Initial Results from ACCESS: An Autonomous CubeSat Constellation Scheduling System for Earth Observation

Andrew K. Kennedy, Kerri L. Cahoy
Space Telecommunications, Astronomy and Radiation (STAR) Lab, Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge, MA 02139; 713-824-8499
akennedy@mit.edu

ABSTRACT
We present ACCESS, the Autonomous CubeSat Constellation Earth-observing Scheduling System, which plans constellation operations using both onboard and ground-based algorithms. We discuss the system’s software architecture, which is oriented towards more scalability to constellations of tens to hundreds of satellites and for better performance of data routing to ground. We describe the progress made on an initial version of a greedy data routing algorithm that incorporates crosslink usage. We present results from data routing simulations over a 24 hour planning window with X-band downlinks and optical crosslinks, multiple constellation orbit geometries, and multiple ground station networks. The results show that average data routing latency is improved significantly in most cases when downlinks and crosslinks are used versus only downlinks. A Walker geometry was found to perform best overall in latency, with a reduction from 213 to 23 minutes when using crosslinks for a ground network with 9 stations. We examined latency for urgent, preemptive observations and found that when using crosslinks average latency was reduced to 16 from 25 minutes for Walker. We also examined execution time and found that the algorithm schedules successfully within about 13 real-world minutes for a 100 satellite Walker constellation with crosslinks, demonstrating scalability.

INTRODUCTION

CubeSat Constellations for Earth Observation
As detailed in Achieving Science with CubeSats, constellation-based Earth Observation (EO) offers measurement advantages, including higher temporal resolution, multi-point instrument coordination, and low-latency data availability [1]. The choice of an appropriate orbital geometry for EO constellations has been extensively studied in the literature, with a focus on designing geometries to provide a large percentage of Earth-surface coverage with a minimum number of satellites, and to minimize the revisit times between observations of surface locations [2-6]. The requirements on revisit time depend on the type of target being observed. Applications with high temporal resolution needs, such as disaster monitoring and meteorology, require average revisit rates ranging from sub-hourly to daily [7,8]. For reference, a constellation of 32 Low Earth Orbit (LEO) satellites could achieve global average revisit times of one-half hour [8].

CubeSats may be able to provide the spacecraft platform for these EO applications, primarily because their low cost to develop and launch enables a single organization to field many (tens to hundreds) dedicated sensor nodes on orbit. While the range of different sensor types that CubeSats can feasibly support is more limited than for larger satellites, many sensing payload capabilities are feasible and maturing rapidly [9]. Instruments that can be hosted on a CubeSat include atmospheric sounders (e.g. MicroMAS [10] and TROPICS [11], Figure 1), visible imagers [12], and even hyperspectral imagers [13].

Figure 1: Notional depiction of scanning the Earth’s atmosphere with microwave radiometers on the TROPICS constellation [11].

The Need for Autonomy
As CubeSat constellation sizes scale up, the complexity of efficiently operating the satellites becomes a major concern. Much of this complexity stems from the need to downlink large amounts of data taken either globally or from target regions from the satellite to a limited set of ground stations. This need is complicated by the
inherent energy limitations present onboard a CubeSat-scale satellite. The satellites are usually significantly limited in power production (from solar panels) and battery storage, and power usage needs to be carefully planned to maximize both data production and delivery to ground [14-17]. Figure 2 illustrates the power needs for both a 3U and 6U platform over a set of different earth observation payload scenarios.

Traditional space mission operations architectures that require significant human involvement will not scale well to constellation sizes of tens or hundreds of satellites. Specifically, satellite operations scheduling with a human-in-the-loop planning process becomes so time consuming with more spacecraft that it can be a performance-limiting issue; without a solution, we can expect the number of human operators to scale linearly with the number of spacecraft.

**Figure 2: Energy usage for both data collection and complete downlink of the data over a set of representative EO CubeSat payloads and communication systems [19]. Note that more capable instruments greatly exceed the current power, UHF, and S-band downlink resources.**

**The ACCESS Approach**

The ACCESS system is intended to fully automate the process of scheduling CubeSat constellation onboard activities in order to best route data from observation collection points through inter-satellite crosslinks and downlinks to ground. It is designed to leverage crosslinks to the best extent possible to both decrease the latency of data delivery to ground and maintain up-to-date delivery of satellite housekeeping telemetry and ground commands.

ACCESS approaches constellation activity scheduling using a two-level hierarchical system, as detailed later in the ACCESS ARCHITECTURE section. High level constellation planning is performed first on the ground with a global view of the constellation’s state, allowing for the best possible selection of data collection and communications timing. Plans are uplinked to satellites, which then perform their own planning process using their most up-to-date assessment of satellite state and observation target priority to reactively modify the plan as needed. Satellite resource usage (e.g. power production from solar panels, battery storage) is considered at both stages to ensure the activities selected for the satellites to perform are within desired operational constraints. However, at the global constellation planning level satellite resource usage scheduling is kept as uncoupled as possible from data route selection, in order to reduce computational complexity and allow for scalability to much larger constellations. We believe this focus on scalability is key for operating the large small satellite constellations of the future.

