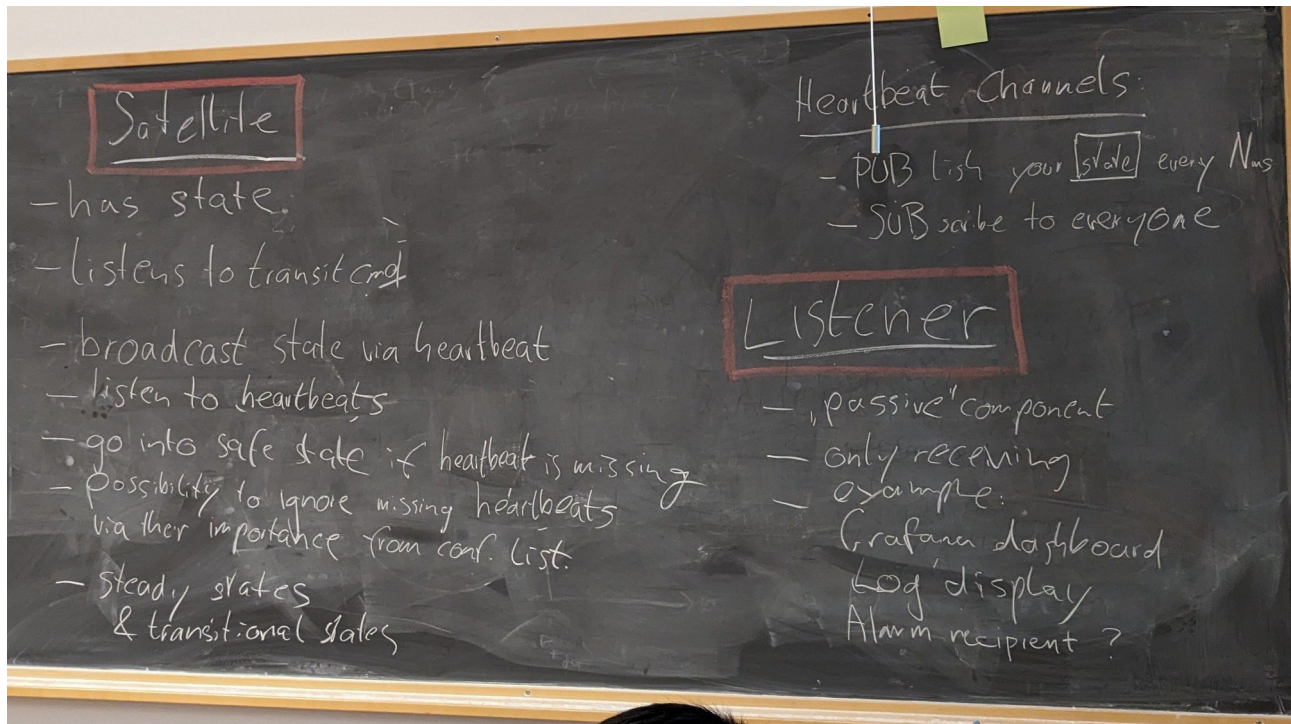




Table of Contents

Outline of the framework.....	2
Satellite states.....	3
Communication channels.....	3
A) Controlling.....	3
B) Logging.....	5
C) Data:.....	6
D) Heartbeating.....	6
CHIRP protocol.....	7
Config files:.....	7

Outline of the framework



network of satellites that have a state and listen to commands

listeners that are passive components that can appear and disappear as they like

last part is a controller that has no state = not a satellite

but is something that can disconnect and reconnect

its the only component that can send commands to the satellites (e.g. a user interface)

question: what happens if we have a problem, e.g. network outage, satellite goes down, ...

