SPRINT Documentation

February 1, 2022

Contents

1 Introduction

Here begins the documentation for SPRINT, or the Scheduling Planning Routing Inter-satellite Network Tool. SPRINT can be used to simulate the actions of a constellation by utilizing crosslinks to consolidate data to be downlinked at scheduled times. The scheduling problem is solved utilizing a MILP (Mixed Integer Linear Problem) solver. If you wish to learn more about the theory and the analysis behind SPRINT, please see the theses and papers that have been published on the subject:

1. Kennedy, A. K., "Planning and scheduling for earth-observing small satellite constellations," Thesis, Massachusetts Institute of Technology, 2018. https://dspace.mit.edu/handle/1721.1/120415
2. Holden, B. G., "Onboard distributed replanning for crosslinked small satellite constellations," Thesis, Massachusetts Institute of Technology, 2019. https://dspace.mit.edu/handle/1721.1/122513
3. Mary Dahl, Juliana Chew and Kerri Cahoy. "Optimization of SmallSat Constellations and Low Cost Hardware to Utilize Onboard Planning," AIAA 2021-4172. ASCEND 2021. November 2021. https://arc.aiaa.org/doi/abs/10.2514/6.2021-4172

This document will cover the overall structure of SPRINT along with how to use it, both for a single simulation and for a separated simulation that can be put on hardware such as Raspberry Pis.

2 File Structure

SPRINT is structured as follows:

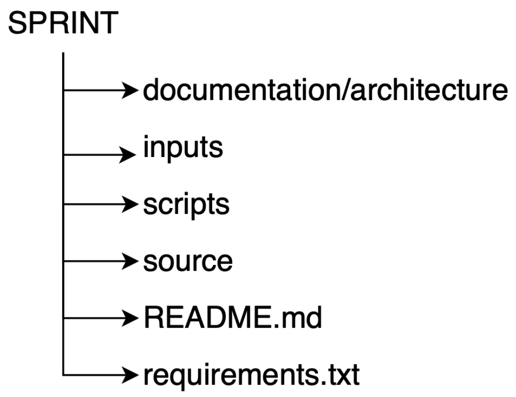


Figure 1: The general file structure of SPRINT. The file SPRINT represents the entire repository.

This entire document lies within the `documentation/architecture` directory, along with diagrams of previous versions. The directory `inputs` houses all input files for the simulation to configure the simulation, satellite constellations, ground station network, and more. How to format said inputs is mentioned in Section 4.1. Under `scripts`, there are `.sh` files for running SPRINT on the terminal, with instructions under Section 4.1. The directory `source` houses subrepositories that have the main simulation software SPRINT. (The main subrepository under `source` is `circinus_sim`, which has the software for satellites, ground stations, and planners.)

3 Setting up SPRINT

Please follow the set-up instructions in the GitHub at
https://github.com/MIT-STARLab/SPRINT

1. Clone the repo: `git clone git@github.mit.edu:star-lab/SPRINT.git`
2. Initialize the appropriate submodules:
   (a) `cd SPRINT/source`
   (b) `git submodule init circinus_global_planner circinus_orbit_link_public circinus_orbit_propagation circinus_orbit_viz circinus_sim circinus_tools`
   (c) `git submodule update`
3. Set up your environment:
   (a) Install and configure your default python and pip to Python 3.6, (recommended in a virtual environment, see next step).
      i. Recommended: Direct installation, if needed: Download from https://www.python.org/downloads/. Note that the global planner code is currently tested with Python 3.6.7.
      ii. Not recommended: alternatively Homebrew, pyenv to set to 3.6.
      iii. Confirm your version of Python (python --version) & location of the installation (which python) is the same for all subsequent steps.
      iv. Consider upgrading pip: python3.6 -m pip install --upgrade pip
   (b) Make a virtual environment
      i. Install virtualenv: python3.6 -m pip install virtualenv
      ii. Create virtual environment:
         A. OS X: `python3.6 -m virtualenv --python=/usr/local/bin/python3.6 VENV_DIR/`
         B. Windows: `virtualenv --python=LOCATION_OF_PYTHON.EXE VENV_DIR`
         C. Ubuntu: `virtualenv -p /usr/bin/python3.6 VENV_DIR`
      iii. Activate the virtual environment
         A. OS X or Ubuntu: source venv_dir/bin/activate

