Optimization of SmallSat Constellations and Low Cost Hardware to Utilize Onboard Planning

Optimization of SmallSat Constellations and Low Cost Hardware to Utilize Onboard Planning

Mary Dahl*, Juliana Chew,† and Kerri Cahoy‡
Massachusetts Institute of Technology, Cambridge, MA, 02139, USA

The Scheduling Planning Routing Inter-satellite Network Tool, or SPRINT, is an open-source software tool that is currently under development at MIT's Space Telecommunications, Astronomy, and Radiation (STAR) Laboratory. It schedules, plans, and re-plans the activities of large Earth-observing satellite constellations under changing conditions. The SPRINT framework seeks to maximize observation data, minimize latency, and autonomously adapt to unexpected events. This is done by utilizing crosslink capabilities on the satellites, a master ground planner, and onboard local planners on each satellite. All planners run multi-integer linear problem (MILP) solvers. SPRINT is distinct from other satellite scheduling tools by utilizing self-replanners on the individual satellites: an unexpected addition to the plan, or an "injected observation," can be commanded by an operator. That satellite will then calculate a new plan for itself and the satellites around it, which will be propagated through the network.

A tool such as SPRINT becomes increasingly important as constellation sizes increase and they become infeasible for operators to plan by hand. However, not every constellation architecture is well suited for observation and data management using crosslinks and replanning. For example, some have orbital configurations that make it difficult to support crosslinks for long enough to transfer a useful amount of data, and some constellation architectures have so many ground stations and overpasses that data downlink becomes trivial. These additional ground stations may become an unnecessary expense when SPRINT and crosslinks are added. We examine the benefits of using SPRINT on example state-of-the-art CubeSat and small satellite constellation missions, such as TROPICS, CYGNSS, and GeoOptics CICERO. We evaluate how modifications to these missions, such as changing the orbital spacing between satellites and the number of ground stations, can lead to benefits in data volume collected and downlinked, latency of data, and overall mission cost. One example finding is that is the addition of the SPRINT framework to the CYGNSS mission (with 8 small satellites) achieves similar data packet latency with one ground station as the base mission achieves with four.

We additionally look at implementing SPRINT on affordable hardware to quantify its feasibility for low-cost small satellite constellations. To this end, we have developed a testbed for implementing the SPRINT platform on multiple Raspberry Pis to examine the speed of performance and processing. The testbed decouples the SPRINT framework to work on separate machines: the ground station network is modelled on a laptop computer, and each satellite is modeled by a Raspberry Pi. All communication between them is done with socket programming. We evaluate the testbed's performance for multiple constellation architectures and ground station network sizes using metrics such as run time and RAM usage and compare this performance to running the simulation on a single computer. We analyze how this performance scales as additional nodes are added to estimate the feasible size of an Earth-observing constellation of small satellites with Raspberry Pi flight computers with access to a small number of ground stations. Additionally, we compare the performance of different commercial and open source solvers on the Raspberry Pi. As Raspberry Pis are ARM-based processors, they can only handle a limited number of MILP solvers; some high end ones, like Gurobi, are incompatible. We explore the solvers CBC and GLPK for Raspberry Pis. We examine the performance of one, two, and three Raspberry Pis used simultaneously.

*Graduate Student, Department of Aeronautics and Astronautics.
†Undergraduate Student, Department of Aeronautics and Astronautics.
‡Professor, Department of Aeronautics and Astronautics, and AIAA Associate Fellow

I. Introduction

As CubeSats [1] and other smallsats become increasingly cost-effective to produce, launch [2], and have demonstrated useful on-orbit capability, from Earth-observation imaging payloads to convolutional neural nets on as small as 1U satellites [3], the demand for constellations of CubeSats with a common, coordinated goal only increases. There have been several small satellite constellation missions of varying platform sizes in recent years, including NASA's CYGNSS [4] mission, which launched in 2016 and consists of eight smallsats, the upcoming NASA-sponsored and MIT Lincoln Laboratory-led TROPICS [5] mission, which consists of six CubeSats, as well as larger constellations sch as GeoOptics CICERO [6], which is planned to consist of twenty-four smallsats, and the Capella X-SAR Constellation [7], which is planned to consist of thirty-six smallsats. On the extreme end, SpaceX's Starlink, in an effort to provide global high-bandwidth communications coverage, is planned to have up to 42,000 smallsats [8].

In particular interest for this work are constellations of Earth observing (EO) satellites and their use in disaster relief. The unpredictable nature of disasters begets the need for dedicated EO missions that can support full coverage of Earth while utilizing few satellites as possible [9], that levy cross-constellation and cross-platform coordination to optimize for rapid revisit and extensive spatial coverage and fast communication relay and data delivery [10]. CubeSat constellations can support many of these goals, as well as helping to keep the costs of manufacturing and launching responsive constellations feasible.

Fig. 1 A coordinated response of a constellation to an unexpected disaster, showcasing the passing of plans and data to rapidly image new data and downlink it. [11]

However, mission costs do not end at getting satellites on-orbit; obtaining meaningful observations and then getting data down is a challenge for many CubeSat teams. Although LEO is the most cost-effective orbit, missions in LEO can also have short lifespans, making it essential to optimize the quantity of data that can be downlinked. Maintaining access to a network of multiple ground stations can be prohibitively expensive for smaller companies or university constellations that need to manage a large number of CubeSats. For instance, the NASA Near Earth Network (NEN) costs $490 per pass [12]. Autonomous operation of satellite data routing can optimize the process, allowing for the same performance to be obtained with fewer ground stations that need to be supported. Missions must carefully balance the capacity to retrieve all their data from orbit and the costs of an extensive ground station network.

The space constellation scheduling problem has been studied in various forms. There have been ground based approaches, such as a web-based coordination planners for different assets [13]. There have also been several examples of planning algorithms on-orbit, such as the CASPER (Continuous Activity Scheduling, Planning, Execution, and Replanning) software system, developed by JPL [14]. It shares the philosophy of self-replanning, called iterative repair, that incrementally makes updates until a schedule is no longer conflicted after anomalous activity. It has successfully been used on-orbit multiple times, including on the Three Corner Sat mission in 2004. A planning algorithm was also run on the COSMO-SkyMed constellation between 2007 and 2010 [15], utilizing a deterministic constructive algorithm. These missions, and others, had the luxury of a larger form factor and greater computational power than CubeSats impose.

One example of planning on CubeSats is a CubeSat-specific implementation of the Automated Scheduling and Planning Environment (ASPEN) [16] scheduling system, able to schedule for a single CubeSat, which was demonstrated on the Intelligent Payload Experiment (IPEX) CubeSat [17] in 2013. There has also been research into scheduling imaging missions for two CubeSat activities [18], utilizing mixed integer linear programming, as well as task scheduling utilizing integer programming, constraining on power and optimizing on task priorities [19]. However, there is a current

gap of focused work on planning on a large-scale specifically for CubeSat operations.