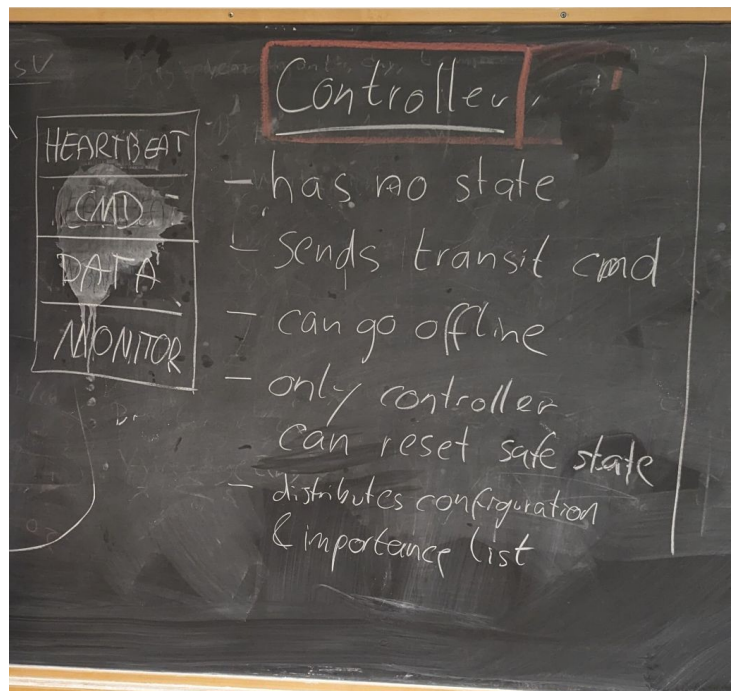
for this we came up with heartbeat system

when the controller distributes setup information, all controllers are told which components they need to care about

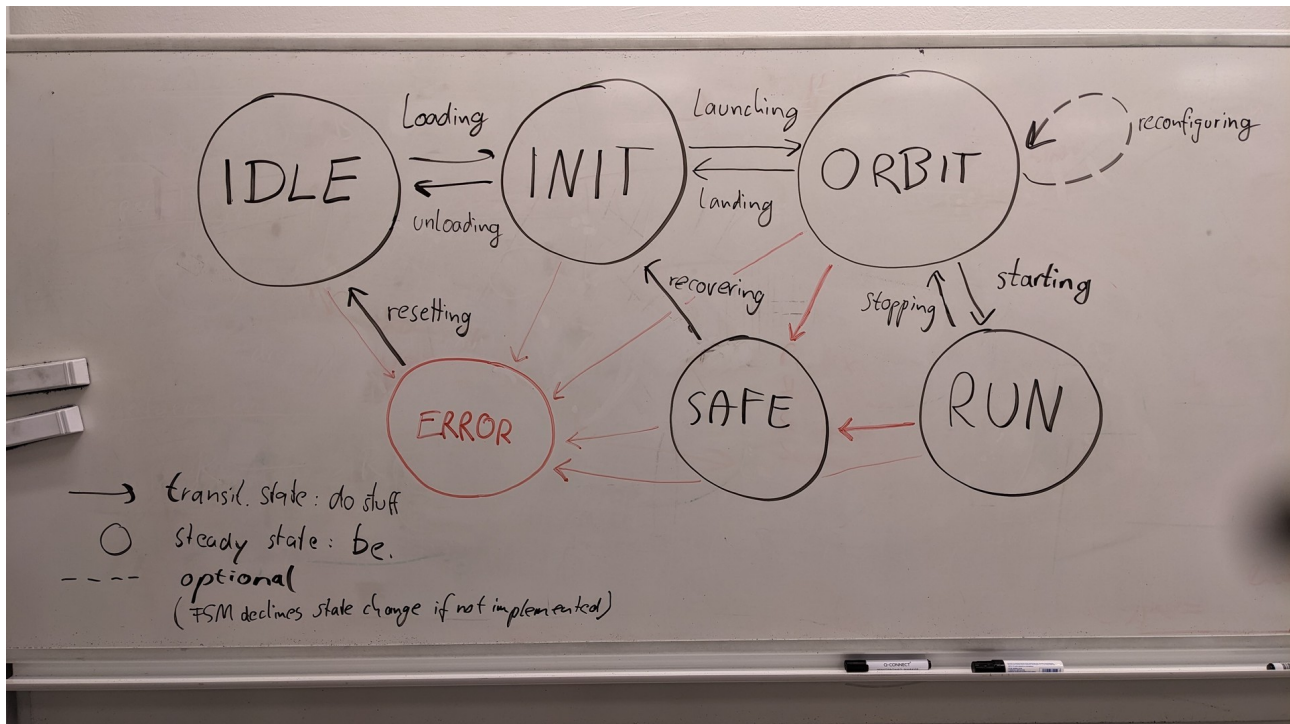
if one goes into error state or its heartbeat stops, the satellite goes into a safe state

the heartbeat channel is also useful because after reconnecting the controller it will know about the state of the entire network after one heartbeat cycle

so the system is not autonomous, but on their own each component can go into a safe state when another has a problem