        B. Windows: source venv_dir/Scripts/activate

  (c) Install required python packages.
    i. cd to SPRINT base directory
    ii. pip install -r requirements.txt
    iii. Install Gurobi
        A. Download and install Gurobi 8.0.0
        B. Acquire and activate Gurobi License (Academic is free if appropriate)
        C. Note: if you utilize Windows Subsystem for Linux, there is a bug that will require you to reinstall the Gurobi license once daily.

  (d) Framework setting (this step may not be required)
    i. `nano  /.matplotlib/matplotlibrc`
    ii. add line: `backend:  TkAgg`

As a note, the package requirements for SPRINT are listed in `requirements.txt`.

4 Using SPRINT

There are two configurations for using SPRINT: single and separated simulations. The single simulation has all aspects of the simulation – satellites, ground stations – running on a single processor, typically a computer. The separated simulation uses socket programming to run SPRINT on multiple pieces of hardware concurrently to simulate message passing between satellites. The separated simulation can be run on separate hardware or on a single machine with multiple processes.

As a note: Please use the master branch for both simulations.

4.1 Setting Inputs

SPRINT supports variable ground station networks, constellations, and (un)scheduled observations. These can be set up using the `case_gen.py` script in

`scripts/tools/case_generator`

There are a number of example models already available, found in

`/inputs/reference_model_definitions`

and full cases in `/inputs/cases`.

Please refer to the GitHub page under `/inputs/` for more information.

4.1.1 Setting MILP Solvers

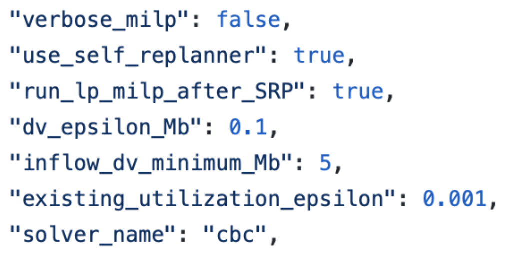
Supported Solvers Currently, the supported solvers are Gurobi, CBC, and GLPK. Here are links below to their respective documentation:

- **Gurobi**:https://www.gurobi.com
- **CBC**: https://projects.coin-or.org/Cbc
- **GLPK**: https://www.gnu.org/software/glpk/

Although Gurobi is quite fast, it is *not* compatible with ARM-based processors such as Raspberry Pis. CBC and GLPK are slower, free alternatives, with CBC performing better.

SPRINT uses Gurobi by default (with Pyomo as its interface) as its MILP solver for both the Local and Global Planner.

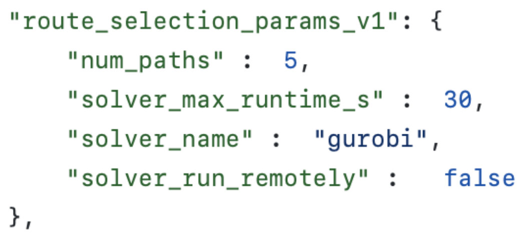
To change the Local Planner's solver, navigate to
`./inputs/general_config/lp_general_params_inputs.json`
at approximately line 30. Change the value for "solver_name":



```
"verbose_milp": false,
"use_self_replanner": true,
"run_lp_milp_after_SRP": true,
"dv_epsilon_Mb": 0.1,
"inflow_dv_minimum_Mb": 5,
"existing_utilization_epsilon": 0.001,
"solver_name": "cbc",
```

Figure 2: A picture of the Local Planner solver specification in `lp_general_params_inputs.json`. Here, the solver is set to cbc.

