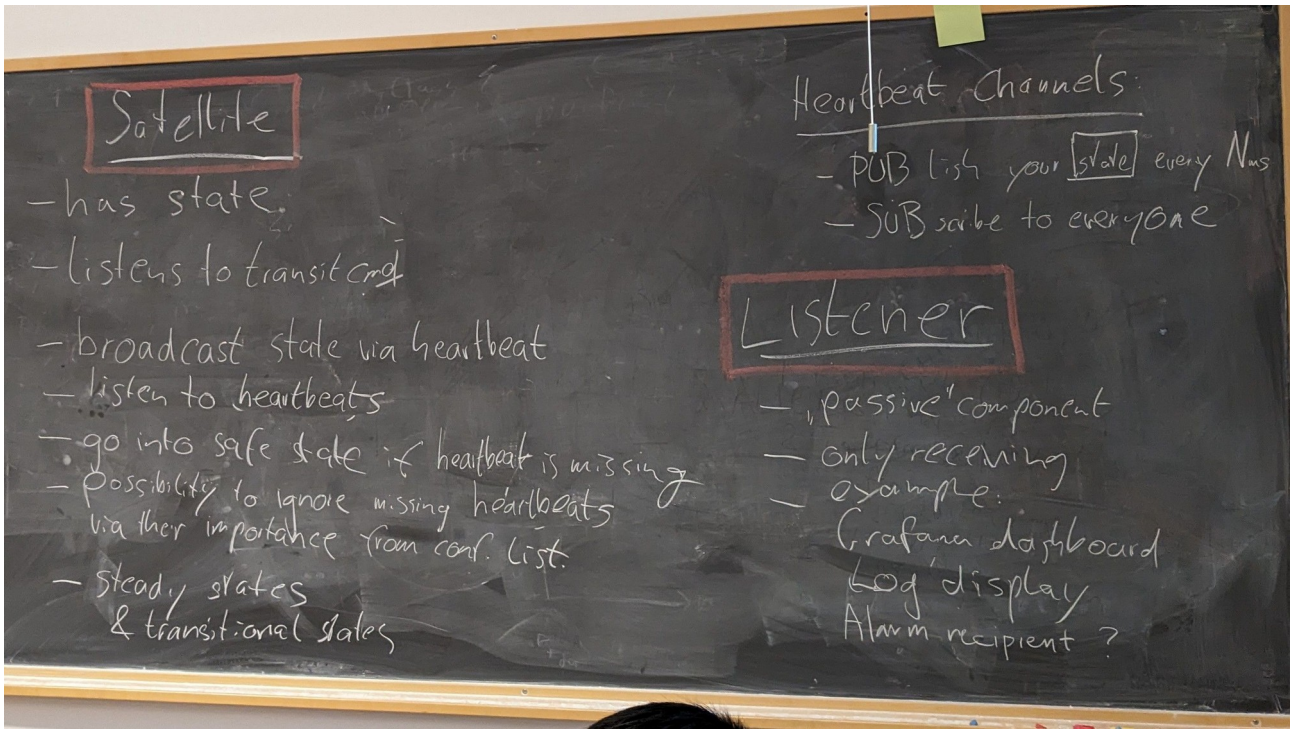
1st Hackathon – Summary of Constellation 3.11.2023