Few commercial options for satellite planning exist, with the most commonly used being the scheduler in Systems Tool Kit [20], which uses heuristic optimization algorithms and resource constraint satisfaction. This scheduler, however, currently lacks the ability to automatically reschedule upon the addition of unexpected observations, does not support open-source cross-linking between different constellation platforms, and does not incorporate modularity to allow for open-source solver module upgrades and substitutions. These capabilities are crucial for disaster relief missions: it may suddenly become necessary to rapidly direct resources to image areas or provide observation support to follow disaster progress and support emergency operations. STK's solver is also not lightweight enough to be ported to and operate on the limited-capability CubeSat flight computers that would be necessary for proliferated sensing constellations with rapid revisit times. It is also clear that cybersecurity and an authentication process would need to be implemented to support emergency coordinated activities among constellations.

Many planning algorithms for data routing also rely the constellation being capable of crosslinking data from satellite to satellite. While crosslinks have been used on conventional satellites for many years, the capability on smallsats and CubeSats is still in its infancy. Few CubeSats have been developed with crosslink capabilities. One example was the EDSN project, which was to demonstrate a swarm of eight low-cost RF crosslink-capable 1.5U satellites, but it was unfortunately lost in launch [21]. Following this mission, there is a followup pair of Nodes satellites [22], which utilized similar technology as EDSN. On the cutting edge, the CLICK mission is an upcoming technology demonstration of both RF and laser communication links on CubeSats [23].

Having identified the needs and challenges facing disaster relief satellites, there is a clear benefit from a tool that is able to plan for coordinated activity across satellite constellations with crosslink capabilities, relying on only a few ground stations, and is robust to unexpected observations.

II. Previous Work

The work in this paper augments and extends previous work on SPRINT [11] [24]. This paper will not go into great detail about how SPRINT functions, but a brief overview will be given. SPRINT can be broken down into three parts: the simulation environment, the global planner (GP) and the local planners (LP). This can be seen in Figure 2.

2.

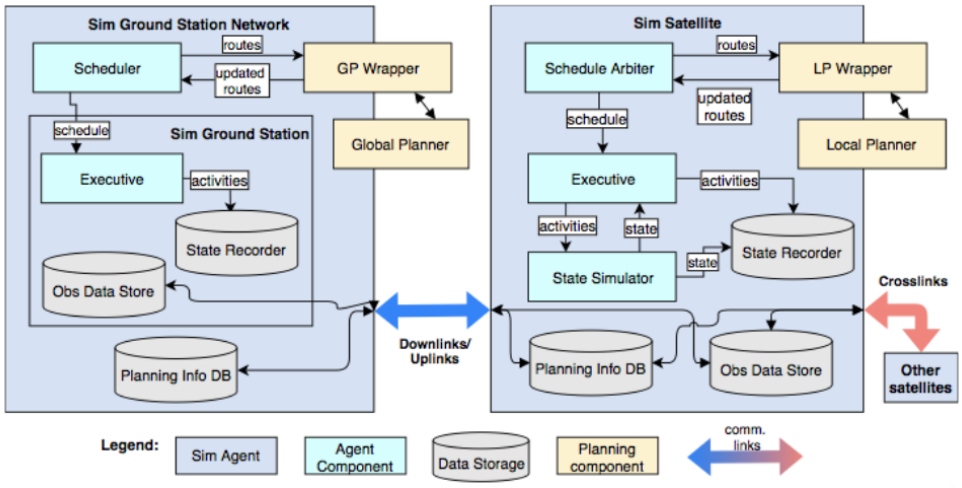

Fig. 2 Software architecture for SPRINT, including the simulation and the parts of the scheduler. Note the way that plans are passed between virtual downlinks, uplinks, and crosslinks, as well as how data storage is simulated. [11]

Put simply, the simulation environment calculates orbit propagation, downlink/uplink windows, crosslink windows, and observation times. The GP then constructs and solves a MILP, creating an initial plan for the constellation. The simulation follows this plan, until an interruption to a satellite or to a ground station is supplied by the user. This initiates the LP on the interrupted satellite. The LP solves for the interruption and propagates the new plan across the network. More detail will be given on each of these, in subsections II.A, II.B, and II.C.

A. Simulation Environment

Although only the GP and the LP would be necessary for hardware implementation (see section VI), SPRINT's simulation environment is integral for studying the benefits of the algorithms. A user provides the following models for the mission:

1) Constellation configuration: The constellation parameters. SPRINT supports both Walker constellations and those specified with Kepler orbital parameters.
2) Operational profile: Specific parameters to the case at hand, including the latitude and longitude of every observational target and and time periods where downlinks or crosslinks would be unavailable.
3) Simulation case configuration: The start and end times of the simulation, schedule disruptions for ground stations, and injected observations for individual satellites. These are able to be manually or automatically generated.
4) Ground station network configuration: the ground station network, including the locations and time delays for uplinking.
5) Satellite model: Models of satellites, including the power usage, half power beam width (HPBW) of the antennas, and additional crosslink parameters, such as the estimated time for slewing. Power is modeled with an idle consumption, with an additional pull from tasks (imaging, transmitting, etc). It is currently assumed that solar energy is collected at a constant rate in non-eclipse.

Given these values, the simulation begins by propagating the satellite constellation using the open source poliastro Python library [25]. The accesses are next calculated for downlinks, uplinks, and crosslinks. It is assumed that a crosslink is available as long as the satellites have line of sight to each other, or that the vector between the satellites passes roughly above the surface of the Earth. The simulation does not currently model the power and time needed to slew for communications, but it does allow for estimated values to be implemented to provide constraints. Downlink/uplink windows are calculated at any time the satellite passes over the ground station with a user-specified elevation cutoff.

The simulation, after running the GP, then simulates the actions of all the members of the mission. Data is virtually passed from satellite to satellite and from satellite to ground.

B. Global Planner

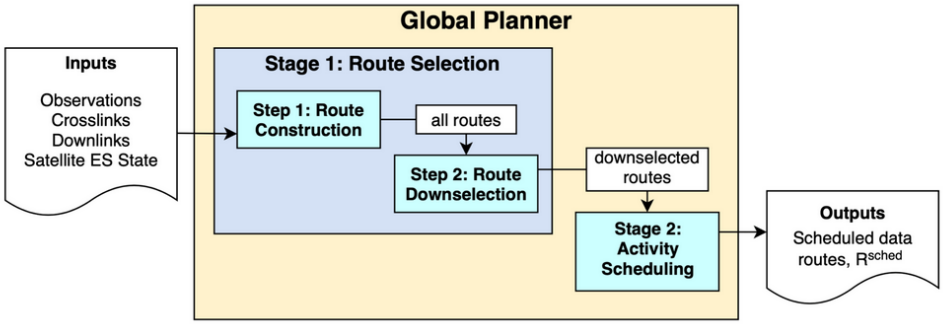

Fig. 3 Global planner inputs, components, and outputs. Note the route construction and downselection stages, which aids in keeping the GP runtime low. Solving for the downselected routes prevents running the solver on infeasible solutions. [11]

The GP is a centralized, ground-based planning and scheduling (P&S) algorithm, utilizing a single, integrated MILP scheduling solution. There are two stages to the GP: route selection, which creates data routes, and activity scheduling, which de-conflicts and schedules routes to meet constraints. Data routes are pre-set allocations of bandwidth throughout the constellation: the potential path any chunk of data might take from being an observation to its final destination. Data routes are constructed from activity windows, or slices of time where any activity (observe, crosslink, or downlink) can be performed. Each data route is given a volume capacity, representing the utilization of the potential data volume throughput.