**Organization**

In this paper, we present the architecture for ACCESS and present results from an initial implementation of a data routing algorithm for the system. First we delve into background information on CubeSat EO, existing constellation scheduling systems, and CubeSat communications. We then detail the ACCESS algorithm and software architecture. Next we talk about software development progress, and the specifics of the data routing algorithm tested in the work. We then present the results from this testing. We conclude with a brief discussion of ongoing work and future plans for ACCESS.

**BACKGROUND**

**CubeSat EO Applications and Data Production**

The scope of Earth observing applications for CubeSats is expanding rapidly as satellite bus and payload technology matures for these platforms. As payload capabilities are augmented their data production rates tend to grow as well, placing more and more importance on the effective management of data across EO constellations. We provide several examples of increasingly data-intensive applications below.

The TROPICS constellation (Figure 1) being designed by MIT Lincoln Lab aims to provide rapidly updated data for weather models using a constellation of up to twelve 3U CubeSats with microwave radiometer instruments [11]. The CubeSats are continually scanning the swath of atmosphere below, producing data at a rate of roughly 16 kbps, which equates to about 1.5 GB per satellite per day of data.

Another example is Planet’s (formerly Planet Labs) flock of Dove satellites that perform moderate resolution visible Earth imaging, with the goal to provide updates of Earth’s full surface every day [12,20]. They accomplish this with an optical telescope that occupies most of the space in a 3U CubeSat bus (bus volume of a 3U is 10 cm x 10 cm x 34cm), with a
3.5 meter ground sample distance from a 400 km orbit. Each picture obtained takes about 4 megabytes (MB) and their constellation must downlink roughly 6 terabytes (TB) of land images every day to accomplish their goal. As of 2016 they are capable of downloading about 550 GB per day from all satellites over X-band downlinks with a data rate up to 84 Mbps (bps for bits per second) [20].

An even more data-intensive application is hyperspectral imaging, where the inclusion of many different frequency channels in the data product significantly increases output data rate. Mandl et al. outlined a CubeSat constellation carrying an instrument similar to the Hyperion hyperspectral imager on the Earth-Observing 1 mission [46]; the CubeSat hyperspectral imager would produce raw data rates in the Gbps range, producing TB of data in the course of a 90 minute orbit, assuming enough power were available to run continuously [13].

These large data production rates produce not only a large daily data volume to downlink to ground, but also large “instantaneous” chunks of data that need to be routed to ground quickly for applications like disaster monitoring. For example, using Tsitas and Kingston’s proposed multispectral CubeSat imager for say, tracking the progress of a flood, 127 Mb of compressed data are produced for every second of observation [21].

**Existing Constellation Scheduling Systems**

The problem of single-satellite Earth observation and communication planning has been extensively investigated in the literature, often with a focus on deciding which observation tasks to execute and how to best meet the myriad timing and priority constraints between observations. Complexities are introduced when moving to constellations, due to the interactions in observation timing between spacecraft. Centralized algorithms offer one avenue for scheduling, in which observation and activity timing for all satellites is solved concurrently. The Optwise and Orbit Logic companies provide commercially available algorithms in the STK Scheduler tool [22,23] which plan satellites’ activity sequences to meet resource constraints and maximize a figure of merit function. STK Scheduler also serves as a platform for operators to define their scheduling problem instances with custom objective functions. Monmousseau discusses algorithms for observation and satellite energy usage scheduling used at Planet, based on simulated annealing and Mixed Integer Linear Programming (MILP) [24]. Spangelo and Cutler [25] and Castaing [26] use a linear programming formulation to schedule downlinks for a single satellite and multiple satellites, respectively, while also satisfying resource constraints.

Other work investigates the use of decentralized scheduling algorithms that leverage direct inter-satellite coordination. Van der Horst and Noble [27,28] used market-based algorithms to perform task assignment between satellites with intermittent crosslinks. Das et al. [29] present a distributed planning system where ground stations distribute high level goals to satellites, which then plan their own activities by negotiating tasks with other satellites.

The main limitation of these algorithms is that they don’t take full advantage of the use of crosslinks for routing data in an EO CubeSat Constellation. The centralized algorithms of Monmousseau, Spangelo, Cutler and Castaing address the optimization of data downlink with energy constraints, but don’t model data exchange between satellites through crosslinks. The decentralized algorithms generally only address observation task allocation, and don’t optimize the routing of data through satellites.

We note that the STK Scheduler tool provides users the ability to define a scheduling problem that incorporates crosslink data routing into an objective function and optimize performance. Crosslinks can be scheduled based on inter-satellite access times calculated by the STK geometric modeling engine. This leaves, however, a significant amount of work for the end user in both modeling how data is shared over crosslinks and selecting the best data routes to schedule. Also, STK Scheduler models satellite activities at a low level as tasks, which are then assigned to availability windows for execution. With a LEO constellation architecture it is challenging to determine how many tasks should be used to model crosslinks, because neighboring satellites in the same orbit have essentially continuous crosslink availability. A crosslink-based constellation scheduling algorithm built on this architecture would likely have to incorporate a higher layer that repeatedly divides crosslink windows up into different numbers of tasks, and then attempts to schedule. In practice, the time complexity of such an approach would be significant and limiting as large constellations (~100s of satellites) are considered.