Satellite states



a satellite is started in idle, the controller can initiate state transitions

regular order of things idle-init-orbit-run and back

extra steps: reconfiguring (e.g. change a voltage) without going all the way down to init

--> fast way to do scans

dashed line = optional, i.e. programmer of a satellite can choose to implement this or not

other special states: error state. only resettable to idle by manual intervention of user (reset the error after fixing the problem - might be something in software, could also be a hardware interlock switch if implemented by user).

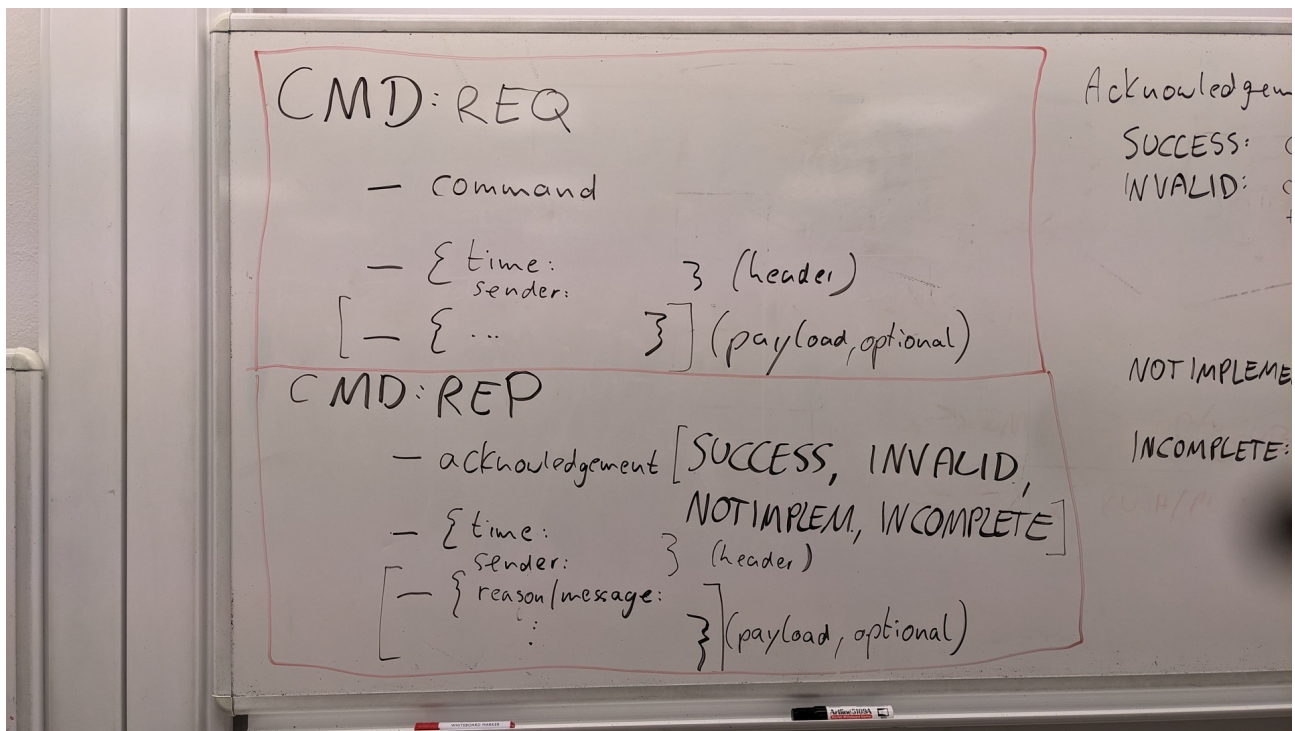
when the other satellites in orbit or run state detect this, they go to safe state

