number for each part
start_gpstime*	BigInt	GPS second when part/rev is activated.
stop_gpstime	BigInt	GPS second when part/rev is de-activated

4.8.4 part_connect :: part_info.PartInfo

Column	Type	Description
hpn*	character varying(64)	HERA part number
hpn_rev*	character varying(32)	HPN revision letter (A-Z)
posting_gpstime*	BigInt	GPS second information was posted
comment*	character varying(1024)	Comment
library_file	character varying(256)	Librarian filename (how to get it there?)

4.8.5 part_connect :: connections.Connection

Column	Type	Description
upstream_part*	character varying(64)	Hera part number of upstream connection
up_part_rev*	character varying(32)	Hera part revision of upstream connection
upstream_output_port*	character varying(64)	Output port on upstream part
downstream_part*	character varying(64)	Hera part number of downstream connection
down_part_rev*	character varying(32)	Hera part revision of downstream connection
downstream_input_port*	character varying(64)	Input port on downstream part
start_gpstime*	BigInt	GPS second when connection started
stop_gpstime	BigInt	GPS second when connection ended

5 Future Plans

5.1 Correlator Table plans

The correlator tables are not all defined yet, the following are notes about suggestions and plans for correlator tables. Most of the correlator data will be recorded in a Redis database (a rolling log, ephemeral), that info needs to be grabbed and put in M&Ctables.

corr_server_status: Correlator version of the server_status table, see 4.2.1, not yet implemented.

1. correlator on/off? **this is a control**
2. Bit statistics (overflows, ADC clipping, bit statistics after bit selects)
3. correlator network stats (dropped packets)
4. Firmware git hash
5. Fengine status
6. Xengine status (might be covered in corr_server_status)
7. Walsh on/off **this is a control** (correlator propagates to node)
8. Noise diode **this is a control** (correlator propagates to node)
9. correlator config (walsh patterns; scaling functions for FFT, bit selection)

10. Test mode outputs (results not control) – very notional
    a. Fengine sync test
    b. Xengine test
    c. Do at beginning and end of night.
    d. Analog tests
        1. Noise diode status
        2. Temperature (i2c device)
        3. Walsh switching (on/off control. Make sure bit pattern is known and put into data set.)
11. SNAP information: all info reported through the correlator
    a. Feed status
    b. PAM status
12. Node information (from Arduino) (Dave, Jack, Zara, Matt Dexter (mdexter@berkeley.edu), Nima) All node info will be reported through the correlator.
    a. Clock status info – syncing
    b. Temperatures (outside + inside, feed?)
    c. Node M&Csoftware git hash

5.1.1 Correlator interfaces complete:

These are done:

1. M&Cinformation the correlator needs to get and write into files
    a. Antenna positions
2. New info added to correlator files (recorded in hera_obs table)
    a. obsid
    b. duration
3. Node information (from Arduino) (Dave, Jack, Zara, Matt Dexter (mdexter@berkeley.edu), Nima) All node info will be reported through the correlator.
    a. SNAP power states
    b. Temperatures in nodes
    c. Power PAM, FEM status (binary)

5.2 QA Future Plans

These are some suggestions for the future, things we might like to see.

1. RTP/online systems
    a. RFI statistics/info (this might be in ant_metrics and array_metrics now)
    b. Calibration statistics (this might be in ant_metrics and array_metrics now)
    c. LST repeatability
    d. TBD other things that come up
2. Offline codes (Major work on how to implement this!! Not on the critical path):
    a. TBD from offline analysis codes

5.3 Site Info Future Plans

The following are suggestions for the future, things we might like to see.

1. site power
2. network status

5.4 Other Future Ideas

1. Basic ionospheric monitoring
2. RFI monitoring