**CubeSat Communications Systems**

Historically, most CubeSats have flown S band or UHF radios [30], with data rates up to 3 Mbps. Such a low data rate significantly limits the amount of daily data volume that can be downlinked to ground from high data rate instruments.
For this reason, organizations are developing CubeSat radios at the X and Ka bands and new optical-frequency-based (“lasercom”) transmitters to enable higher data rates. Planet has flown X-band radios and registered spectrum for downlink data rates of up to 200 Mbps [30], and Syrlinks has developed commercially-available X band systems [31,32]. Astro Digital has demonstrated Ka-band data rates of 40 Mbps and expects to achieve 320 Mbps in 2017 [33-35]. The OCSD (Optical Communications and Sensor Demonstration) mission from Aerospace Corporation aims to demonstrate lasercom up to 100 Mbps [36-38], and Sinclair Interplanetary is developing a lasercom system for up to 1 Gbps [39]. MIT is developing several demonstrations, including the Nanosatellite Optical Downlink Experiment (NODE) [19] for LEO lasercom downlinks at 10-70 Mbps as well as full-duplex lasercom crosslinks. Additionally, deployable antennas are in development to improve the transmitter gain on the space terminal (including patch arrays, reflectarrays, and parabolic umbrella-style deployables) [40].

We focus on two communications systems: (1) an X-band system for downlink and (2) a lasercom system for crosslink. Details for these systems are included in Table 1. X-band has greater maturity for CubeSats and avoids many of the negative weather effects on Ka and optical downlinks. For crosslinks, there are not many mature high rate communications system options. Lasercom power efficiency (see Figure 1) and the lack of regulation on access to optical spectrum offer advantages. We base the X-band downlink system on published data for the Syrlinks transmitter and the lasercom crosslink system on an upgraded version of MIT’s full duplex crosslink architecture, which is a COTS-based 1550 nm architecture that uses a microelectromechanical system (MEMS) Fast Steering Mirror (FSM) to augment the bus pointing capability to < 10 arcsec in each axis, enabling lasercom with narrower beams than only a body-pointed system could support. The crosslink system requires sufficient pointing knowledge from on-board sensors such as star trackers, or a lasercom beacon. The transmitter uses a Master-Oscillator-Power-Amplifier (MOPA) architecture. The crosslink receiver is assumed to have an 8.5 cm aperture, which can fit on a CubeSat.

Table 1: Link Budget Details for Communications System Models [19,31,32]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>X-band (downlink)</th>
<th>Lasercom (crosslink)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization</td>
<td>Syrlinks</td>
<td>MIT</td>
</tr>
<tr>
<td>SWAP</td>
<td>0.25U, 10 W</td>
<td>1U, 12W</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>33 MHz</td>
<td>N/A</td>
</tr>
<tr>
<td>Transmit Power</td>
<td>1 W</td>
<td>0.5 W</td>
</tr>
<tr>
<td>Beamwidth</td>
<td>60 degrees</td>
<td>200 microradians</td>
</tr>
<tr>
<td>EIRP</td>
<td>0 dBW</td>
<td>81.5 dBW</td>
</tr>
<tr>
<td>Typical Atmospheric Losses</td>
<td>-1 dB</td>
<td>0 dB</td>
</tr>
<tr>
<td>Receiver Diameter</td>
<td>5.5 m</td>
<td>8.5 cm</td>
</tr>
<tr>
<td>Receiver G/T</td>
<td>25 dB/K</td>
<td>N/A</td>
</tr>
<tr>
<td>Detector Info</td>
<td>N/A</td>
<td>InGaAs APD w/ 10 dB optical pre-amplifier</td>
</tr>
</tbody>
</table>

ACCESS ARCHITECTURE

In this section we provide more detail on the architecture of ACCESS. We first discuss ACCESS at a high level, then the CubeSat operational model used by the ACCESS algorithms, and finally the planning algorithms themselves.

The ACCESS System

We are developing ACCESS to improve CubeSat Earth observation constellation science performance using both onboard and ground-based algorithms. It is based on our previous work [41] that separates earth observation constellation activity scheduling into a two-level hierarchical system for automated scheduling of constellation operations, as shown in Figure 3.

![Figure 3: Block diagram of ACCESS including planning elements (blue), and flight and ground software (red).](image)

At Level 1 is a global planning and scheduling element (Global Planner, GP), which optimizes observation and communications scheduling across the constellation by reasoning about how data is routed through communications links between satellites and ground stations.
At Level 2 is the Local Planner, LP, which runs onboard each satellite as a more detailed local planner and scheduler. The LP serves as a reactive planning element that can reprioritize observation and communications timing based on an up-to-date accounting of the satellite’s state (e.g. battery level, presence of onboard faults) and real-time updates of observation target priorities. For instance if a forest fire arises in an area that the constellation has been tasked to monitor, LP handles how the constellation reallocates its resources to capture more observations of the crisis area and to downlink that data faster.

The main goal of ACCESS is to balance constellation performance by minimizing as best as possible a set of operations metrics including:

1. The average refresh rates of data obtained from viewing a set of ground targets
2. The average latency of downlink-to-ground of high priority observation data
3. The average refresh rates of housekeeping telemetry downlinked from the satellites and commanding data sent to the satellites
4. The average and maximum depth-of-discharge of onboard batteries

**CubeSat Operations Model**

A key point of ACCESS’ construction is the simple CubeSat operational model it uses to reason about the scheduling of onboard activities. Figure 4 depicts some of the activities performed onboard a CubeSat in an EO constellation, including target observation (obs), downlink to a ground station (dlnk) and crosslink to a neighboring satellite (xlnk). Downlink is used to send data from the satellite to the ground, and crosslinks can be used to route data between satellites for low-latency transfer to ground.