Communication channels

A) Controlling

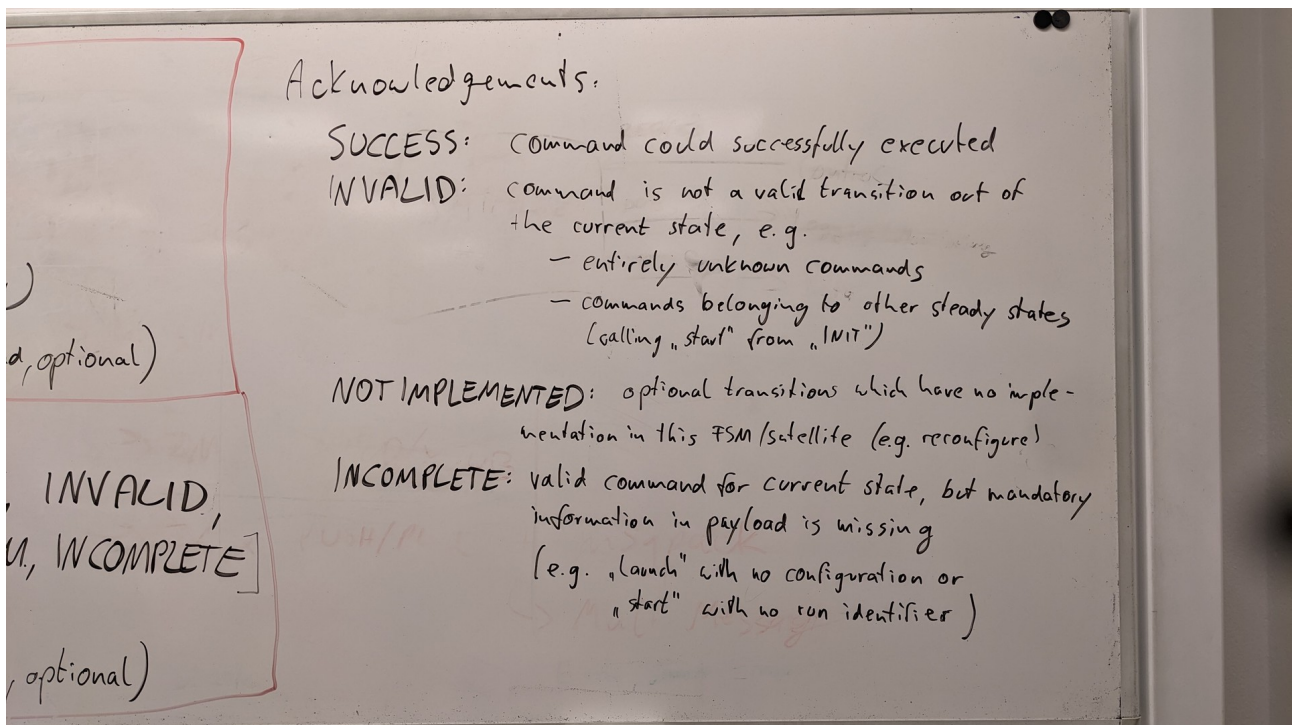
command request: consist of multiple parts

- 1) command, e.g. "load"
- 2) header (time, sender)
- 3) optional parameters, e.g. config options and parameter values for a configure commands