To change the Global Planner's solver, navigate to
`./inputs/general_config/gp_general_params_inputs.json` at about line 30. Change the value for "solver_name":



```
"route_selection_params_v1": {
    "num_paths" :  5,
    "solver_max_runtime_s" :  30,
    "solver_name" :  "gurobi",
    "solver_run_remotely" :   false
},
```

Figure 3: A picture of the Local Planner solver specification in `gp_general_params_inputs.json`. Here, the solver is set to gurobi.

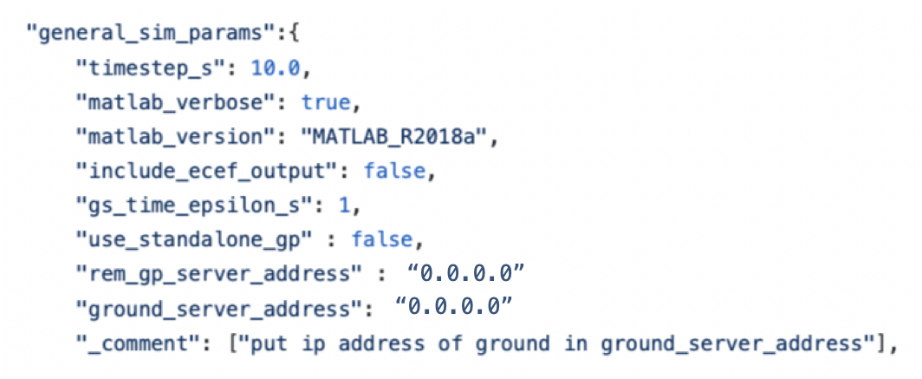
Installing Gurobi Gurobi is a paid MILP solver. However, if you are affiliated with an educational institution, you are in luck!

To get a free academic license, navigate to this link: https://www.gurobi.com/academia/academic-program-and-licenses/.

Follow the directions there. As a note, these licenses periodically expire. If you use Windows Subystem for Linux 2, there is a bug that requires the license to be renewed every day. You can renew your license from here as well.

4.1.2 Separated-Specific

Before running the separated SPRINT, determine the address of the ground station network (this is usually the IP address of the computer). Enter this IP address in inputs/sim_general_config.json under the field `rem_gp_server_address` as shown below:



```
"general_sim_params":{
    "timestep_s": 10.0,
    "matlab_verbose": true,
    "matlab_version": "MATLAB_R2018a",
    "include_ecef_output": false,
    "gs_time_epsilon_s": 1,
    "use_standalone_gp" : false,
    "rem_gp_server_address" : “0.0.0.0”
    "ground_server_address": “0.0.0.0”
    "_comment": ["put ip address of ground in ground_server_address"],
```

Figure 4: A picture of the ground station network server address initialization. Here, the IP address of the ground station network is 0.0.0.0.

At initialization, the ground station network's server waits for connections from satellites that send JOIN messages. Thus, each satellite should have the ground station network IP address entered. If the satellites are running on individual Raspberry Pis, for instance, each Raspberry Pi should have the IP address of the ground server entered.

As a note, the port number for the ground station network is assumed to be 54201. (This can be changed in `./source/Ground_Sim/Ground_Sim.py`.) The port numbers for satellite servers are dynamically assigned and reported.

4.2 Running SPRINT

First, make sure your virtual environment is activated, and then navigate to `./scripts/` under the main SPRINT folder in your terminal.

4.2.1 Single Simulation

SPRINT supports various options for solving. The general format is the following:

```
./run_const_sim.sh --use [CASE] [OPTIONS]
```

The tag `CASE` specifies which input case SPRINT should run. `CASE` should be the name of the folder under `./inputs/cases` (`orig_circinus_zhou` as an example).