We model the satellite as having a set of operational “power states” in which it can be, based on the activities it is performing. We assume that the satellite has some baseline power consumption, or an “idle” level. On top of this, different onboard devices can be turned on or off, adding a specified amount to the power consumption, as indicated in Figure 4. We allow, in general, for these additional power states to overlap; however, different operational power states might not be allowed at the same time (e.g. downlink and crosslink, because the satellite may only have one transmitter).

**Figure 4:** The CubeSat “power state” operational model for ACCESS. Possible onboard activities are depicted on the left, and shown consuming different (notional) levels of power over time on the right.

We emphasize that this choice of operational model is distinct from modeling the satellite as a state machine, where the satellite can only be in one operational mode at a time. This model simplifies the design of ACCESS’ scheduling algorithms by allowing activities like observation and downlink to be reasoned about as additional activities that produce a “reward”, as opposed to required activities that preclude other events.

We assume the satellite produces power whenever the sun is visible, based on an orbit average input power. Distinct “recharge” power states in which the satellite tracks the sun are also possible, but not modeled here. In addition, we assume that all required attitude maneuvers can be modeled as time transition constraints between other power states (e.g. some time padding is added before an observation for the satellite to slew to look at the target).

**Global Planner (GP)**

The GP optimizes observation and communications scheduling across the constellation. It reasons about observation data collection timing and priority as well as data routing through communications links. The GP operates on the ground at a topologically central location to all ground stations used for communicating with the satellites, effectively giving it a global view of the constellation’s operations. This global view allows the planner to select activity timings for individual satellites that best contribute to the overall constellation goals of collecting the highest priority observation data and ensuring that data reaches the ground as quickly as possible.

The global planner ingests a set of input calculations of satellites’ orbits up to a desired time window. These calculations include timing for observation and downlink overpasses, crosslink access times between satellites, and communications parameters (e.g. link distance, elevation angle) during both downlinks and crosslinks. It consists of two modules in of itself:
1. **GP1 Data Routing**: this module first schedules downlinks from satellites to ground stations, both selecting times for each satellite to have exclusive access to a ground station (which is assumed for this analysis; multiple access consideration will be a future version) and balancing ground station access across satellites. Next, the module selects how to route the data collected at observation events to downlinks via suitable crosslinks. These communications link selections are used to calculate weightings for the Activity Scheduling module – these weightings essentially specify how important it is to perform on observation or communications activity at a given time. Satellite resource usage is not explicitly considered in this module.

2. **GP2 Activity Scheduling**: this module takes the rough time windows and weightings determined by the data routing module and uses them to determine a detailed schedule of activities for the satellite to perform, with specific start and stop times for activities. It determines this schedule based on satellite resource usage and operational constraints on these resources.

After the activity scheduling module, we have a set of detailed activity (power state) timelines for each of the satellites. These timelines are uplinked to satellites during uplink access windows, for execution and reactive replanning by the LP. The GP incorporates the ability to plan based on the performance metrics detailed earlier in this section, i.e. observation data freshness and latency, telemetry and commanding refresh rates, and battery depth of discharge.

For this work we implemented a greedy algorithm for data routing (described below), but we intend in future to implement a Mixed Integer Linear Programming (MILP) or heuristic algorithm (e.g. simulated annealing, genetic) for selection of data routes.

**Local Planner (LP)**

The LP ingests the operations schedule produced for it by GP, which it uses as a strong, but not compulsory, guide for how to schedule its onboard activities in detail. It monitors the spacecraft’s state of health and onboard resources and makes any necessary last minute changes to adhere to the global operations schedule as best as possible while preventing the spacecraft from violating operational constraints. In addition, the LP is allowed to reprioritize its observations schedule if necessary to focus on high-priority observation events that arise, e.g. new evidence of the occurrence of a natural disaster. This reprioritization can be triggered by a data assessment software module running onboard the satellite. Updates to the satellite’s activity plan are reported in the housekeeping telemetry stream to ground.

**Advantages of ACCESS Architecture**

We believe there are several important advantages to the ACCESS architecture, including:

1. **Scalability to Large Constellations**: the hierarchical architecture of ACCESS enables scalability to large (dozens to hundreds of satellites) constellations. The GP separates scheduling into two steps, first reasoning about data routing utility and then onboard resource usage. This significantly reduces computational complexity by decoupling inter-satellite routing constraints from each satellite’s resource constraints.

2. **Faster Data Routing and Coordination**: the incorporation of crosslink usage in the planning process allows for faster routing of data through the constellation and the rapid coordination of observation reprioritization between satellites in response to updates in target priority. Coordination can be accomplished over the same link as data routing, or over a lower rate telemetry telecommand and control (TT&C) link.

3. **Separation of Onboard Scheduling and Constellation Planning**: the ability for a satellite on-orbit to actively replan its own activities would benefit large sets of satellites, possibly from many different originating organizations, and enable them to work together on orbit to form ad hoc networks. The ability to delegate some of the scheduling decision making to the satellite itself can allow the satellites to operate together without having to cede control over their operations.

