ABSTRACT

Title of Thesis:	A simulation tool to study routing in large broadband satellite networks
Degree candidate:	Benjamin Sky Kempton
Degree and Year:	Master of Science, 2020
Thesis directed by:	Anton Riedl, Dr.-Ing., Department of Physics, Computer Science, and Engineering

Large, low Earth orbit, broadband satellite networks have recently become a growing area of research as a number of implementations are currently being constructed by commercial businesses. Unlike previous systems, these new constellations are comprised of 100's to 1,000's of interconnected satellites. This thesis project has resulted in a novel simulation tool that enables the study of inter-satellite linking methods in large satellite constellations: SImulator for Large LEO-Satellite Communication NetworkS (SILLEO-SCNS). Additionally, the thesis presents the analysis of several network designs and constellation structures, with regards to their routing performance.

A SIMULATION TOOL TO STUDY ROUTING IN LARGE BROADBAND SATELLITE NETWORKS

by

Benjamin Sky Kempton

Thesis submitted to the Graduate Faculty of
Christopher Newport University in partial
fulfillment of the requirements
for the degree of
Master of Science
2020

Approved:

Anton Riedl, Chair _____

Jonathan Backens _____

David C. Conner _____

ACKNOWLEDGEMENTS

Thank you Dr. Riedl, Dr. Conner, and Dr. Backens for your encouragement and feedback.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I:
INTRODUCTION

Traditional satellite internet service providers such as HughesNet [1] and Viasat [2] use geosynchronous Earth orbit (GEO) satellites to deliver broadband connectivity across the globe. At an altitude of thirty-six thousand kilometers, a GEO satellite has the advantage of providing coverage to about one third of the Earth's surface. For example, a GEO satellite located over the equator at the same longitude as Dallas TX, or about the middle of the continental US, can establish a connection between Washington DC and Los Angeles. However, the high altitude of a GEO satellite also has a significant downside. The long distances between users on the ground and the satellite introduce a minimum one-way propagation delay of about 240 milliseconds, which is in addition to any other delays, such as processing and queueing delays. For comparison, the latency using terrestrial technologies from LA to DC is on the order of 60 milliseconds. While 230 ms may be acceptable for pure data traffic, it leads to significant performance degradation for other types of services. For instance the International Telecommunications Union (ITU) recommends that applications like audio or video conferencing have less than 150 ms end-to-end delay to maintain acceptable quality of service (QoS) [3]. To be competitive with terrestrial internet, the latency must be reduced by bringing the satellites closer to Earth.

In the past two years Large Low Earth Orbit Satellite Communication Networks (LLEO-SCNs) have become a new space race [4]. A LLEO-SCN consists of hundreds to thousands of communications satellites in Low Earth Orbits (LEO), with altitudes around 550 kilometers, thus, drastically reducing the latency to about 2 milliseconds for up and down links. Additionally, electromagnetic waves propagate faster in vacuum than in fiber optic cables. The lack of geographic obstructions like mountains and oceans allow for more direct communication paths. The shorter paths and faster signal propagation between satellites will allow satellite networks to have latencies similar to or better than the current land-based internet infrastructure. Therefore, propagation delay between two nations will be lower through a satellite network than land-based infrastructure. Although a LEO satellite

communications network is not a new idea, the past two years have seen an explosion of commercial interest in actually building one [4][5][6][5]. The most prominent three projects are SpaceX's Starlink [5], Amazon's Project Kuiper [6], and the OneWeb constellation [7]. All three are generally working to "deliver high speed broadband internet to locations where access has been unreliable, expensive, or completely unavailable." [5]. According to the International Telecommunications Union, in 2019 53.6% of the global population uses the internet [8]. To the aforementioned three companies, the 46.4% of humans who do not currently use the internet are potential satellite internet subscribers. Of course, not all of the 46.4% are offline because they lack infrastructure to access the internet, but those that are represent profit and a reason companies are willing to invest in extremely expensive satellite constellations. In addition, a robust, high speed network with global coverage is undoubtedly of interest to militaries and commercial businesses. For example, high frequency and algorithmic trading companies can gain massive competitive advantages from even small latency reductions in long distance communications [9]. Therefore these organizations are potentially the first (and most lucrative) customers of LLEO-SCNs [9].