These data routes are first constructed by the GP and then immediately downselected. Data routes that contain redundant crosslinks (passing between the same satellites), low data volume, or high latency, are pruned. The routes are then chosen using three heuristics:

1) Highest data volume
2) Lowest latency
3) Least overlap between routes

After these are performed, the model is written into a MILP. The decision variables for this MILP focus on maximizing utility for activities and data routes, ensuring that data routes that have sufficient data volume for downlinking are chosen, and minimizing latency for an observation. The constraints for this problem include constraining data volume for routes passing through activities to be less than the throughput of that activity, enforcing transition times between activities (slewing, downlinking), energy storage, and data storage timing. Finally, the objectives for the global planner maximize data volume and energy margin and minimize latency.

C. Local Planners

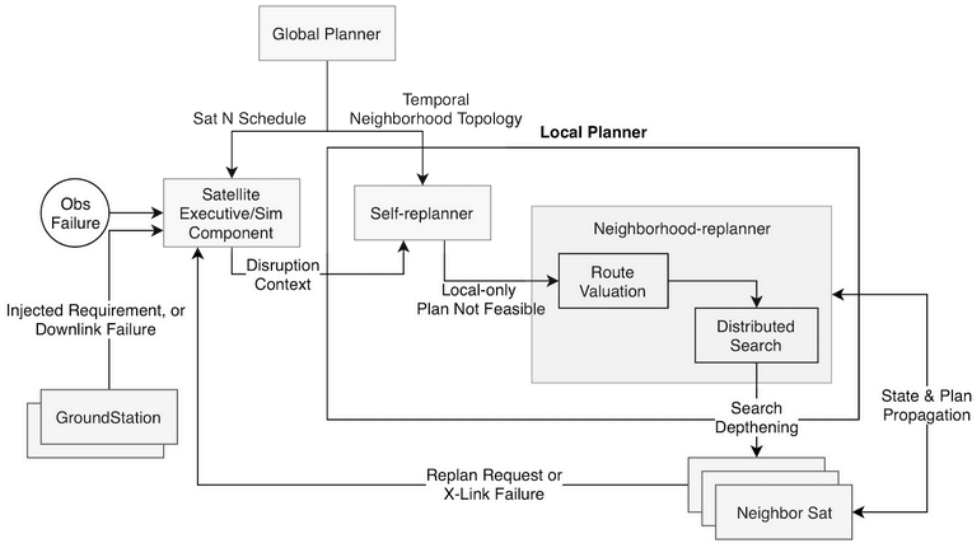

Fig. 4 LP block diagram, depicting the relationship between different modules in SPRINT. Unique to SPRINT, the LP incorporates the GP plan along with failures and injected observations to generate and propagate a new plan. [24]

The local planners on each satellite essentially break all data flows coming into a satellite into its constituent parts. When an unexpected observation or a schedule disruption (such as a ground station being inoperable) is communicated to the satellite, it looks at all of the data inflows and outflows (pre-planned, and unexpected). It then constructs and runs a MILP with onboard solvers. Its decision variables focus on data volume allocation. The constraints restrict this allocation based on the throughput that was scheduled. The objective function maximizes data volume allocations for existing and injected data routes and minimizes latency for injected observations.

III. Methodology

A. Case Definition

For this study, we focused on applying SPRINT to specific missions that have been or will be implemented on orbit with constellations of sizes less than 30 satellites. The cases that have been chosen came from a variety of recent or planned small satellite missions with Earth-observation targets that had publicly available orbit and satellite information. These case of interest are defined in Table 1. More detail about the precise assumptions that were made for the orbital parameters can be found in the Appendix.

Table 1 Parameters of Baseline Constellations

	TROPICS	CYGNSS	GeoOptics CICERO
Number of Satellites	6	8	24
Satellite Sizes	3U	SmallSat	6U
Number of Orbital Planes	3	1	10
Number of Ground Stations	3	4	11
Number of Observation Targets	20	20	20

We additionally examine modifications made to these cases to better showcase the benefits of crosslinks and schedule planning; namely, requiring less ground stations to receive all of the data. The summary of these adjustments can be seen in Table 2. For all but GeoOptics CICERO, we have decreased the ground stations to 1. Empirical testing showed that CICERO produced low data throughput for any less than 7 ground stations. The ground stations chosen are based on publicly available information for the satellite networks being used for these missions, and the modified ones chose the ground station(s) that is most centered on the constellation (see Section IV).

Note that the base case of TROPICS, with 3 planes and 2 satellites per plane, cannot support crosslinks under the assumptions we have made for orbital spacing. To this end, we examine versions of TROPICS with differing orbital configurations, as summarized in Table 3. The additional planes were made at the same altitude (60 km), eccentricity (0), and inclination (30 deg) as the existing planes and incremented the RAAN such that each increment would be $\frac{360}{number of\ planes}$. The TROPICS case in the ground station modification study is one of these modifications, with 3 planes and 3 satellites per plane.

Table 2 Parameters of Modified Constellations for Ground Station Analysis

	TROPICS	CYGNSS	GeoOptics CICERO
Number of Satellites	9	8	24
Satellite Sizes	3U	SmallSat	6U
Number of Orbital Planes	3	1	10
Number of Ground Stations	1	1	7
Number of Observation Targets	20	20	20

Table 3 Parameters of Modified TROPICS Constellations

	Case 1	Case 2	Case 3	Case 4
Number of Orbital Planes	2	3	3	3
Number of Satellites per Plane	2	2	3	4

Because none of these constellations utilize crosslinks, assumptions had to be made for the crosslink hardware. It is assumed that they all use RF crosslink technology. Additionally, the observation targets for each constellation were not based on specific locations, but were generated over the latitude and longitude of the constellation. Each case has 20 of these observation targets.

B. Test Parameters

To study these cases, each was modeled in SPRINT. All cases were run a total of 20 times: 10 with the local planner and crosslinks on, and 10 with them both off. This was done to simulate operation of the constellation with only a ground based planner or an operator. Further study may be done in the future to compare performance to commercially available solvers.

Between each run of the test, the only modifications were the injected observations. For each case, we had 2 injected observations per satellite per 24 hours. This may be extreme for what would be seen on orbit, but we wanted to simulate a day of rapid activity on the ground that would most benefit from a modular observation satellite constellation, such as

an active forest fire. These injected observations were randomly generated throughout the day for each satellite, with restrictions to ensure that none would overlap on a single satellite. The results of all 10 tests were then averaged together.