The tag `OPTIONS` specify parameters for SPRINT. The options are below:

- `ground_sim`–Starts the ground simulation for the separated simulation. Waits until specified number of satellites have joined before starting simulation
- `satellite`–Starts the satellite simulation for the separated SPRINT. Requests to join the SPRINT simulation at the given ground server's address.
- `rem_gp`–Starts standalone Global Planner server in background before launching simulation. This is used ONLY in the single simulation.
- `F_all`–Forces all modules to be recomputed instead of using previously computed input files
- `F_prop`–Forces only the propagation module to be recomputed
- `F_link`–Forces only the link module to be recomputed
- `fromPkl`–Skips the Orbit Propagation Phase
- `ipdb`–Starts all modules with IPDB

4.2.2 Separated

The terminal command has the same overall structure as the single simulation. However, there is a specific order in starting separated SPRINT:

1. Start separate terminal sessions (one for the ground station network and one for each satellite), each with an activated virtual environment and all navigated to `./scripts`. These terminal sessions can "live" on one computer, or be from multiple Raspberry Pis with a computer.
2. Start the ground station with the command
   `./run_const_sim.sh --use [CASE] --ground`
3. Start each satellite with the command
   `./run_const_sim.sh --use [CASE] --satellite`

The separated simulation should then run. As a note, the satellites and ground can be started in *any* order. Each satellite will try to reconnect to the ground server 100 times, once every second. Once all satellites (the number specified in `./inputs`) have connected to the ground server, the simulation will start.

5 How SPRINT Works

5.1 General Simulation

All simulations have three overall component types:

1. The modeled Ground Station Network, which holds the Global Planner, Ground Stations (and their locations), planned Ground Station outages, and uplink time delays.
2. The modeled Satellites, each of which houses a Local Planner. These models include power usage, antenna half beam power width (HBPW), and crosslink parameters like estimated slewing time. SPRINT assumes that solar energy is collected at a constant rate in non-eclipse.
3. The Simulator, which propagates satellite orbits, models all links, and orchestrates SPRINT for each time step.

5.1.1 Global Planner

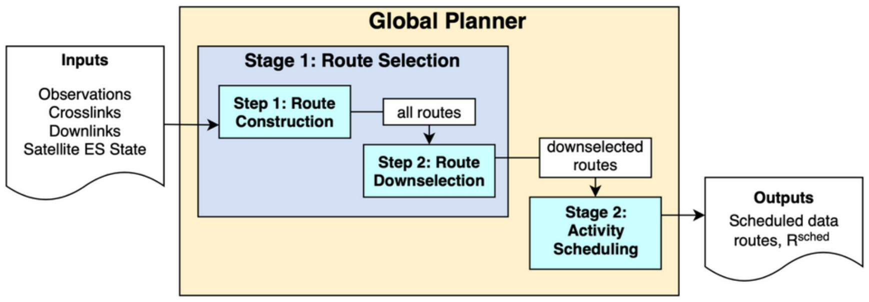


Figure 5: A block diagram of the Global Planner run by the ground station network. The global planner accounts for all planned observations, crosslinks, downlinks, and satellite states to generate an overall schedule for the constellation [Paper 1]

The Global Planner (GP) in Figure 5 is the ground station network's centralized planner and scheduler. It first selects data routes, which are the potential paths (consisting of activities like crosslinks, downlinks, and observations) that data may travel from collection to its final destination. These routes are filtered, or downselected, by prioritizing high data volume, low latency, and low overlap with other routes. After finalizing routes, the GP constructs a MILP to schedule activities. It maximizes data volume and energy margin, minimizes latency, and applies constraints related to energy storage, data storage, activity transition times.