Problem

The commercial interest in LLEO-SCNs has come with an explosion of related research publications that look for ways to make large satellite constellations better, faster, and cheaper. One facet of the research is treating the satellite constellation as a network optimization problem. Data can enter and exit the network from any node (satellite), and can cross the network through links (inter-satellite communication channels). The goal of network optimization may be to minimize average latency through the network, reduce the average hop count, maximize bandwidth, or overcome node failures. For a static network, routing that minimizes latency is often done using Dijkstra's shortest path algorithm [10].

One challenge is that the topology of a satellite network is dynamic; the Earth rotates beneath satellites at about 1600 kph[1], and satellites in different planes may have relative

[1] circumference of the equator, ≈40,000 km, divided by 24 hours per day

velocities of around 28,000 kph[2]. Node position and links change as satellites move, causing optimal paths to quickly become invalid. Further adding to the complexity is the asymmetric load on a symmetric network. The mechanics of LEO mean a single satellite will pass over the ground in a different place almost every orbit. Every satellite in the constellation must be designed to handle the network load from a densely populated urban area, even if it will be passing over uninhabited ocean much of the time. Therefore, satellite performance is generally uniform throughout the constellation, while network load (estimated by population density) is not [11]. However, we can take a snapshot of LLEO-SCN at some moment in time, and study it as a static network.

There are many tools designed to studying static networks and to provide insights for optimizations like minimizing hop count. Network Simulator 3 [12] can simulate a static network at the packet level for detailed analysis, studying features like error rates and protocols. NetworkX [13] can be used to look at network graphs, calculate shortest paths, and provide some visualization. However, studying and optimizing a *static* snapshot of a *dynamic* LLEO-SCN for some characteristic like hop count is problematic. An optimal network design for LLEO-SCN snapshot A, may perform very poorly in network snapshot B taken several seconds later, and extremely well in snapshot C taken after a few minutes. Therefore, to study the performance of a LLEO-SCN, we need to take many snapshots at short time intervals and analyze the network at each. Then characteristics like hop count and average latency can be studied as satellites and the Earth move over time.

The dynamics of large LEO satellite communication constellations are a relatively new field of networking research, primarily because it has not been possible to construct such networks until recently. The past 10 years have seen enabling factors including a massive reduction in satellite launch costs, and the development of technologies like inter-satellite high bandwidth optical communication links. As a new field, there is a lack of available research tools to study such networks. In particular, we could not find a tool to simulate the

[2]avg orbital velocity: $v = \sqrt{\frac{\mu}{r}}$ where $\mu = GM$ and $r = 6,600,000$ meters for LEO. $v = 7765$m/s, which converts to ≈ 28000 kph

motion of satellite and Earth-based network nodes, apply some network design to connect all the nodes, and export snapshots of the network at some timestep.

This thesis presents a tool that performs the aforementioned functions: SImulator for Large LEO-Satellite Communication NetworkS (SILLEO-SCNS) [14]. In addition, SILLEO-SCNS also provides interactive 3D network visualization and animation, and is designed to be extensible for implementing novel network designs and constellation structures. The remainder of this thesis is as follows. In Chapter 2 we present background of the problem, key related work, and goals of our research. In chapter 3 the details of SILLEO-SCNS are presented including program architecture, implementation details, features, and performance. We provide analysis of satellite constellation routing performance, using data generated with SILLEO-SCNS in chapter 4. Chapter 5 summarizes the results, discusses future work, and concludes this thesis.

CHAPTER II:
BACKGROUND, RELATED WORK, AND THESIS GOALS

There are four major parts to simulating a dynamic network:

1. The satellite constellation and Earth's physical dynamics (orbital motion, Earth's rotation, etc.).

2. The network design, or interconnection between satellite and Earth-based network nodes.

3. The routing, or paths that packets take through the network.

4. The packet flow due to protocol specifics (packet errors, queueing, congestion control, etc.).

This thesis focuses on presenting a novel tool (SILLEO-SCNS) that covers the first two parts of simulation (physical dynamics, and network design), while using the third (routing) to demonstrate the tool's successful implementation. We leave part 4, packet flow, to other researchers and case specific needs. In this chapter, we first introduce the the fundamentals of satellite constellations and networking, before we discuss related work, and finally present the project's goals.