IV. Crosslink Cost Benefit

If the goal of a constellation is to minimize latency to the ground, rather than implementing crosslinks and planning, one could simply add more ground stations for downlinking data, such as utilizing the KSAT Lite ground station network. While this would decrease latency (to a point), we argue that crosslinks and planning would be both more mission effective and more cost effective in the long term. This topic has been studied more extensively by others [26], but even a simple analysis such as this shows immediate promise to the concept.

Adding ground stations is certainly the less technically complex option, as adding crosslinks requires additional communication hardware, backups for those, additional pointing and slewing capabilities, and the technical know-how to implement it. However, we argue several reasons why more ground stations is undesirable for constellation operation. First, ground stations are not always reliable. If a network is relying on all of its nodes to ensure low-latency data and one of them goes down, the performance of the constellation will suffer. Without an automated planner, an operator must decide where to route data that can no longer downlink to a particular ground station. In some instances, the data that are stored on that satellite may not be able to be downlinked for a day or more, causing the satellite to miss on important observations. This causes a tension: more ground stations decrease latency and increases the likelihood data will be able to downlink, but it increases the strain on the operators to determine viable data routes. More than anything, however, more ground stations incurs additional cost.

Utilizing ground station networks has significant monetary cost that is recurring for every pass. As we are focusing on smaller satellite missions, it may not be feasible for these programs to spend an excess of money on unnecessary ground stations to meet their latency requirements. The cost of most ground station networks is not public information and is negotiated between customers, and thus an exact number could not be made for this paper. We will assume that the cost of a pass is $100. Most networks require additional setup costs in the range of $30,000, as well as monthly staffing and support costs, assuming and additional $200 per month. These numbers can be treated as a parameter to be varied and a sensitivity analysis will be discussed in future work.

Quantifying the inclusion of crosslinks on a satellite is difficult, as much of it depends on the exact implementation for individual missions. There are two options for crosslink technology: RF and laser. RF based technology is cheaper and simpler to implement, both due to the legacy of the technology, meaning it is easier to find or train individuals with the skill necessary to implement it, as well as the less stringent pointing requirements. The main financial cost is a dedicated antenna, which is typically in the realm of $6,000 for a small satellite, and a radio, which we estimate the cost of $6,000. Laser communication, while offering better data rates, requires significant pointing technology in order to operate, driving requirements and price to match. As this technology has limited legacy on small satellite missions, we will not consider laser communications in this paper.

The computational hardware required to run planning is another concern with providing onboard planning. This is certainly true with heavier-weight planning algorithms, but as seen in section VI, we have shown adequate performance of the SPRINT software on Raspberry Pi computers, which are lightweight and low enough cost that the majority of programs can add them to their payload with little effort.

We assume that a single ground station would have approximately 3 overpasses per day.

Assuming a 20 satellite constellation, the results of this cost trade analysis can be seen in Figure 5. Although this is a simple calculation, it still shows valuable analysis that automation of constellation operation pays off in approximately 130 days of operation.

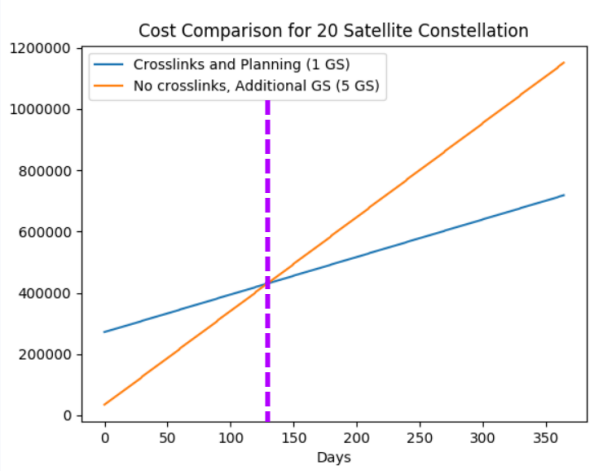

Fig. 5 Results from a simple cost over time analysis. Implementing crosslinks and planning becomes cost effective after 130 days.

V. Results

Figure 6 shows the results of the ground station modification study, and Figure 8 shows the results of the orbital parameter modification study. In general, we can see that adding crosslinks and planning decreases latency in all situations, maintains this decreased latency in situations with less ground stations, and maintains the data throughput.

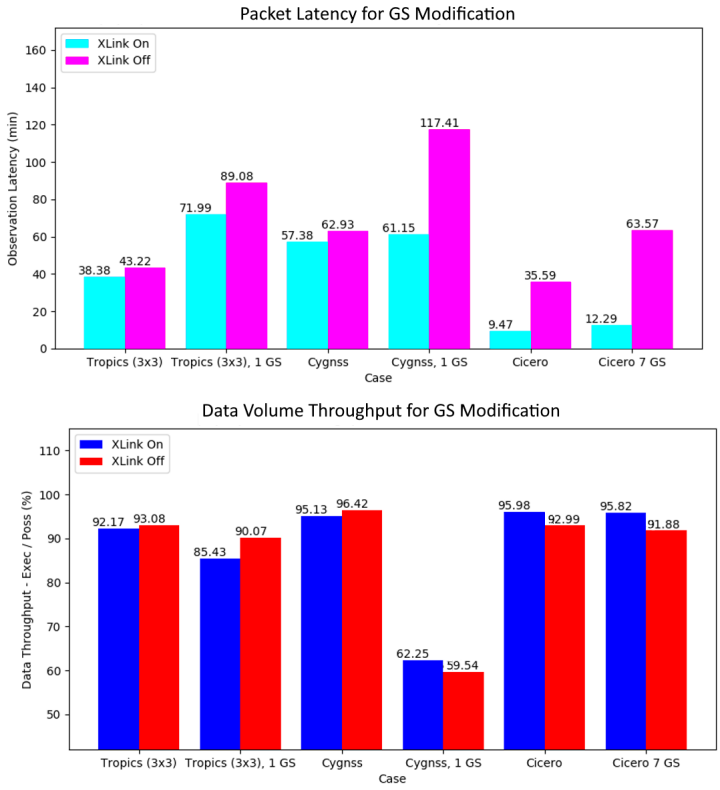

Fig. 6 Results from the Ground Station modification study. We see increased performance in latency when and steady performance in data throughput when utilizing crosslinks. This increased latency performance is maintained even when ground stations are removed.

We can see several things from the ground station modification study. Considering first the performance of the constellations with less ground stations and without crosslinks and replanning, we see a severe increase in latency, but generally consisten throughput. This implies that getting the data that was collected down was still fully possible, but that the time it would take would be increased. The exception is CYGNSS, which had its data throughput cut almost in half when it was assigned only one ground station, both with and without crosslinks. This is due to the "clumped" nature of the CYGNSS constellation; because all of the satellites are right next to each other, crosslinks are easy to perform, hence the adequate performance in latency with SPRINT on, but when an injected observation prevents one satellite from downlinking, it is likely none of the rest would be able to in time either. The data must wait until the next time the constellation is over that ground station, possibly deleting it due to new observations, and overall decreasing throughput. The orbital configuration causing this effect can be seen in Figure 7.