**DEVELOPMENT PROGRESS AND DATA ROUTING ALGORITHM SIMULATION SETUP**

In this work, we describe the initial version of the Data Routing module of the Global Planner. As part of this work, we developed a MATLAB and Python-based software framework for orbit calculations and representing and reasoning about activity timing, as well as a visualization front end based on AGI’s Cesium.

**MATLAB and Python Framework**

We developed an orbit analysis module in MATLAB to produce the required orbital geometry inputs for the
ACCESS scheduling algorithms. The orbit module consists of two parts. The first is a simple, low-fidelity orbit propagator to give the position of every satellite as a function of time through the simulation period of interest. We use an open source tool that considers only the effects of Earth’s gravitational field with the flattening of the poles [47], which is sufficiently accurate for the purposes of the high-level operations scheduling performed by ACCESS. The second part of the orbital simulation involves feeding the satellite position data through a series of routines to determine access windows for ground target observations, downlinks to ground stations, and crosslinks to other satellites, as well as solar eclipse timing. Data rates during both downlinks and crosslinks are also calculated during this step.

We developed a custom Python-based framework as a foundation for future ACCESS scheduling algorithm work. Our primary goal for this framework are for it to leverage the benefits of Object Oriented Programming in Python to simplify the different types of data structures needed in the scheduling process. We found that the easiest strategy was to encapsulate all information about a specific type of onboard activity in a single ActivityWindow class, with start and end times represented as native Python datetime.datetime objects. This representation introduces a bit more computational intensity than using a simple integer representation of date (e.g. Modified Julian Date), but it made the process of debugging the code significantly easier.

Cesium Visualization

We developed a custom Python library for interfacing directly with AGI’s Cesium, the open source Earth and space visualization engine [42]. The data routing code automatically extracts scheduled observation, downlink, and crosslink windows and outputs them to a .czml file (based on the JSON standard) which can then be loaded into the Cesium visualizer running on a web server on the local machine. We also derive a data storage history for every satellite based on their scheduled production, reception, and transmission of data packets.

Our use of Cesium was a key advantage over the course of this work. It helped us visualize all of the timing and data storage output produced in the data routing code, with the benefit of making debugging a much less painful process. It also served as a good validation of our final results using the visual rendering of the activity improves the user’s understanding, for example, leading to a better understanding of why different constellation geometries have different revisit intervals than expected.

Figure 5: Image from the Cesium-based visualization front end developed for ACCESS. CubeSats are shown along their orbits (yellow lines) with changing stored data volume (cyan horizontal bar), downlinking to ground stations (blue lines), and crosslinking with each other (red lines).

Global Planner Data Routing Module

We implemented both of the main steps in the GP Data Routing module, including downlink scheduling and observation data routing.

Downlink scheduling is implemented as a simple heuristic algorithm that attempts to select the best possible ground station for each satellite to downlink to at a given time, but also handles overlapping downlinks to the same ground station from multiple satellites. Note that we assume every satellite can only talk to a single ground station at a time, and vice versa. We first calculate the total data volume produced during all downlinks windows between all satellite-ground station pairs. We then step through the simulation period timestep by timestep. When a satellite reaches a downlink start time, it schedules itself to that downlink, subject to a few rules: 1. the new downlink has a larger data volume than any already-ongoing downlink, and it has surpassed a minimum required time in the already-ongoing downlink 2. If another satellite is already transmitting to the ground station of interest, it can steal the downlink away if a minimum “exclusive access” time has already passed in the other downlink, and it has a comparable data volume to the other satellite’s downlink. These rules were chosen to both maximize the amount of total bandwidth to ground across the constellation and balance ground station access time between satellites. While we did not attempt to fully optimize downlink scheduling, by inspecting in Cesium the downlink schedules produced by this algorithm we
were able to confirm that satellites choose to downlink over the highest volume links, and that they switch off frequently enough that they all get roughly the same amount of time to downlink.

We also implemented observation data routing in this work as a greedy scheduling algorithm. Figure 6 illustrates the general method used in the algorithm. For every observation, we attempt to trace routes through crosslinks to every other satellite in the constellation and identify the earliest possible downlink that other satellite has that could be used to downlink the observation. We construct a list of such routes to downlinks on as many other satellites as possible, to give a variety of options. Then one of those routes is chosen; preference is given to the earliest end downlink time possible, as long as the routable data volume to that downlink is not significantly smaller than for all other routes.

Routable data volume is the minimum of the data volume throughput of all links in a route. We determine crosslink data volume throughput by dividing long crosslink windows into smaller sizes (for the simulations run in this work, 2 minutes) and determining how much data can be transmitted via the crosslink in that time. Downlinks window data volumes are also calculated and accounted for. In Figure 6 for example, the route to downlink 1 via crosslinks 1 and 2 will be chosen, even though crosslinks 3 and 4 could carry more data volume – simply because downlink 1 is earlier.

This data routing algorithm is considered greedy because we simply make an ordered list of observations to route and then successively assign them the best routes possible, given crosslink and downlink availability. This availability is always decreasing as data packets get routed, because certain routes get cut off for future packets. As mentioned previously, in future work we plan to apply optimization to the assignment of data routes to observation packets, but the greedy algorithm was able to produce worthwhile results.