Table of Contents

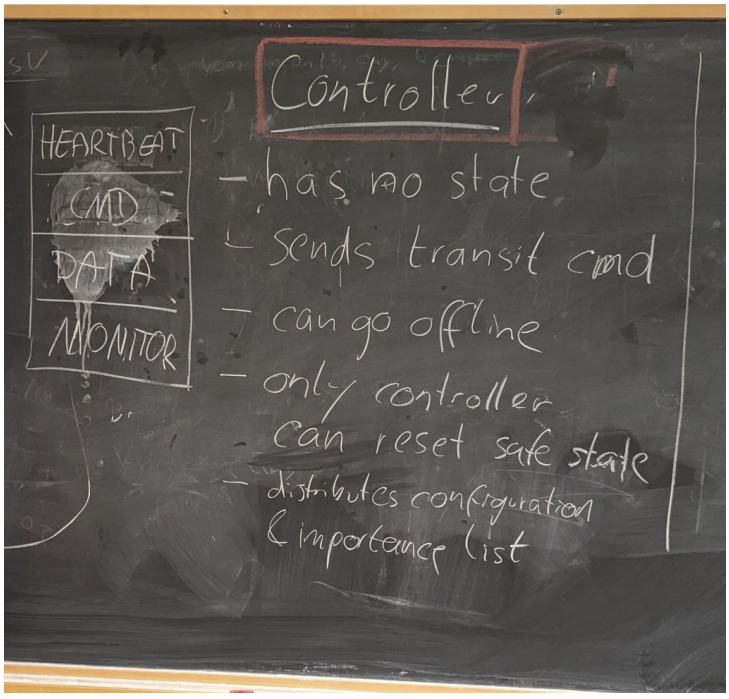
Outline of the framework



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)

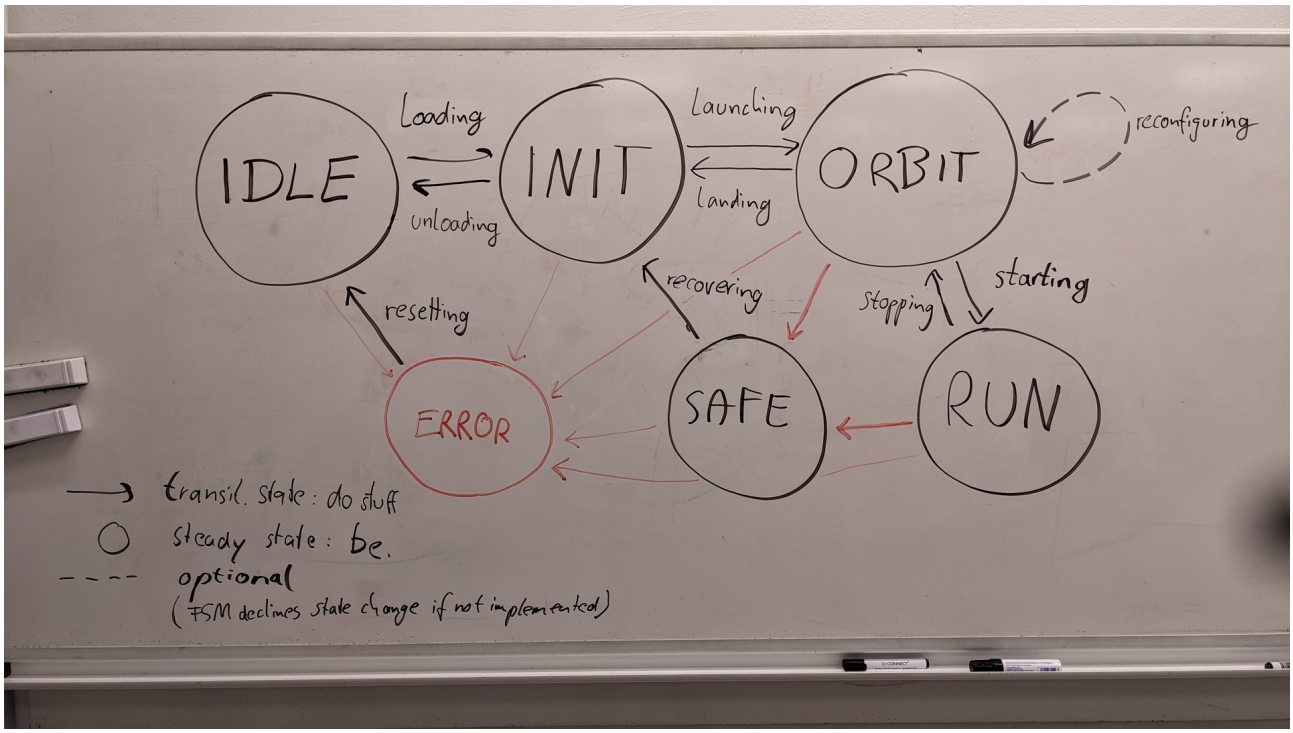


question: what happens if we have a problem, e.g. network outrage, satellite goes down, ...
for this we came up with heartbeat system
when the controller distributes setup information, all controllers are told which componentes they need to care about
if one goes into error state or its heartbeat stops, 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 transiions