Satellite Constellation

Given the prohibitive cost (launching 1000's of communications satellites is costly) of a physical system, simulation is needed to study latency and other characteristics of satellite networks. The simulation of a satellite network requires the representation of the network at any given time as well as its dynamic behavior. At any time, the network is represented as a network graph, an example of which is shown in Figure 1. In this system the physical devices or *nodes* are represented by circles, while the communication channels or *edges* are represented by lines and have some associated cost value (often the signal propagation time determined by the length of the link). Network graphs can be analyzed by tools like the Python library NetworkX [13]. However, a network graph is only a snapshot in time

of a satellite network. To know the positions of satellites and ground-stations, one needs to simulate the movement of the the satellites and the rotation of the Earth over time. The dynamics of satellites mean a network graph representation of the system will only be valid for a short period of time and requires frequent regeneration to study the evolution of network characteristics. At the time of this thesis, we have not found any available tools that can provide the aforementioned position and orbit propagation information for a network of thousands of satellites and interface easily with a network analysis tool. Although there are tools like NASA's GMAT [15] to propagate orbits and the NS-3 based Satellite Network Simulator 3 (SNS3) [12] to analyze networks down to the physical layer, neither is acceptable for our purpose. The first, GMAT, targets mission planning and detailed orbit propagation for single satellites. An example use case would be planning orbital manuevers and course corrections for a NASA probe sent to Mars. It *can* generate position data for multiple satellites over time, but has more features and details than are needed for a networking simulation. The second, SNS3, is aimed at broadcast service type networks, where an example use case might be studying antenna radiation patterns for satellite TV constellations. This tool is able to provide network simulation, but is not targeted at large LEO satellite networks discussed in this thesis. For performing an initial analysis of characteristics like latency based on signal propagation time, the only information we need from each satellite is its position. Information like physical structure, orientation, battery status, and antenna radiation patterns need to be ignored.

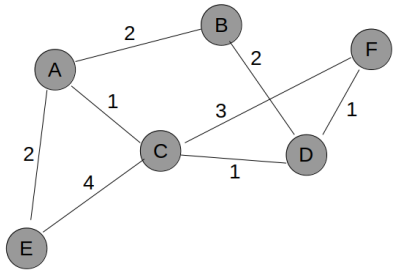


Figure 1: A simple network graph

Satellite positions can be approximated by solving a one-body gravitation problem. In a well known two-body problem, the motion of Earth and the Moon, the two bodies have comparable mass, and the gravitation effects of each must be applied to the other. However in an Earth-Satellite system, the satellite has a negligible mass compared to the Earth, and we only need to worry about effects of the Earth's gravity on the satellite. A well known method of solving this two-body or one-body problem uses a Kepler Ellipse [16].

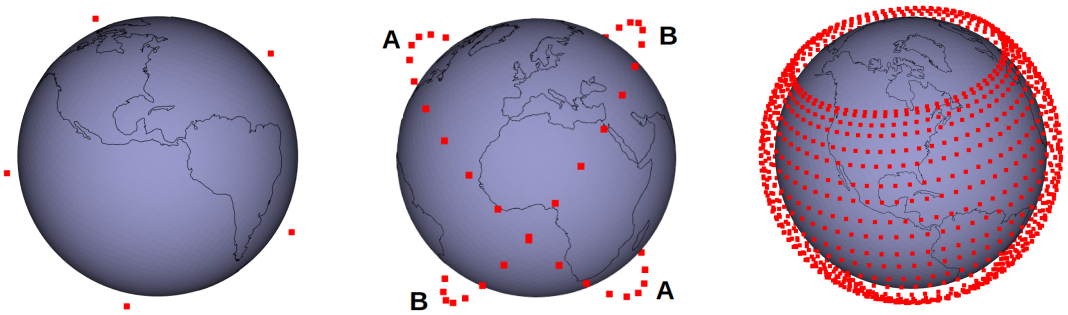
However, the motion of a satellite in low orbit is not influenced solely by gravity. Atmospheric drag, the gravity of the moon, Earth's oblateness, and other factors can influence the orbit of a satellite. Of particular significance in low Earth orbit, atmospheric drag can quickly degrade the orbit of satellites, and necessitates the use of propulsion systems to occasionally re-boost the system. Therefore, in tools like GMAT, where accuracy in orbit propagation is critical, many different forces must be considered. The resulting orbit propagation is accurate, but can be computationally more expensive than a simple two-body problem.