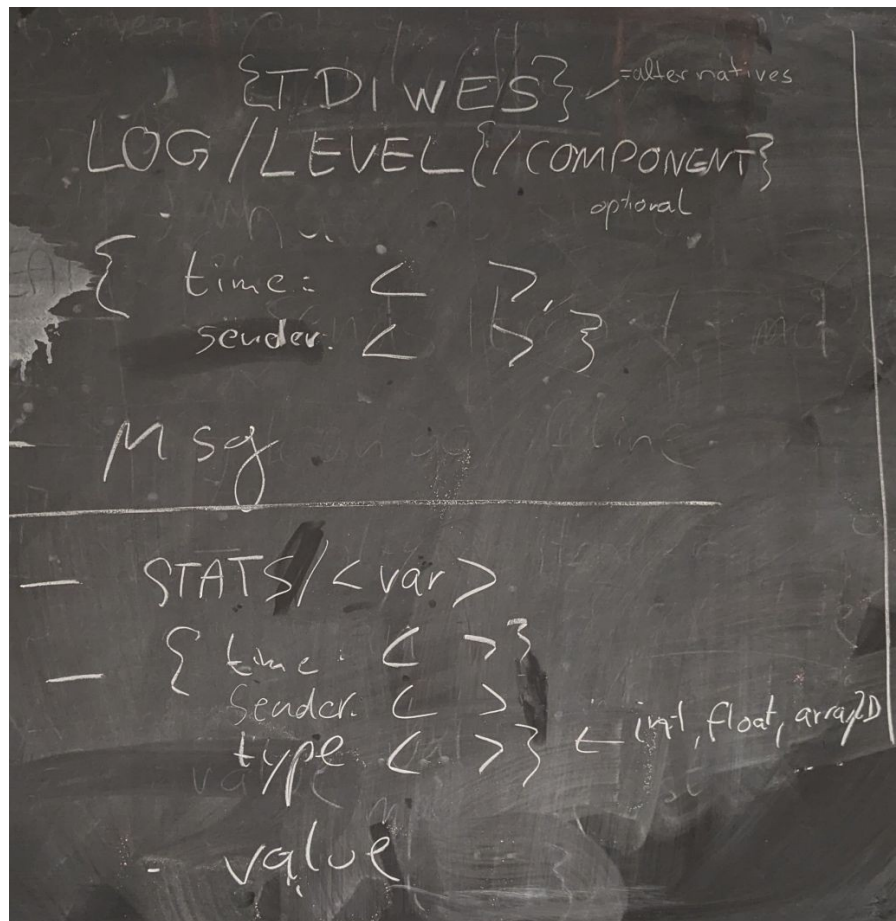


reply

- 1) one of a number of possible acknowledgments (see below)
- 2) header (time, sender)
- 3) optional payload in return