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

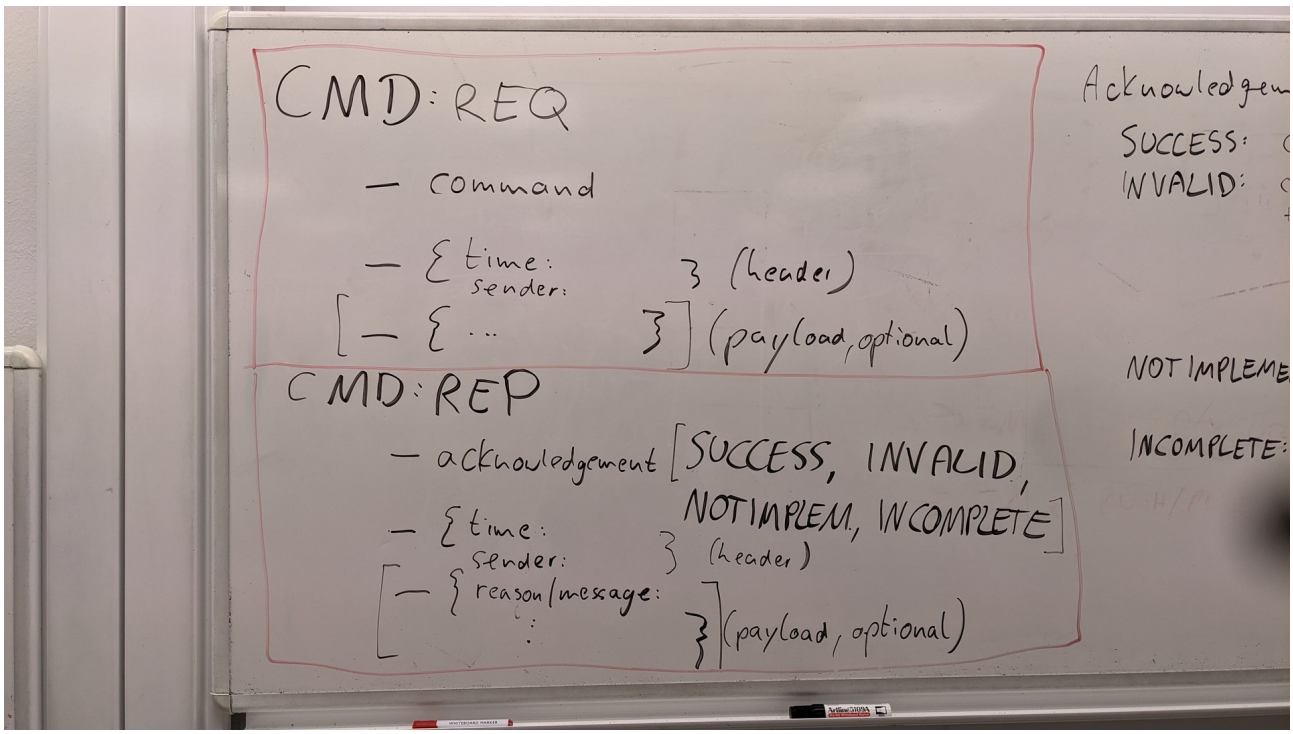
extra steps: reconfiguring (e.g. change a voltage) without going all the way down to init
--> fast way to do scans
dashed line = optional, i.e. programmer of a satellite can chose to implement this or not

other special states: error state. only resetable 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