A networking simulation is unlikely to be used for planning orbital manuevers, and does not need to incorporate factors like atmospheric drag. In addition, we expect that the operators of large satellite networks will manuever satellites to maintain a stable constellation structure. Also, the higher computational complexity of a more accurate orbit propagation method could hinder the performance of a simulation. Therefore, using a Kepler Ellipse propagator offers a good balance of accuracy and performance.

Constellation Structure

Large satellite constellations are typically described by the number of orbital planes, and the number of satellites in each plane. In Figure 2a, a one-plane example with five satellites, or a structure of 1:5, is shown. Figure 2b shows two planes (A and B) with twenty-two satellites each, resulting in a constellation structure of 2:22. All the satellites within a plane follow the same orbit but are evenly distributed along it, in this example the twenty-two satellites in each plane are spaced 16.36^o apart. To give a sense of scale, the planed SpaceX

Starlink constellation will have a structure of 72:22 [17]. In Figure 2c, a 72:22 structure is illustrated with a plane inclination of 50^o relative to the equator. The visualization reveals satellites move closer together at higher latitudes, and are farthest apart at the equator. The inclination of the orbits keeps the constellation concentrated in the mid latitudes that have high population densities, and away from the sparsely populated north and south polar regions.



(a) posistion of five satellites in a single plane (b) two orbital planes (c) A 72:22 constillation, at 50^o inclination

Figure 2: Structure of a constellation

Network Design

The information generated by an orbit simulation tool is not enough on its own to perform network analysis; it provides satellite position (graph nodes), while the inter-satellite links (graph edges) still need to be defined. Defining the inter satellite links in the network at any time step, or the linking strategy, is a crucial step in the network design process. This project implements three methods for defining Inter-Satellite Links (ISL).

The first is an ideal case, where a satellite can communicate with any other satellite or ground station in its field of view as limited by the Earth and its atmosphere shown in Figure 3 (the following chapter describes this in more detail). It is likely satellite operators will refrain from transmitting ISLs through the lower atmosphere, to minimize signal attenuation. This results in an enormous number of links per satellite which is unrealistic because satellites have limited numbers of inter-satellite link transceivers. However, this

case does allow for the determination of the theoretical shortest paths through the satellite network between ground stations.