Looking at the effect of SPRINT on data throughput, we can see, generally, the throughput is consistent with and without SPRINT on. In the smaller constellations, there is a bit of a loss, which is in part due to the fact it takes time for running the planning algorithm and performing the slewing, causing it to miss the window to pass the data. For larger constellation, however, we see the throughput actually increase, as there are more options for routing data, freeing up satellites to date additional observations.

Packet latency, however, is where the benefits of SPRINT can be most clearly seen. With it on, we see a improvement in the latency for both the base cases and cases with fewer ground stations. The base cases vary in improvement from 5 minutes to 26 minutes. This time saved is may not be worth the implementation costs on its own, but the similarly performing low latency on the same cases with fewer ground station is; despite the decrease in ground stations, the performance approaches the performance of the base cases in most circumstances. This benefit increases as the number of satellites increases, as expected: the more options there are for data routing, the more likely an optimal path can be found. Even TROPICS, which has the fewest crosslink opportunities, shows a 17 minute improvement in latency in the one ground station case.

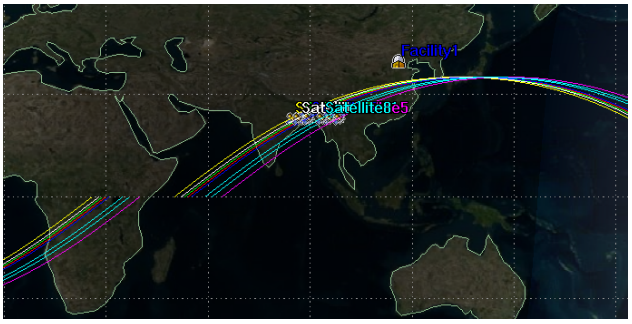

Fig. 7 The mission design of the single ground station CYGNSS case, as shown in STK. Note that all the satellites would reach the ground station around the same time.

The varying satellite configuration study shows similar results. First, we see that both cases with only two satellites per plane cannot perform crosslinks and thus get no significant benefit from utilizing SPRINT. The cases that can utilize crosslinks do see benefit. As with the previous examples, the more opportunities there are for crosslinks, the more benefit there is to the latency and the throughput.

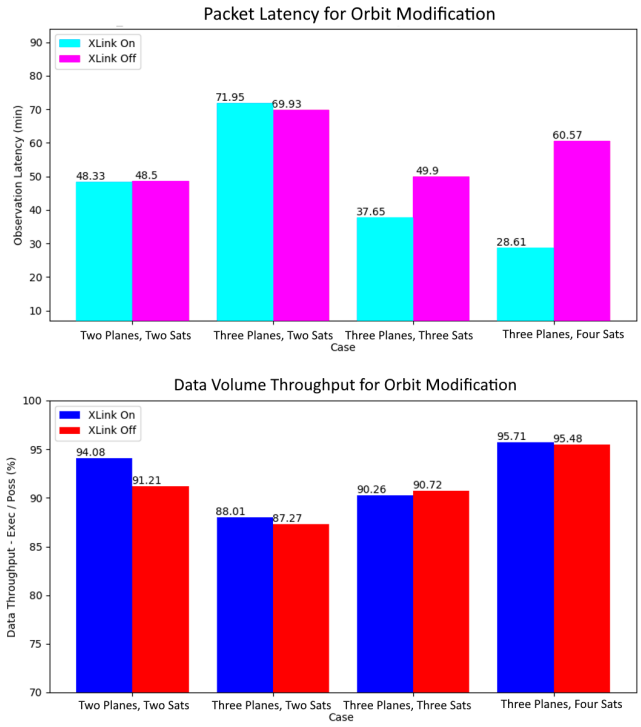

Fig. 8 Results from the TROPICS constellation architecture modification study. We can see that the more opportunities for crosslinks, the lower the latency.

Overall, we prove that SPRINT is effective at improving the performance of many types of missions. We additionally find that SPRINT most benefits missions with many opportunities for crosslinking as well as ones that have ground station(s) that are visited by the nodes with enough time to allow for crosslinks to replan data routing.

VI. Hardware Implementation

To demonstrate the functionality of SPRINT on low cost flight hardware, we adapted SPRINT to run on Raspberry Pis. This is both to prepare for an on orbit demonstration mission where Raspberry Pis could serve as the flight computer, as well as a proof of concept for a concurrent software environment. In the following section, we refer to this adapted SPRINT as the "separated" simulation.

The first step for adapting SPRINT was finding a suitable MIP solver for the Raspberry Pis; our default solver, Gurobi [27], is not compatible with the ARM-based Raspberry Pis. We conducted a trade study between two open-source MIP solvers, CBC [28] and GLPK [29]. For the trade study, we ran the entire SPRINT simulation on one Raspberry Pi for varying constellation sizes. As a reference, we also ran SPRINT on a Macbook Pro using the Gurobi solver.

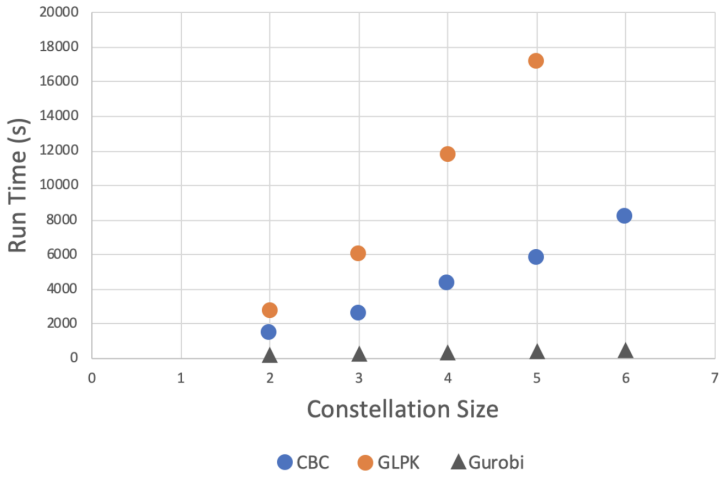

Fig. 9 Computational run times of two MIP solvers run on a Raspberry Pi compared to a Macbook Pro running Gurobi. CBC is a feasible solution for the Pi.

As shown in Figure 9, CBC outperforms GLPK in all cases. While CBC scales linearly with more satellites, GLPK's run times increased exponentially, to the point that the 6 satellite case was terminated after over six hours of run time. As SPRINT would be implemented on constellations of far more than six, CBC is the feasible option. We developed the separated simulation to utilize CBC for the LP on the satellites and the default Gurobi solver on a Macbook Pro for the ground station network.

A. The Separated Simulation

The separated simulation has two main components: the satellites, each run on a separate Raspberry Pi and the ground station network, run on a Macbook Pro. As stated earlier, each satellite has its own LP and each ground station has its own GP. We model message passing using socket programming. Figure 10 displays the structure of the separated simulation's satellite.