5.1.2 Local Planner

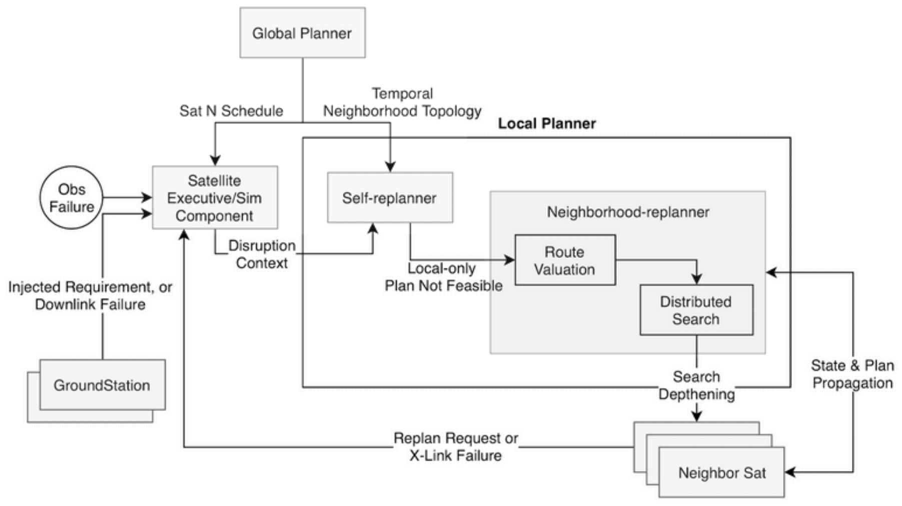


Figure 6: The block diagram for the Local Planner, a feature unique to SPRINT. The Local Planner incorporates the global plan along with any disruptions to create and propagate a new, local plan [Paper 2].

The Local Planner (LP) in Figure 6 runs on each satellite. It uses the global plan as its basis, replanning as necessary with a MILP to account for failures (such as observation failures), disruptions (such as a ground station being unavailable), and unplanned, high-priority "injected" observations. Whenever possible, it shares its new, local plan to other satellites.

5.1.3 Simulator Steps

When SPRINT starts, the Simulator propagates the satellite orbits and then calculates all possible downlink, uplink, and crosslink opportunities. Crosslinks are deemed feasible if the receiver and transmitter share a line of sight, and uplink and downlink opportunities can be limited to specified elevation angle cutoffs.

For each time step, the simulation runs the following steps:

1. Execute actions for satellites and ground stations.
    - For satellites, these can include observations (both injected and planned), or bulk data transfers (referred to as BDTs) as crosslinks or downlinks. As a note, ground stations in SPRINT do *not* have actions. The code does allow for future additions, though.
2. Update states for satellites and the ground station network. This may also involve running the Local or Global Planners (which run MILP solvers),

respectively. How these planners run are outlined in Sections 5.1.1 and 5.1.2.

3. Propagate satellite state to other accessible satellites.
4. Ground Stations share information amongst each other.
5. Propagate ground stations' global plan to accessible satellites
6. Satellites share plans with other satellites, if applicable. This feature can be turned on or off in `./inputs`.
7. Satellites share new local plan if they have run their Local Planner
8. Update simulation time

5.2 Separated Simulation: A Special Case

The Separated Simulation is a variation of the general simulation described in Section 5.1.3. Instead of SPRINT running on a single computer or process, the Separated Simulation can run such that satellites "live" on their own hardware or separate processes (with their own planners) and the ground station network lives on its own processor.

For clarity, we will refer to satellites and the ground station network in the Separated Simulation as *removed* satellites (RSAT) and the *removed* ground station network (RGSN).

5.2.1 Removed Satellite and Removed Ground Station Network Structure

Given the highly concurrent nature of this simulation, the RSATs are structured differently from the original satellites.

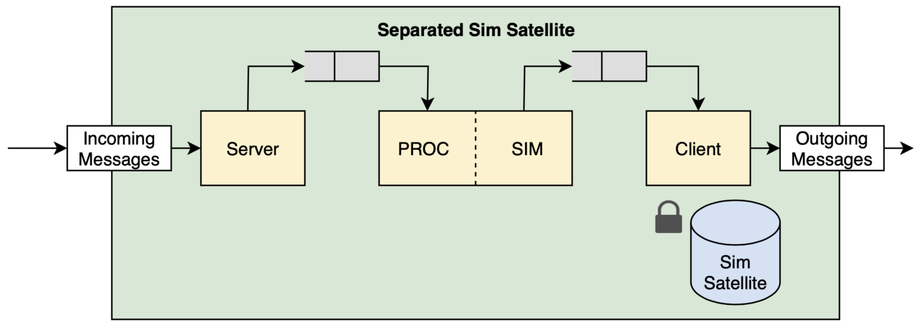


Figure 7: The structure of the Removed Satellite. The SimSatellite is the original Satellite object used in the single simulation, and is protected by a lock.