**Data Routing Simulation**

We set up a simulation to assess the performance of the GP’s data routing module (GP1) over several LEO constellation geometries and ground station networks. The algorithm was run over a window of 24 hours to capture multiple orbits (~16) for each satellite and capture multiple repetitions of the same inter-orbit-plane crosslink topology (satellites in perpendicular orbit planes at roughly the same altitude and true anomaly pass close to each other about every half orbit). For this work, we assume the GP has perfect state knowledge of every satellite at the time of scheduling.

The three orbit geometries simulated are summarized in Table 2. All orbit altitudes were fixed at 600km, and all the satellites in a single orbit are assumed to be equally spaced in true anomaly using propulsion or differential drag. The Sun Synchronous Orbits (SSO) were intended to model an orbit with generally good viewing geometry and lighting conditions; 10:30 and 14:30 Local Time of Ascending Node (LTAN) orbits pass over ground targets at a desirable time of day. The 10 satellite constellation simulates a set of satellites released from a single launch vehicle. The 30 satellite SSO-Equatorial constellation provides better coverage from SSO orbits and includes a perpendicular equatorial orbit to help with “stitching” the other orbits together. The 30 satellite Walker constellation is in a 60°:30/3/1 configuration [43], and is intended to provide all around good observation coverage up to 60° latitude as well as frequent crosslink connectivity.

![Figure 6: Depiction of routing an observation data packet through the constellation (time increases to the right). In the current algorithm, the solid line will generally be picked before the dotted one.](image)
Table 2: Constellation Geometries for Data Routing Simulation

<table>
<thead>
<tr>
<th>Constellation Name</th>
<th>Orbit 1 Description</th>
<th>Orbit 2 Description</th>
<th>Orbit 3 Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Satellite SSO</td>
<td>10 satellites, 10:30 LTAN SSO</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>30 Satellite SSO-Equatorial</td>
<td>10 satellites, 10:30 LTAN SSO</td>
<td>10 satellites, 14:30 LTAN SSO</td>
<td>10 satellites, 0° inc.</td>
</tr>
<tr>
<td>30 Satellite Walker</td>
<td>10 satellites, 60° inc., 0° RAAN</td>
<td>10 satellites, 60° inc., 120° RAAN</td>
<td>10 satellites, 60° inc., 240° RAAN</td>
</tr>
</tbody>
</table>

We calculated downlink and crosstalk data rates based on the link budgets presented in Table 1. Downlink windows were calculated with a 20 degree elevation angle mask. Crosslink accesses were deemed available whenever the line-of-sight vector between two satellites passes above the surface of the Earth. For this work we did not model a required transition time between crosstalks and downlinks, or between a crosslink with one neighbor and a different one with another neighbor.

Three different ground station geographical layouts ("networks") were modeled, all assuming X-band downlink. These are shown in Figure 7. The first had a single ground station located at Wallops Island, Virginia, to model very little ground connectivity. The second had stations at all the locations of the BridgeSat network [44]. This is in reality an optical network, but provides a good middle-of-the-road connectivity case with 9 stations. The third used the Space Flight Networks (SFN) ground station network, with 17 stations [45]. This provides a good high-connectivity test case.

For observations, we modeled a narrow field-of-view payload instrument, with accesses restricted to only those times when a satellite is above a 60 degree elevation angle mask as viewed by the ground target. This was chosen to model the type of high resolution instrument necessary for activities like disaster monitoring. We assumed a continuous payload data rate of 127 Mbps during target overpasses, based on the multispectral imager design by Tsitas and Kingston [21]. In future work we want to extend this to alternative FOV models. A geographically distributed set of 33 observation targets was chosen for data collection in the simulation (Figure 7).

RESULTS

We present performance results from the execution of our initial greedy version of the data routing algorithm over the three constellation geometries and three ground station networks. We first examine routing latency for the constellation, both for “normal” data and urgent data that receives preference for earlier downlink. We then investigate the execution time for the algorithm, and finally discuss energy usage on board the satellites.

All results in this section, except where noted differently, were produced by running the greedy data routing algorithm a single time, with a 24 hour planning window. We did not have time to implement a receding horizon based planning approach; we leave this for future work. Also, as discussed above, we did not have time to implement the activity scheduling module of the Global planner. The results thus do not consider the effect of satellite onboard energy availability on the performance of data routing. Yet we have good reason to believe this effect is minor, as discussed at the end of this section.

Latency of Routing to Ground – Regular Data Only

Figures 8–10 show the average routing latency for observation data packets to ground, for the case where the algorithm applied a 1 gigabit (Gb) observation data preference. That is, the observation windows are ordered temporally from first to last, and then for every observation, the algorithm creates and routes enough data packets to downlink the first Gb of every observation. Only after that does it finish downlinking the rest of every observation. The algorithm attempted to downlink every possible observation window, and thus all observation data collected by the constellation over the scheduling window. In most cases, a few observations did not get fully downlinked; these were not included in the averaging. The average routing...
latency in Figures 8-10 was calculated in the following manner:

1. For each observation that had packets downlinked, count the packet data volumes up in order of packet downlink until 1 Gb is reached. Subtract the downlink time for each packet from its creation time. Average all of these differences together to get the average observation 1 Gb preference routing time for this observation.