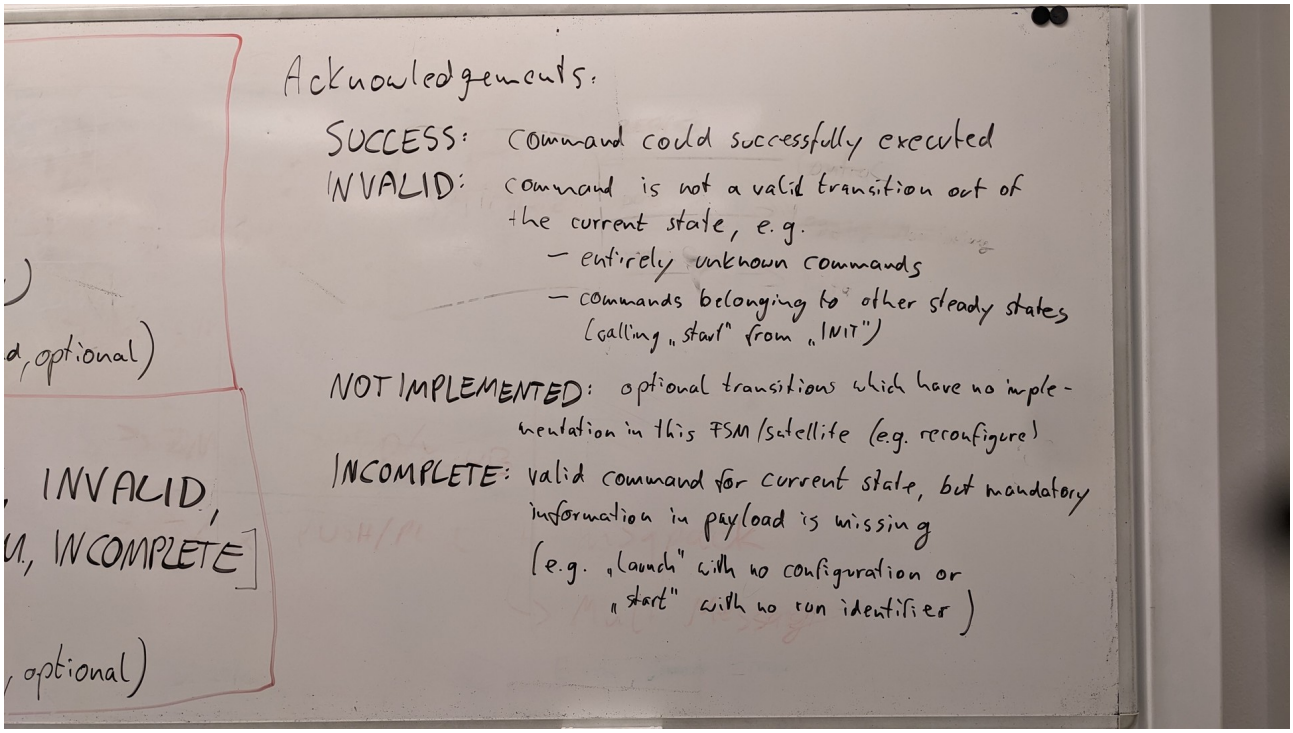
Communication channels

A) Controlling

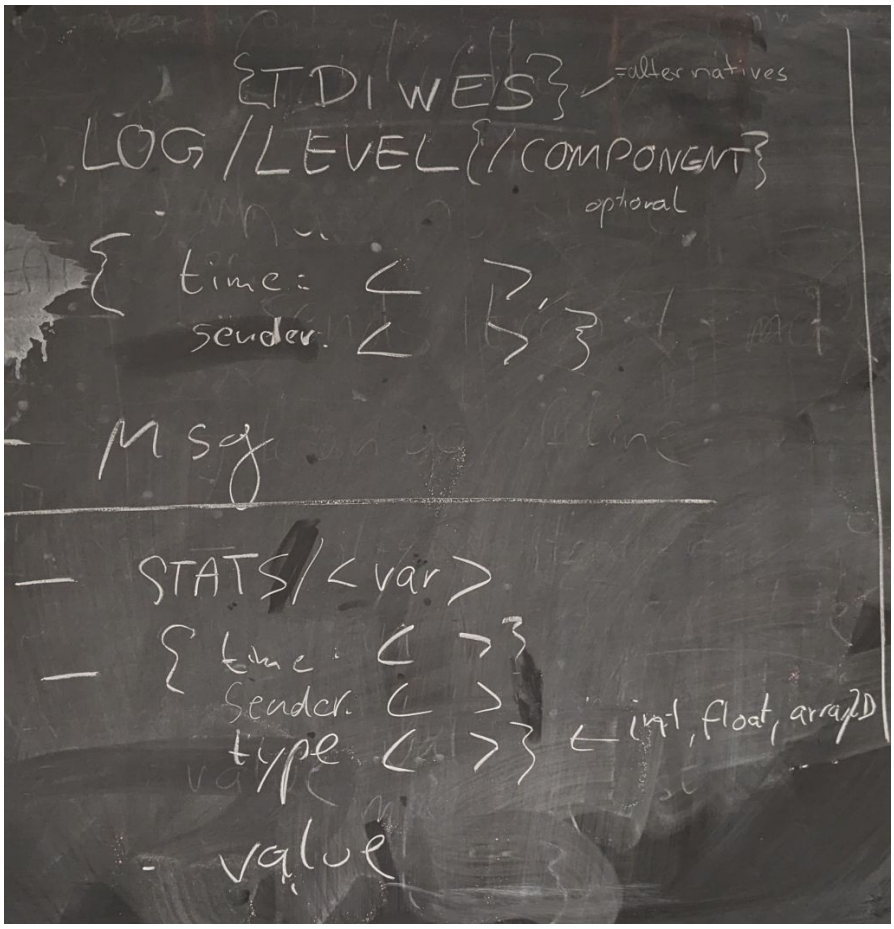
command request: consist of multiple parts
1) command , e.g. “load”
2) header (time, sender)
3) optional parameters, e.g. config options and parameter values for a configure commands