B) Logging

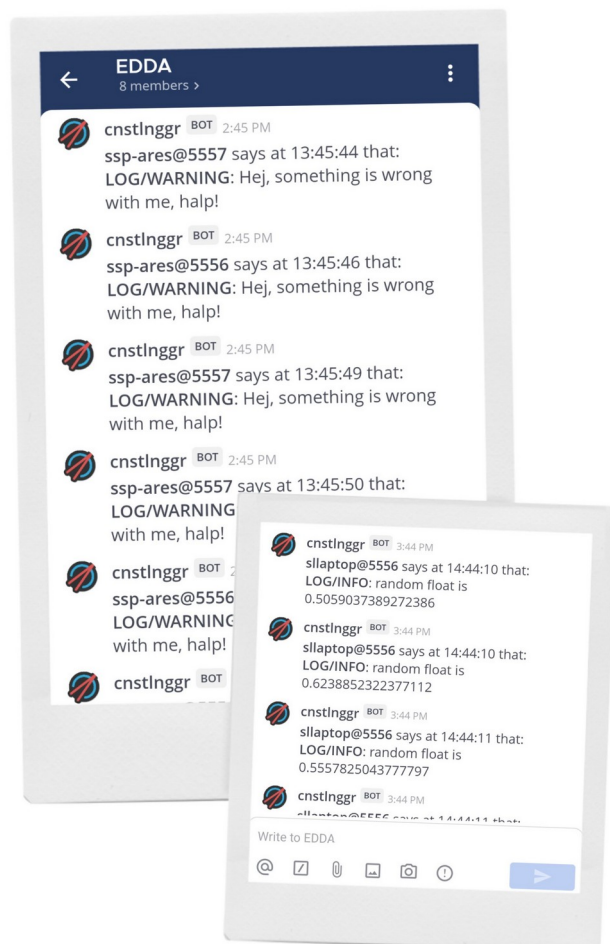


two other channels, work similar

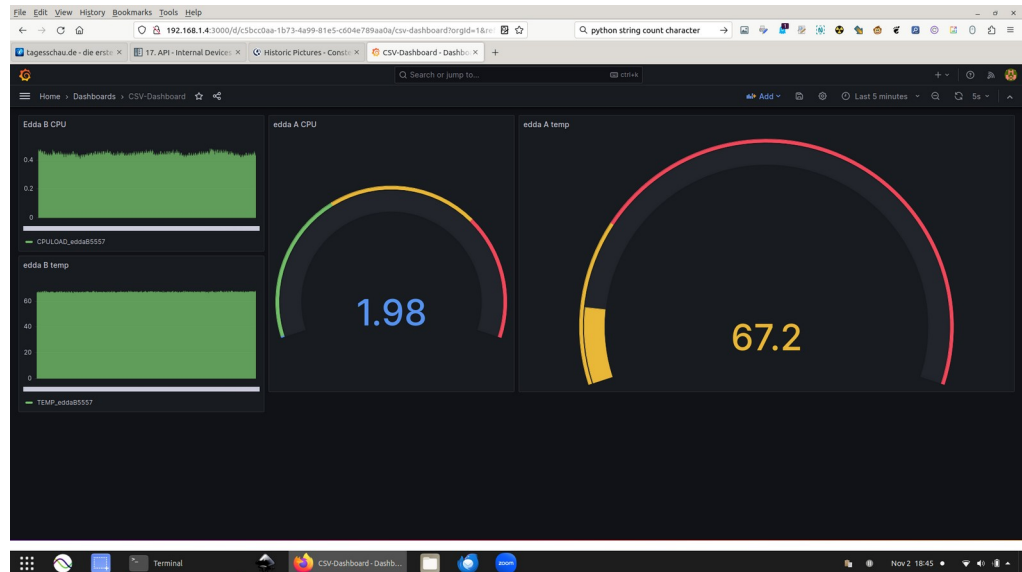
1. monitoring channel, e.g. trigger rates, temperatures – auxiliary information that helps the user monitor the state of the system contains first an identification then a header – time, sender, type of data to follow
2. log channel, that contains a log level (e.g. trace, debug, info, warning, error, status) also a header, then a log message

the listeners can chose which topic to subscribe to (“LOG/DEBUG”, “STAT/TEMP”) only messages which have a subscriber will be distributed over the network

example implementations: mattermost logger that collects log messages and sends them to the network



Other example: first hacky implementation: python code to read stat of raspberry cpu temp and publish on a grafana dashboard



C) Data:

always a pipeline, ie point-to—point connection
satellite that delivers data, another satellite that writes it to disk

very fast transfer for message blocks >1kb
(just for very small data blocks the network overhead takes over)

D) Heartbeating

two classes

- a) a thread that every second broadcasts the state of the satellites
- b) a heartbeat checker that listens to other's heartbeats

Tested implementation

once you go into orbit state, you listen to the heartbeats of the other satellites
screenshot example:

once one of them is gone or failure triggered, the other two go into safe state

```
python satellite.py -h/cconstellation-tests
~/constellation-tests [main] python satellite.py
2023-11-02 11:13:07,233 INFO __main__: Writing current state machine layout to fsm.png
2023-11-02 11:13:07,339 INFO __main__: Satellite listening on command port 2399
2023-11-02 11:16:15,231 INFO root: Satellite Initialized.
2023-11-02 11:16:52,443 INFO root: Registered heartbeat check for 192.168.1.13061234
2023-11-02 11:16:57,456 INFO root: Registered heartbeat check for 192.168.1.197061234
2023-11-02 11:18:07,322 INFO root: started hb thread
2023-11-02 11:18:07,323 INFO root: Satellite Prepared. Acquisition ready.
2023-11-02 11:18:07,323 INFO root: Thread 0 starting heartbeat check
2023-11-02 11:18:07,324 INFO root: Thread 1 starting heartbeat check
2023-11-02 11:18:31,565 ERROR root: There is a failure.

python sub_client_multimessage.py
~/constellation-tests/logsdashbox [main] python sub_client_multimessage.py
-pub 192.168.1.113:5555 --pub 192.168.1.113:5556 --topic LOG/INFO
Connecting to 192.168.1.113:5555
Subscribing to topics ['LOG/INFO']
Connecting to 192.168.1.113:5556
Subscribing to topics ['LOG/INFO']
10:18:07 INFO ssp-ares@5556 Satellite Prepared. Acquisition ready.
10:18:09 INFO ssp-ares@5556 Satellite Prepared. Acquisition ready.
10:18:11 ERROR ssp-ares@5556 There is a failure.
10:18:31 WARNING ssp-ares@5556 Transitioned to Safe state.

python data-receiver -h/cdata
Received event {'eventid': 1106483, 'time': 1698918993.9420443}
Received event {'eventid': 1106484, 'time': 1698918993.9420483}
Received event {'eventid': 1106485, 'time': 1698918993.9420521}
Received event {'eventid': 1106486, 'time': 1698918993.9420562}
Received event {'eventid': 1106487, 'time': 1698918993.9420603}
Received event {'eventid': 1106488, 'time': 1698918993.9420644}
Received event {'eventid': 1106489, 'time': 1698918993.9420685}
Received event {'eventid': 1106490, 'time': 1698918993.9420726}
Received event {'eventid': 1106491, 'time': 1698918993.9420767}
Received event {'eventid': 1106492, 'time': 1698918993.9420808}
Received event {'eventid': 1106493, 'time': 1698918993.9420849}
Received event {'eventid': 1106494, 'time': 1698918993.9420890}
Received event {'eventid': 1106495, 'time': 1698918993.9420931}
Received event {'eventid': 1106496, 'time': 1698918993.9420972}
Received event {'eventid': 1106497, 'time': 1698918993.9421013}
Received event {'eventid': 1106498, 'time': 1698918993.9421054}
Received event {'eventid': 1106499, 'time': 1698918993.9421095}
```

CHIRP protocol

for network discovery – don't want to have to enter all IPs and ports by hand when setting up a measurement environment

when a satellite o.ä starts it will send a message offering up what it is and what services it offers

If a satellite starts later, it can send a request, e.g. for a logging service, and all logging services on the network that offer this service will reply again

<https://gitlab.cern.ch/constellation/constellation/-/blob/master/docs/protocols/chirp.md>

Config files:

Two example ideas (toml, yaml)

Need to establish a way that the config is provided to the satellites
then the rest can also be opened to user implementations

example.yaml 1.16 KiB

```
1 # This is a collection of ideas how a config file could look like in YAML
2 myConstellation:
3   satellites:
4     eddaA:
5       threshold: 123
6       system:
7         importance: essential
8
9     eddaB:
10      # parameters are the ones for the specific satellite
11      compliance: .105
12      voltage: 50
13      # system section defines behavior in constellation
14      system:
15        # system parameters throw error for "additional" params, e.g. one with a typo
16        importance: essential
17        start_after: eddaA
18
19    eddaTemp:
20      refresh_rate: 1
21      calib_file: "/data/eddaTemp/2023
22      system:
23        importance: optional
24
25    storageA:
26      file_pattern: "edda_a_run_%i.bin"
27      receive: eddaA
28
29    storageB:
30      file_pattern: "edda_b_run_%i.bin"
31      receive: eddaB
```

example.toml 756 B

```
1 # This is a collection of ideas how a config file could look like in TOML
2 [Constellation]
3 name = "myConstellation"
4
5 [eddaA]
6 importance = "essential"
7
8 [eddaB]
9 voltage = 50
10 compliance = .105
11 [eddaB.system]
12 importance = "essential"
13 start_after = "eddaA"
14
15 [eddaTemp]
16 refresh_rate = 1
17 calib_file = "/data/eddaTemp/20231102/my_calib_file.csv"
18 [eddaB.Temp]
19 importance = "optional"
20
21 [storageA]
22 file_pattern = "edda_a_run_%i.bin"
23 receive = "eddaA"
24
25 [storageB]
26 file_pattern = "edda_b_run_%i.bin"
27 receive = "eddaB"
28
29 [alarm]
30 mattermost_url = "https://mattermost.web.cern.ch/key/hufw87cw4fgc"
31 alarm_level = "WARNING"
```