The RSATs have 5 main parts:

1. **Server** which receives incoming messages and places them on a queue to be processed. By this, the Server is acting as the producer in the producer-consumer pattern.
2. **PROC**, or the **Processor**, which processes the incoming messages from the Server, acting as a consumer in the producer-consumer pattern.
3. **SIM**, or **Simulator**, which runs the satellite simulation steps listed in Section 5.1.3. It often produces messages and passes them to the Client.
4. **Client** which sends outgoing messages that were created by the SIM.
5. **SimSatellite**, the general simulation's original Satellite object. This object models the satellite's data storage and local planner. As the separated simulation is highly concurrent, this SimSatellite is protected by a lock to avoid multithreading issues such as race conditions.

As a note, the Server and Client run on their own separate processors, while the PROC and SIM run on separate threads in a single processor.

The Removed Ground Station Network is structured nearly identically, except that it houses *multiple* Ground Stations from the single simulation. Each Ground Station is protected by its own lock to avoid concurrency issues.

5.2.2 Separated Simulation Steps

The Separated Simulation runs the simulation steps listed in Section 5.1.3 in a distributed manner. Thus, RSATs only run satellite-specific steps, and the RGSN only those for the ground station. Here are the separated simulation steps, along with the initialization process.

As a note, many of these steps refer to messages, denoted in all caps. Descriptions for these messages are covered in Section 5.3.

Initialization

1. RSATs send JOIN to Ground server.
2. RGSN propagates simulation and satellite-specific inputs to RSATs
3. RGSN sends START message to all RSATs
4. RGSN and RSATs enter main simulation loop

Removed Satellite

1. Execute actions, if any. Actions are either observations, downlinks, or crosslinks.

2. Send ACTS_DONE message and wait until all other RSATs and the RGSN have finished executing their actions

3. Update RSAT state. This may involve the Local Planner running, as outlined in Section 5.1.2.

4. Propagate state to other accessible RSATs if haven't done so within a time period (the exact cadence is specified in `./inputs`

5. If specified in `./inputs`, propagate PLAN to other accessible RSATs

6. If the Local Planner has run, send new local plan to accessible ground stations

7. Send FINISHED_PROP and wait for all other RSATs and the RGSN to finish propagating plans and states

8. Send READY_FOR_TIME_UPDATE and wait for all other RSATs and the RGSN to be ready for the next time step

9. Update the simulation time

Removed Ground Station Network

1. All Ground Stations execute actions. Currently, Ground Stations do not do anything; this is a placeholder for potential additions.

2. Wait for all satellites to complete their actions.

3. RGSN updates state. The Global Planner may run here, as specified in Section 5.1.1.

4. Send ACTS_DONE message

5. Ground stations update state

6. If the Global Planner has run, all ground stations exchange information with each other

7. Ground stations propagate global plan to all accessible satellites

8. Send FINISHED_PROP message

9. Wait for all satellites to finish propagating local plans

10. Send READY_FOR_TIME_UPDATE message

11. Wait for all satellites to be ready for next time step

12. Update time

5.3 Messages

In both simulations, satellites and ground stations send messages for communication. These messages are essential for propagating current states, plans, and gathering information for planning.

Messages are generally grouped into two main categories: Data and ACK (short for Acknowledgement) messages. Data Messages hold new, novel data such as plans or states. ACK messages are sent as acknowledgment of having received a Data Message.

As a note, Data Messages sometimes "piggyback" off of ACK messages in cases where an exchange of information between satellites is needed.