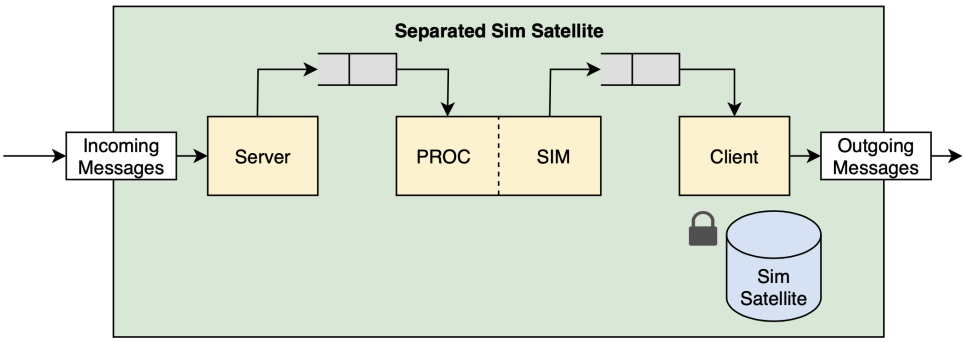

Fig. 10 Diagram of the separated simulation's satellite structure. The SimSatellite is the Satellite object in the original simulation managed with a lock.

In essence, the separated simulation's ground station network and satellites are wrapper classes managing I/O processing and the original simulation's ground station network and satellites, respectively. In Figure 10, for instance, the Sim Satellite is the same Sim Satellite from Figure 2. This Sim Satellite object simulates a satellite and stores and ingests data and tasks. It also houses the LP.

The four new main components are the Server, PROC, SIM, and Client. The Server receives incoming messages,

and the Client sends them. The PROC ("Processor") runs continuously in the background and consumes messages passed from the Server. While PROC runs, SIM ("Simulator") runs the main simulation loop, decides when to send messages, and passes these messages to the Client. The Server, PROC/SIM, and Client each run on their own processes, where PROC and SIM run on the same process but in separate threads.

As a note, the structure of the Separated Sim Ground Station Network is nearly identical to Figure 10; the only difference is that the new Ground Station Network holds multiple Sim Ground Station objects, each guarded with a lock. By adding locks, we aim to avoid deadlocks, race conditions, and other common problems in concurrent algorithms.

B. Performance Comparison

To test the separated simulation, we compared its performance to the nonseparated version (denoted as "PC"), which was run on a Macbook Pro with 32GB of RAM. The test cases found in Table 4 were used.

Table 4 Test cases to compare the separated SPRINT to the PC version. Each test case ran for 24 hours of simulation time at 10 second intervals.

Test Case	Satellites	Ground Stations	Observation Targets	Scheduled Disruptions
1	1	1	4	1
2	2	4	4	1
3	3	4	4	1

Compared to the "PC" case, the separated simulation has similar packet latency trends, as seen in Figure 11. We conclude that the separated simulation performs comparatively to the PC version; the average latency values are identical across simulations. Enabling crosslinks, as expected, lowers packet latency, as described in Section IV.

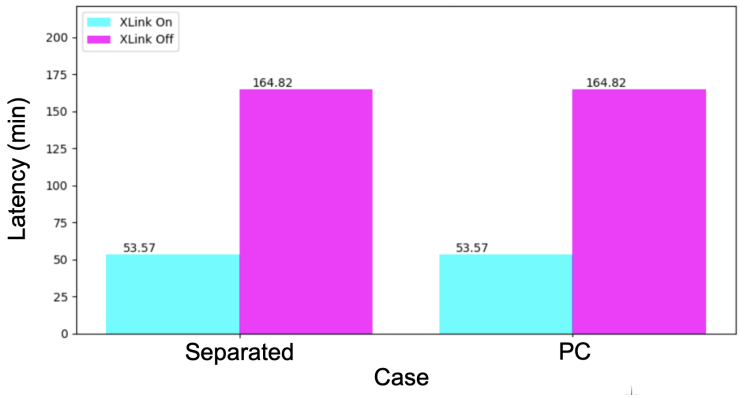

Fig. 11 Average packet latency in two satellite case for PC and separated simulation. We can see that the performance is identical.

Along with latency results, computation run time is essential for evaluating the decoupled simulation. We examine the run time for varying numbers of satellites in three different variations: the original simulation, the separated PC version, and the separated Pi version. In the separated PC version, we ran the entirety of the separated simulation on a Macbook Pro, whereas in the separated Pi version, we used Raspberry Pis for satellites. Raspberry Pis have a less computational power, so we expect the run time of the Pi version to be longer.

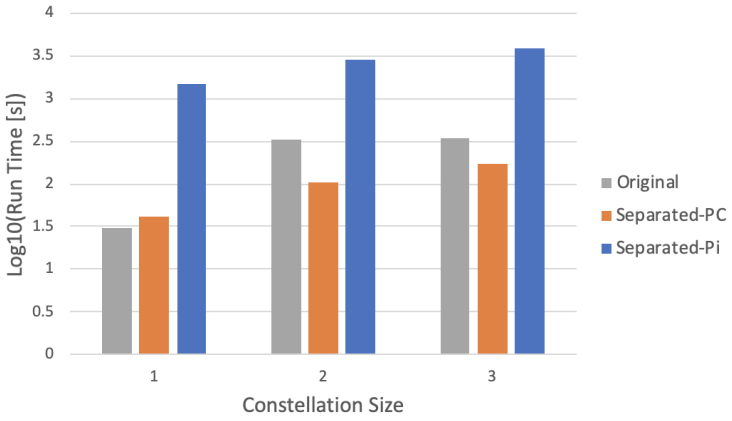

Fig. 12 Computational run times for 1, 2, and 3 satellites in the original, separated PC simulation, and separated-Pi simulations. The separated-PC simulation runs the fastest, while the separated-Pi simulation suffers from low network speed. The exact values from this graph are located in the Appendix.

Indeed, as shown in Figure 12, the run time increases as the constellation size increases. For the separated PC runs, the run time was slightly higher for one satellite but decreased with larger constellations, implying that concurrency adds roughly a constant amount of time for initialization and message passing. With larger constellations, the benefits of distributing the workload across processors increases, lowering the run times; the run time for two satellites was 358 seconds in the original simulation, but dropped to 103 seconds when separated.

However, the run times for the separated simulation with the Raspberry Pis are much longer than expected, with the two satellite case taking 2870 seconds. These slower run times are likely due to the speed of the network; with slower WiFi, message propagation via socket programming slows down. We plan to utilize radios for future testing to both control for network speed and to more realistically simulate satellite communications. Despite these drawbacks, we consider the time taken to run the simulations within the bounds of real-time information passing.