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



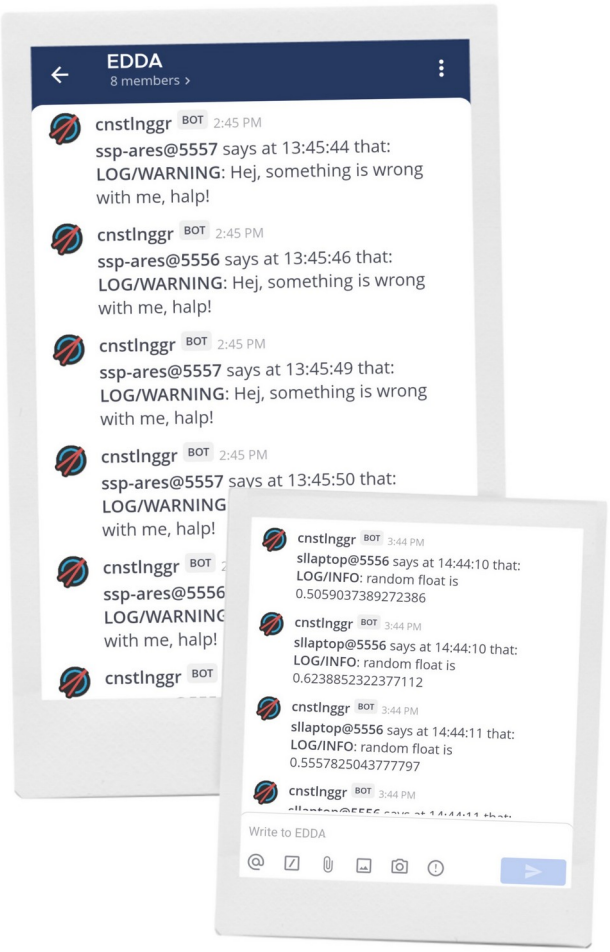
B) Logging



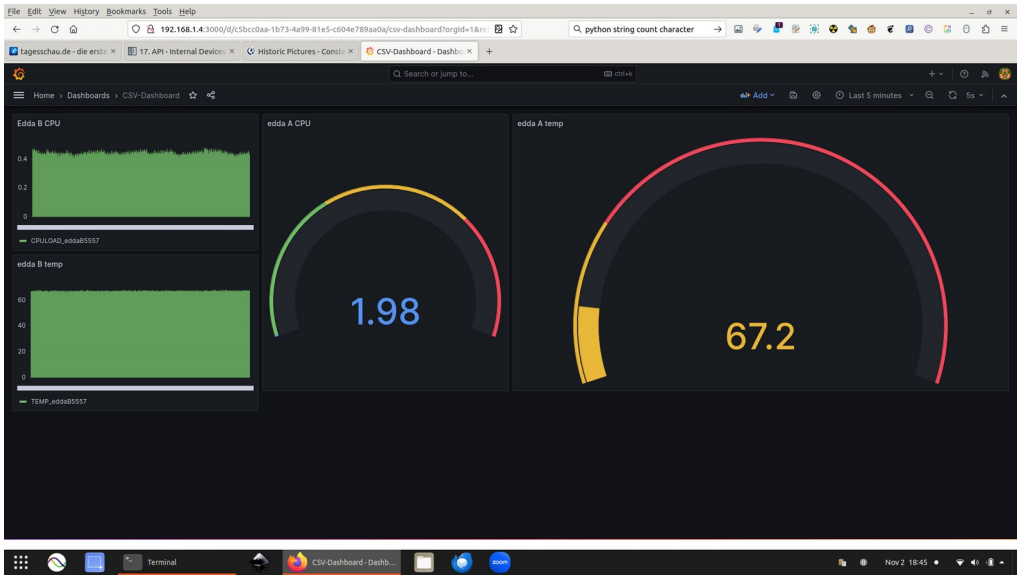
two other channels, work similar
1. monitoring channel , e.g. trigger rates, temperatures – auxiliary information that helps the user monitor the state of the system
contains first an idenfitication
then a header – time, sender, type of data to follow
2. log channel, that contains a log level (e.g. trace, 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:

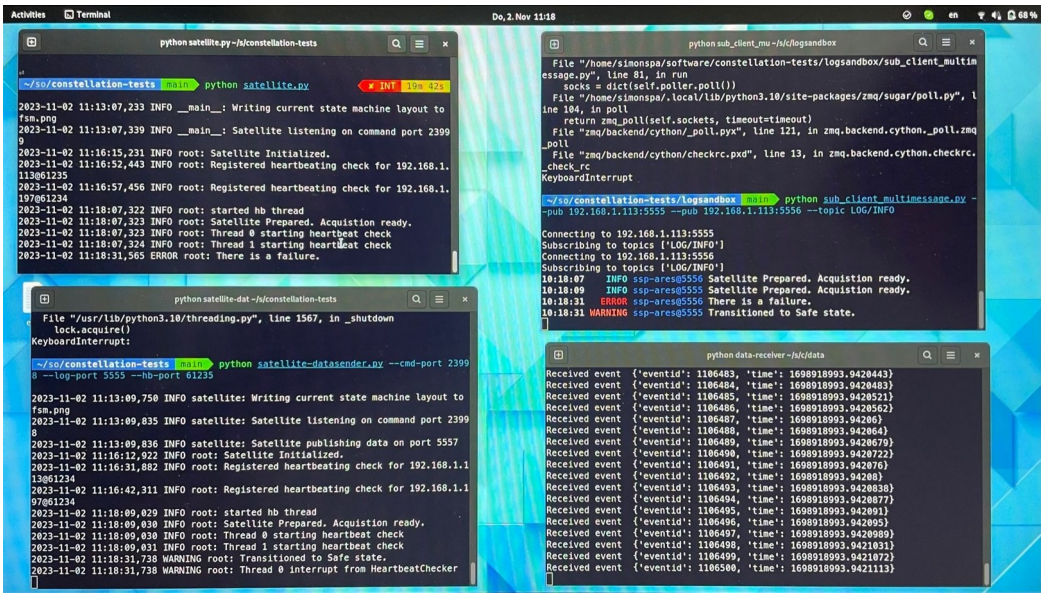
always a pipeline, ie point-to—point connection
satellite that delivers data, another satellite that writes it to disk

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



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

```yaml
# This is a collection of ideas how a config file could look like in YAML
myConstellation:
  satellites:
    eddaA:
      threshold: 123
      system:
        importance: essential

    eddaB:
      # parameters are the ones for the specific satellite
      compliance: .105
      voltage: 50
      # system section defines behavior in constellation
      system:
        # system parameters throw error for "additional" params, e.g. one with a typo
        importance: essential
        start_after: eddaA

    eddaTemp:
      refresh_rate: 1
      calib_file: "/data/eddaTemp/2023
      system:
        importance: optional

    storageA:
      file_pattern: "edda_a_run_%i.bin
      receive: eddaA

    storageB:
      file_pattern: "edda_b_run_%i.bin
      receive: eddaB
```

example.toml 756 B

```toml
# This is a collection of ideas how a config file could look like in TOML
[Constellation]
name = "myConstellation"

[eddaA]
importance = "essential"

[eddaB]
voltage = 50
compliance = .105
[eddaB.system]
importance = "essential"
start_after = "eddaA"

[eddaTemp]
refresh_rate = 1
calib_file = "/data/eddaTemp/20231102/my_calib_file.csv"
[eddaB.Temp]
importance = "optional"

[storageA]
file_pattern = "edda_a_run_%i.bin"
receive = "eddaA"

[storageB]
file_pattern = "edda_b_run_%i.bin"
receive = "eddaB"

[alarm]
mattermost_url = "https://mattermost.web.cern.ch/key/hufw87cw4fgc"
alarm_level = "WARNING"
```