5.3.1 General Message Structure

Messages are dictionaries. They are serialized using pickle for transmission. Sections 5.3.1 and 5.3.1 describe the general keys associated with these messages.

Data Messages

- "`req_type`": A String describing the message type
- "`payload`": The information to send, if any. Typically, the information is a dictionary.
- "`dest`": The name of the destination as a String. If a ground station, this is the ground station ID (i.e. GS#), but if it is to the simulation itself, this is 'ground'. Else if sending to a satellite, use satellite's ID (S#)
- "`sender`": The ID of the sender (GS#, S#, or ground) as a String
- "`id`": An integer used as the message ID for the sender's record keeping. This value is unique *within* the sender's records.
- "`waitForReply`": True if the host should wait for an ACK for this message before proceeding further, False otherwise. This field is only used in the separated simulation.

ACK Messages

- "`ACK`": True if no errors in transmission or processing
- "`payload`": Usually empty, but may be an entirely other message that is "piggybacking" off the ACK
- "`dest`": The String destination ID of the ACK
- "`sender`": The sender of the ACK
- "`id`": The integer ID of the received data message. This field specifies which message the host is acknowledging.

5.3.2 General Simulation Messages

Here are the simulation messages sent by satellites and ground stations, regardless of the simulation type.

- "**PLAN**" communicates the global or local plan of the satellites and ground stations.
- "**STATES**" holds the states of the satellites and ground stations. Receivers send a simple ACK back.
- "**BDT**" is the Bulk Data Transfer. Unlike all these other messages above, this one is sent in explicit stages.

    1. The transmitter sends the first part of BDT. It waits for the ACK from the receiver *and* the amount of the BDT that was actually received.
    2. Then, the transmitter and receiver exchange PLAN messages, with the receiver's PLAN message piggybacking off the ACK for the transmitter's PLAN message. This process repeats until the receiver sends that it cannot process anymore data, or when there is no data left to send in this time stamp.

5.3.3 Separated Simulation Messages

Along with the messages listed in Section 5.3.2, the separated simulation has the following messages to ensure the RSATs and RGSN are on the same steps. How these messages are used is shown in Section 5.2.2.

Initialization Messages These messages are used to set up the simulation.

- "**JOIN**": Sent by a RSAT to the Ground's Server to join the simulation
- "**INIT_PARAMS**": Sent by the Ground to each newly joined RSAT. These parameters are simulation and satellite-specific.
- "**ALL_IPS**": Sent by Ground to each RSAT once all RSATs have joined the simulation. Includes all IP and port addresses to all RSAT Servers.
- "**INJECT_OBS**": Ground Sim sends to each satellite. Used to create disruptions specified by the simulation inputs.
- "**SAT_WINDOWS_INIT**": Ground Sim sends to each RSAT information for their window initialization. This is done immediately before the simulation starts.
- "**START**": Ground Sim sends to all RSATs to start main simulation loop

Simulation Messages These messages are used to keep RSATs and the RGSN in the same phase of the same time step.

- "**ACTS_DONE**": Sent by RSATs and Ground when their assigned actions are completed for a given time step. By this, Ground and RSATs "wait" for all others to finish their actions before continuing the simulation. (This can take a few seconds, as this is where the Global Planner often runs)
- "**FINISHED_PROP**": Sent by RSATs and Ground after finishing propagating any PLAN (especially if the Global Planner or a Local Planner has run) and STATES messages. This does NOT mean they are done *processing* any incoming messages.
- "**READY_FOR_TIME_UPDATE**": Sent by RSATs and Ground when all components are ready to continue to the next time step (i.e. all have sent FINISHED_PROP. The assumption is that if all RSATs/Ground are done propagating information, then all have also finished processing any incoming information from their peers, as the Client component of the transmitter can only send FINISHED_PROP after receiving an ACK from all recipients for their propagated STATE and PLAN messages). All RSATs and Ground wait until *all* have sent this message before moving to the next time step.