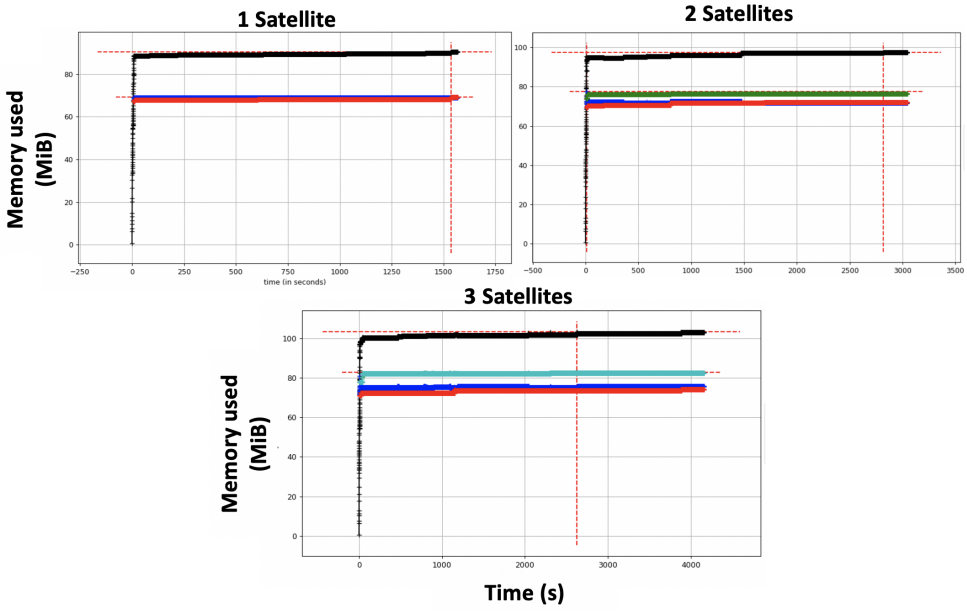

Fig. 13 The memory usage of the Raspberry Pis during the separated SPRINT simulations, showing it is well within the capabilities of the Pi.

We also measured the memory usage in the Raspberry Pis for all cases in Figure 13. As expected, as the constellation

size increases, the memory usage slightly increases, as in our implementation, larger constellations cause more processes to be spawned. However, processes are usually dormant and draw minimal memory, so the overall cost is small. Considering Raspberry Pis typically have at least 700MiB memory available, the memory draw demonstrated is feasible for on-orbit use.

It is also important to note that the run times and memory usages shown above are strict upper bounds for on orbit activity. Currently, both the Raspberry Pis and the laptop are running the simulation to determine which actions to perform in which order. This simulation adds extra overhead for the system that satellites in orbit would not have.

VII. Conclusions and Future Work

In this paper, we have demonstrated the performance and effectiveness of the SPRINT software for improving constellation efficacy when presented with schedule disruptions on a variety of mission architectures, showcasing its utility in decreasing the number of ground stations necessary to perform a mission and thus decreasing the costs of the mission. We have shown the types of cases that most benefit from crosslinks and planning: constellations that simultaneously have a high density of crosslinks as well as a centralized ground station network. We have shown by designing a constellation with these factors in mind, less ground stations can be utilized for similar throughput and improved latency. We have also demonstrated the feasibility of SPRINT in flight hardware by decoupling the simulation and running the satellites on separate Raspberry Pis.

For future work, we plan to continue to take steps towards flight readiness. To this end, we seek to continue to optimize SPRINT's performance on Raspberry Pis and potentially other pieces of flight hardware. We additionally will replace the socket programming with radios to remove the reliance on WiFi. We also wish to continue to scale up our testing with additional Raspberry Pis to see how the performance and speed continues to scale into the tens.

On the simulation side, we seek to integrate more robust attitude and pointing modelling than simply a blanket time to slew. This will lead to more accurate estimates on the time and power consumption needed, and will additionally allow for a more robust analysis of the trade between RF and laser crosslink communication. We also seek to continue our analysis of the cost benefits of RF vs laser crosslinks for constellations of varying sizes.

By implementing these features, we will bring SPRINT closer to an on orbit demonstration that verifies the results of this paper. Tools like SPRINT will only become more important as we continue to send up more and larger constellations; the only way that is feasible to ensure we are getting the maximum effectiveness from them is by having them autonomousy plan and replan for themselves.

Appendix

A. Orbital Parameters

The orbital parameters for the three constellations used are given.

Table 5 TROPICS constellation parameters

	Plane 1	**Plane 2**	**Plane 3**
A [km]	600	600	600
E	0	0	0
I [deg]	30	30	30
RAAN	0	120	240
Sats in Plane	2	2	2

Table 6 CYGNSS constellation parameters

	Sat 1	Sat 2	Sat 3	Sat 4	Sat 5	Sat 6	Sat 7	Sat 8
A [km]	520	6891	520	520	520	520	520	520
E	0	0	0	0	0	0	0	0
I [deg]	35	35	35	35	35	35	35	35
RAAN	2	4	351	353	354	355	356	360
Argument of Perigee	86	98	90	96	82	92	95	91

Table 7 CICERO constellation parameters

	Plane 1	Plane 2	Plane 3	Plane 4	Plane 5	Plane 6	Plane 7	Plane 8	Plane 9	Plane 10	Plane 11	Plane 12
A [km]	500	500	500	500	500	760	760	760	760	760	760	760
E	0	0	0	0	0	0	0	0	0	0	0	0
I	97.44	97.44	97.44	97.44	97.44	28	28	28	28	28	28	28
RAAN	0	60	120	180	240	300	0	60	120	180	240	300
Sats in Plane	2	2	2	2	2	2	2	2	2	2	2	2

Table 8 Run times of the separated SPRINT simulation on different hardware configurations

	1 Satellite	2 Satellites	3 Satellites
PC Run Time (s)	30	330	343
Separated-PC Run Time (s)	41	103	170
Separated RPi Run Time (s)	1503	2870	3890

Acknowledgments

The authors would like to thank the NASA Small Spacecraft Technology Program for funding support through grant 80NSSC18M0042 and for their guidance through the program. We'd like to thank Dr. Jeremy Frank and Gary Crum for mentoring us. We'd finally like to thank Kit Kennedy, Bobby Holden, Warren Grunwald, and Joe Kusters, without whom SPRINT would not be possible.

References

[1] California Polytechnic State University, "CubeSat Design Specificiation," Technical report Rev. 13, Cal Poly SLO, San Luis Obispo, CA, 2014, ???? Issue: Rev. 13.

[2] Saing, M., "NASA and Smallsat Cost Estimation Overview and Model Tools," ????, p. 51.

[3] Maskey, A., and Cho, M., "CubeSatNet: Ultralight Convolutional Neural Network designed for on-orbit binary image classification on a 1U CubeSat." *Engineering Applications of Artificial Intelligence*, Vol. 96, No. 103952, 2020. URL sci-hub.se/10.1016/j.engappai.2020.103952.

[4] Allen, B., "Cyclone Global Navigation Satellite System (CYGNSS)," , May 2015. URL http://www.nasa.gov/cygnss.

[5] "Mission Overview | TROPICS," , Jan. 2021. URL https://tropics.ll.mit.edu/CMS/tropics/.

[6] "CICERO - Satellite Missions - eoPortal Directory," , Jan. 2021. URL https://directory.eoportal.org/web/eoportal/satellite-missions/content/-/article/cicero.

[7] "Capella X-SAR - Satellite Missions - eoPortal Directory," , Jan. 2021. URL https://directory.eoportal.org/web/eoportal/satellite-missions/content/-/article/capella-x-sar.