2. Average the 1 Gb preference routing times over all observations that had at least 1 Gb downlinked.

Note that observation windows ranged in length from as little as 10 seconds to as much as 120 seconds; with the payload data rate of 127 Mbps, the windows ranged in data volume from around 1 Gb to as much as 15 Gb. Packets were variable in size, ranging from a few hundred Mb to a few Gb.

We see in all of Figure 8-10 that latency performance is quite poor with only one ground station (Wallops) but improves significantly with the 9 ground stations of BridgeSat, and even more with the 17 ground stations of SFN. With the second two networks, performance is better across the board when using crosslinks. This is as expected and desired, per the design of the greedy data routing algorithm – the crosslinks route observation data packets to other satellites that have earlier downlink opportunities, causing downlink time to be closer to creation time.

We see that routing latency with crosslinks improves significantly from the 10 satellite constellation to either of the 30 satellite constellations, dropping by a factor of about 2: 52.5 minutes to 26.9 and 23.3 minutes for BridgeSat, and 37.9 minutes to 19.7 and 17.2 minutes. This shows the benefit of having a large constellation for crosslink data routing: as the number of satellites increases, the network density increases and there are more crosslink opportunities with high data rates. Inter-plane crosslinks are particularly important, because the satellites pass much closer – and thus have much higher crosslink data rates - than satellites that maintain stable separations in the same orbit plane.

Overall, the Walker constellation appears to perform best in terms of routing latency, which makes sense because its geometry provides the most temporally diverse access to ground stations. The 30 satellite SSO-Equatorial constellation also benefits significantly from interplane crosslinks, but does not cover the BridgeSat and SFN ground stations quite as well. Interestingly, the performance gap between no crosslinks and crosslinks is very wide with Walker – 212.9 versus 23.3 minutes – and it is much smaller for SSO-Equatorial. Most of this difference in performance gap is due to the poorer downlink-only performance of Walker however; it appears that the Walker constellation is not well suited for using the BridgeSat network, at least in this particular investigation context. Nonetheless, the close performance parity between these two constellations when using crosslinks shows...
their key ability to equalize routing performance for different geometries.

**Latency of Routing to Ground – Urgent and Regular Data**

Figure 11 shows average routing latency performance again, but in this case we have included a subset of observation targets that are designated as “urgent”, meaning that any observation data collected from them is scheduled for routing first before all other observations. We show average routing latency for both the urgent targets subset and the regular targets subset in the same simulation (over two simulations, with crosslinks and with no crosslinks).

In Figure 11 we increase the number of urgent targets from 5 out of 33 to 15 out of 33 – about half the targets, in the final case. The first thing we notice is that both urgent and regular latency is much lower for the case with crosslinks versus the case without them – both types of latency are about a factor of 10 less. Considering just performance with crosslinks, we see that as we vary the number of urgent targets, it barely affects routing latency. The urgent latency slowly approaches the regular latency (21.4 versus 26.4 minutes for 20 urgent targets), but overall the effect is small. With no crosslinks though, urgent latency degrades by more than an hour, from 105.0 to 172.2 minutes. We note that the relative or normalized change is roughly the same for both the “xlnks” urgent curve and the “no xlnks” curve – however it is more important to focus on the absolute magnitudes here.

![Figure 11: Average observation packet routing latency for 1 Gb preference for both urgent and regular data over varying number of urgent targets, 30 satellite Walker constellation, BridgeSat network, 24 hour window.](image)

**Execution Time**

We examined the execution time for the data routing algorithm to gain an understanding of how scalable this approach is to very large (100s of satellites) constellations. Table 3 summarizes results from runs of the algorithm with a 12 hour and 24 hour scheduling window for Walker constellations of increasing size. The 30 satellite constellation features 3 planes with 10 satellites each, the 60 satellite constellation had 3 planes with 20 satellites each, and the 100 satellite constellation had 4 planes with 25 satellites each. These results were produced on a late 2013 Macbook Pro with a 2 GHz Intel Core i7 processor and 8 GB of RAM – a capable machine, but far from cutting edge in terms of sheer processing power.

We see that **even with 100 satellites in the constellation and planning for 24 hours, the time required for scheduling is manageable, at about 13 minutes**. Scheduling time did increase significantly from the 60 satellite case to the 100 satellite case for both time windows, suggesting that scheduling for multiple hundreds of satellites could be prohibitive. As currently constructed, the greedy algorithm is not parallelizable because routes have to be scheduled and then the availability for the crosslinks and downlinks used removed before new routes can be considered. However, the algorithm is currently implemented in Python, and far from optimized – a significant speedup can be expected from porting to a lower level language like C for production.

**Table 3: Execution Time for Greedy Data Routing Algorithm with Varying Constellation and Scheduling Window Size**

<table>
<thead>
<tr>
<th>Number of Satellites</th>
<th>12 Hour Window</th>
<th>24 Hour Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0.18</td>
<td>0.56</td>
</tr>
<tr>
<td>60</td>
<td>0.94</td>
<td>2.92</td>
</tr>
<tr>
<td>100</td>
<td>4.57</td>
<td>13.23</td>
</tr>
</tbody>
</table>

Table 4 provides an idea of the spatial complexity of the algorithms, showing how many activity windows are available for scheduling and how many are actually scheduled. We also indicate the number of observation data packets routed. We note in particular that not all observations get scheduled, mostly due to the fact that towards the end of the scheduling window there aren’t enough crosslinks and downlinks left to route them.