[8] "SpaceX submits paperwork for 30,000 more Starlink satellites," , Oct. 2019. URL https://spacenews.com/spacex-submits-paperwork-for-30000-more-starlink-satellites/.

[9] Singh, L. A., Whittecar, W. R., DiPrinzio, M. D., Herman, J. D., Ferringer, M. P., and Reed, P. M., "Low cost satellite constellations for nearly continuous global coverage," *Nature Communications*, Vol. 11, No. 1, 2020, p. 200. https://doi.org/10.1038/s41467-019-13865-0, URL http://www.nature.com/articles/s41467-019-13865-0, number: 1.

[10] Perea-Tamayo, R. G., Fuchs, C. M., Ergetu, E., and BingXuan, L., "Design and Evaluation of a Low-Cost CubeSat Communication Relay Constellation," *2018 IEEE MTT-S Latin America Microwave Conference (LAMC 2018)*, 2018, pp. 1–4. https://doi.org/10.1109/LAMC.2018.8699047.

[11] Kennedy, A. K., "Planning and scheduling for earth-observing small satellite constellations," Thesis, Massachusetts Institute of Technology, 2018. URL https://dspace.mit.edu/handle/1721.1/120415, accepted: 2019-02-14T15:49:32Z.

[12] Campbell, A., "Near Earth Network," , Apr. 2015. URL http://www.nasa.gov/directorates/heo/scan/services/networks/nen, publisher: Brian Dunbar.

[13] Robinson, E., Balakrishnan, H., Abramson, M., and Kolitz, S., "Optimized Stochastic Coordinated Planning of Asynchronous Air and Space Assets," *Journal of Aerospace Information Systems*, Vol. 14, No. 1, 2017, pp. 10–25. https://doi.org/10.2514/1.I010415, URL http://arc.aiaa.org/doi/10.2514/1.I010415, number: 1.

[14] Knight, S., Rabideau, G., Chien, S., Engelhardt, B., and Sherwood, R., "Casper: space exploration through continuous planning," *IEEE Intelligent Systems*, Vol. 16, No. 5, 2001, pp. 70–75. https://doi.org/10.1109/MIS.2001.956084, conference Name: IEEE Intelligent Systems.

[15] Bianchessi, N., and Righini, G., "Planning and scheduling algorithms for the COSMO-SkyMed constellation," *Aerospace Science and Technology*, Vol. 12, No. 7, 2008, pp. 535–544. https://doi.org/10.1016/j.ast.2008.01.001, URL https://linkinghub.elsevier.com/retrieve/pii/S1270963808000187.

[16] Smith, B., Sherwood, R., Govindjee, A., Yan, D., Rabideau, G., Chien, S., and Fukunaga, A., "Representing Spacecraft Mission Planning Knowledge in ASPEN," ????, p. 15.

[17] Chien, S., Doubleday, J., Ortega, K., Tran, D., Bellardo, J., Williams, A., Piug-Suari, J., Crum, G., and Flatley, T., "Onboard autonomy and ground operations automation for the Intelligent Payload Experiment (IPEX) CubeSat Mission," 2012. URL https://trs.jpl.nasa.gov/handle/2014/42586, publisher: Pasadena, CA : Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2012.

[18] Nag, S., Li, A., and Merrick, J., "Scheduling Algorithms for Rapid Imaging using Agile Cubesat Constellations," *Advances in Space Research*, Vol. 61, 2017. https://doi.org/10.1016/j.asr.2017.11.010.

[19] Rigo, C. A., Seman, L. O., Camponogara, E., Morsch Filho, E., and Bezerra, E. A., "Task scheduling for optimal power management and quality-of-service assurance in CubeSats," *Acta Astronautica*, Vol. 179, 2021, pp. 550–560. https://doi.org/https://doi.org/10.1016/j.actaastro.2020.11.016, URL https://www.sciencedirect.com/science/article/pii/S0094576520306858.

[20] Fisher, W. A., and Herz, E., "A Flexible Architecture for Creating Scheduling Algorithms as used in STK Scheduler," ????, p. 10.

[21] Cockrell, J., Alena, R., Mayer, D., Sanchez, H., Luzod, T., Yost, B., and Klumpar, D., "EDSN: A Large Swarm of Advanced Yet Very Affordable, COTS-based NanoSats that Enable Multipoint Physics and Open Source Apps," *Proceedings of the Small Satellite Conference*, 2012. URL https://digitalcommons.usu.edu/smallsat/2012/all2012/89/, technical Session I: The Horizon, SSC12-I-5.

[22] Hanson, J., Luna, A. G., DeRosee, R., Oyadomari, K., Wolfe, J., Attai, W., and Prical, C., "Nodes: A Flight Demonstration of Networked Spacecraft Command and Control," ????, p. 13.

[23] Cahoy, K., Grenfell, P., Crews, A., Long, M., Serra, P., Nguyen, A., Fitzgerald, R., Haughwout, C., Diez, R., Aguilar, A., Conklin, J., Payne, C., Kusters, J., Sackier, C., LaRocca, M., and Yenchesky, L., "The CubeSat Laser Infrared CrosslinK Mission (CLICK)," SPIE, 2019, pp. 111800Y–111800Y–12.

[24] Holden, B. G., "Onboard distributed replanning for crosslinked small satellite constellations," Thesis, Massachusetts Institute of Technology, 2019. URL https://dspace.mit.edu/handle/1721.1/122513, accepted: 2019-10-11T21:59:43Z ISBN: 9781121262690.

[25] Rodríguez, J. L. C., M.G., J., Hidalgo, A., Bapat, S., Astrakhantsev, N., Eleftheria, C., Yash-10, Meu, Dani, Chaurasia, A., Márquez, A. L., Sondhi, D., Mrugalski, T., Selwood, E., Ousoultzoglou, O., Robles, P. R., Lindahl, G., andrea carballo, Rode, A., Eichhorn, H., Garg, H., and Goyal, H., "poliastro/poliastro: poliastro 0.15.2 (astroquery edition)," , Jun. 2021. https://doi.org/10.5281/zenodo.5035326, URL https://doi.org/10.5281/zenodo.5035326.

[26] Aguilar, A., Butler, P., Collins, J., Guerster, M., Kristinsson, B., McKeen, P., Cahoy, K., and Crawley, E. F., "Tradespace Exploration of the Next Generation Communication Satellites," *AIAA Scitech 2019 Forum*, American Institute of Aeronautics and Astronautics, San Diego, California, 2019. https://doi.org/10.2514/6.2019-0768, URL https://arc.aiaa.org/doi/10.2514/6.2019-0768.

[27] Gurobi Optimization, LLC, "Gurobi Optimization," , Oct. 2021. URL https://www.gurobi.com.

[28] Coin-OR, "Cbc," , Sep. 2021. URL https://github.com/coin-or/Cbc.

[29] Makhorin, A., "GLPK (GNU Linear Programming Kit)," , Jun. 2012. URL https://www.gnu.org/software/glpk/.