![Table 4](image)
Table 4: Numbers of Activity Windows Available and Scheduled for 24 Hour Scheduling Window

<table>
<thead>
<tr>
<th>Item</th>
<th>Number of Satellites</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
</tr>
<tr>
<td>Number of Obs. Windows</td>
<td>643</td>
</tr>
<tr>
<td>Number of Obs. Windows</td>
<td>624</td>
</tr>
<tr>
<td>Scheduled</td>
<td></td>
</tr>
<tr>
<td>Number of Obs. Packets Routed</td>
<td>3244</td>
</tr>
<tr>
<td>Number of Dlnk Windows</td>
<td>944</td>
</tr>
<tr>
<td>Scheduled</td>
<td></td>
</tr>
<tr>
<td>Number of Dlnk Windows</td>
<td>784</td>
</tr>
<tr>
<td>Scheduled</td>
<td></td>
</tr>
<tr>
<td>Number of Xlnk Windows</td>
<td>45718</td>
</tr>
<tr>
<td>Scheduled</td>
<td>6072</td>
</tr>
</tbody>
</table>

**Satellite Resource Usage**

We have not yet tied the data routing algorithm into a module for scheduling satellite battery usage. However, in previous, currently unpublished work we implemented a satellite battery model with the same power parameters as the satellites assumed in these simulations. Figure 12 illustrates the battery storage curve for such a satellite. It shows roughly the first 13 hours of a 3 day simulation. The large dips are caused by eclipses, with only minor slope changes due to transmitter and payload usage – we assumed fairly lenient power parameters for the satellite for ease of scheduling. The satellite dips below the minimum desired battery level only for very brief periods (10s of minutes) during a 3 day simulation, and doesn’t dip that far. This gives us confidence that the data routing results in this paper are representative of performance when battery usage is included.

**Figure 12:** Example satellite battery storage curve implemented in previous work. Satellite battery level is black line, red bars are crosslink usage, blue bars are downlink usage, and green bars are observations. Over a 3 day simulation, the satellite only briefly dipped below the minimum desired battery level (purple line, 35 W-hrs).

**CONCLUSION AND FUTURE WORK**

We presented an overview of ACCESS’ scheduling system architecture and discussed how its design was chosen to address the limited ability of current CubeSat constellation scheduling systems to take full advantage of crosslink usage for routing observation data. We detailed our current progress in developing software to implement the data routing portion of ACCESS Global Planner layer as well as software to validate scheduling results in an intuitive visualization front end based on AGI’s Cesium.

Initial results from a greedy data routing algorithm implementation were presented. The results show that average data routing latency is improved significantly in many cases when downlinks and crosslinks are used for routing rather than only downlinks. The 30 satellite Walker constellation was found to perform best overall in latency – when using crosslinks, latency reduced from 212.9 to 23.3 minutes and 78.8 to 17.2 minutes for BridgeSat and SFN respectively. We examined latency for urgent, preemptive observations with the Walker constellation and found that the urgent data packets have lower latency, as desired – using crosslinks with 5 urgent targets out of 33, the average urgent latency was 15.9 minutes versus 25.0 minutes for regular latency. We also examined the execution time for the data routing algorithm, and found that the algorithm executes successfully within about 13 minutes for a 100 satellite Walker constellation with crosslinks.
**Ongoing Algorithm Improvements**

We are currently improving the data routing algorithm in several ways. First, we are implementing continuous planning with a receding time horizon, to allow for satellites to propagate their states forward and for urgent observations to appear in the middle of a simulation scenario. Second, we will soon implement the Activity Scheduling module of the Global Planner, which accounts for satellite onboard energy usage and can modify satellite activity schedules to keep energy within desired operational constraints. Third, we are incorporating a cloud coverage model to account for observation target obscuration as well as allow for accurate modeling of lasercom downlinks.

**Future Plans for ACCESS**

We also have a list of future development tasks for ACCESS, including:

1. Implementation of a non-greedy data routing algorithm that uses optimization to choose a close-to-optimal data routing schedule
2. Deployment of the Global Planner on an open source ground software stack, for example Ball Aerospace’s COSMOS
3. Implementation and test of the Local Planner on a representative flight software platform, for example the cFS (Core Flight System) flight software platform from NASA
4. Test of the integrated ACCESS scheduling system in a “day-in-the-life” planning scenario.
5. Improved satellite and payload instrument modeling and configurability, including wide field of view and scanning instruments, and heterogeneous constellations with different types of satellite nodes.
6. Open sourcing of ACCESS for use by the wider small satellite community.

We place a particular emphasis on the last task – we believe that open sourcing is the best way to make ACCESS a useful asset for the community as a whole. Note that we chose to use the open source Cesium visualizer specifically for this reason – to complement and augment the community that has already formed around that tool.

**Acknowledgments**

The authors would like to thank Emily Clements for her help in developing and documenting the CubeSat operational and communications models.

We would also like to thank Patrick Kage for his significant support and time in developing the interface with Cesium, as well as Hyosang Yoon for his help in orbit modeling.

This work is supported by NASA Earth Science Technology Office grant number NNX14AC75G and NASA Space Technology Research Grant NNX12AM30H. This work was also sponsored by the National Oceanic and Atmospheric Administration under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and not necessarily endorsed by the United States Government.

**References**


42. https://cesiumjs.org/