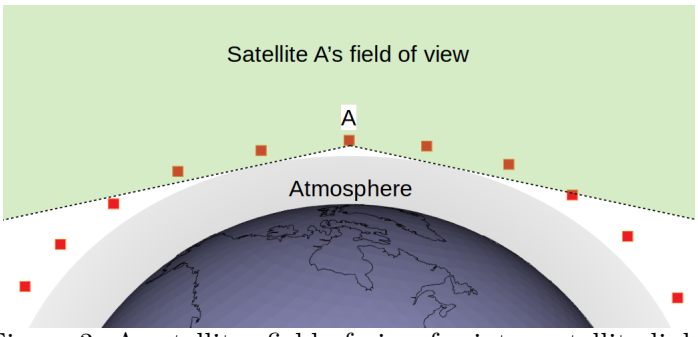
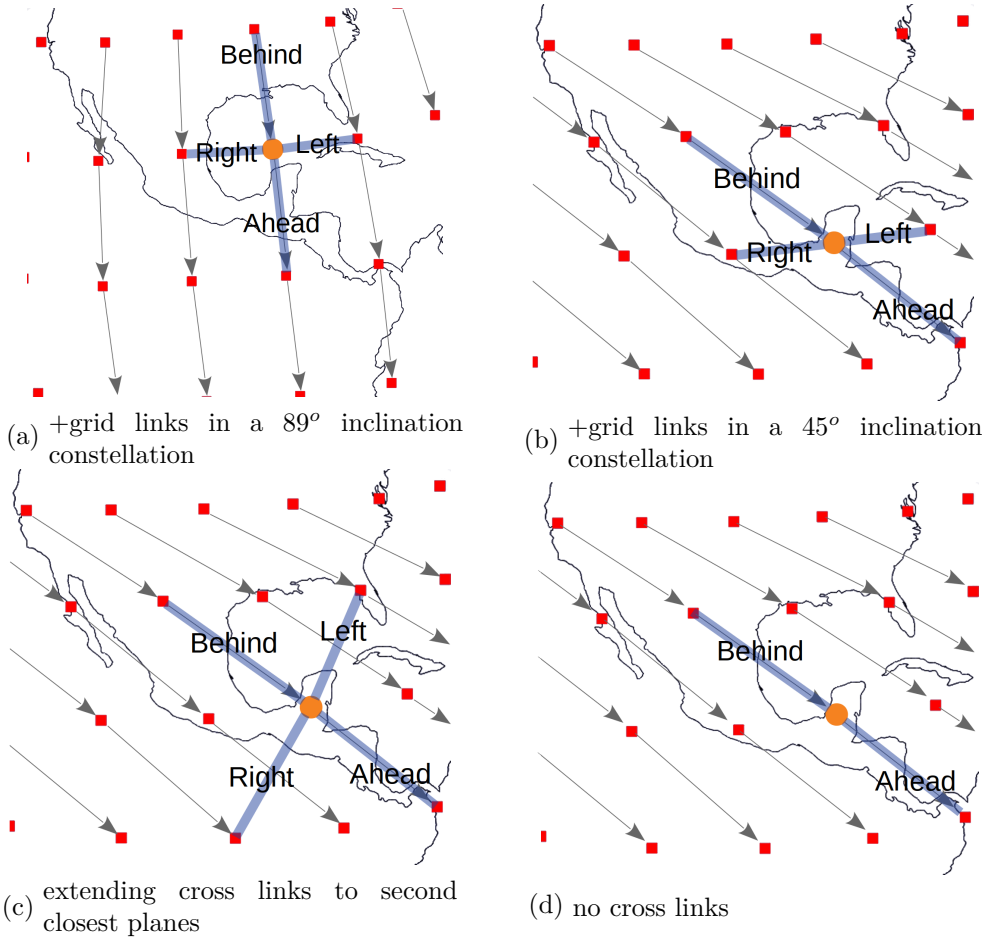


Figure 3: A satellites field of view for inter-satellite links

The second linking method is well known in the field, and used in the operational Iridium NEXT constellation [18]. The strategy, referred to here as +grid, assumes that each satellite can maintain links to four adjacent satellites. Two of the links are always to the satellites ahead and behind (relative to the velocity of the host) in the same orbital plane. The other two links are between the host and the nearest satellite in the planes to the left and right. As shown in Figure 4a, the resulting link pattern appears as a grid of '+' connections in a network with a very high inclination like Iridium. However, unlike Iridium, the large constellations that will be analyzed by this tool have inclinations of around 50^o. This results in a slightly different pattern as seen in Figure 4b. It is not necessary for the cross links to always be with the closest adjacent satellites, as explored in [19]. There may be a latency benefit to using cross links to the second or third closest planes, as shown in Figure 4c. The third linking method that we consider here is a modification of +grid. It assumes that satellites can only support two inter-satellite links, that will be configured as shown in Figure 4d. This design is less expensive to implement (less power and transmission equipment mass in each satellite), at the cost of performance. The lack of crosslinks means groundstation may have to act as relays between orbital planes. However, BSNs are expected to have many small ground stations or user terminals worldwide, so the performance impact may be low.



(a) +grid links in a 89^o inclination constellation

(b) +grid links in a 45^o inclination constellation

(c) extending cross links to second closest planes

(d) no cross links

Figure 4: Comparison of linking methods

In a high inclination constellation like Iridium, the orbital planes can be arranged such that all the satellites on one hemisphere of the planet are moving from south to north and the satellites on the opposite hemisphere are moving from north to south. We call this arrangement a π constellation, referring to the spacing of the constellation's ascending nodes. An ascending node is the point where a satellite moves from south to north across the satellite coordinate system's reference plane ($\approx$ the equator). There will only be one such point per orbital plane. Thus if the ascending nodes of multiple orbital planes make a 180^o arc around the Earth's equator, it is a π constellation. A 2π structure is when the ascending nodes are spaced evenly around the whole equator, making a full circle. A π constellation provides global coverage when the orbital planes have an inclination close to 90^o, but results in asymmetrical partial coverage then the satellites have a inclination less than about 70^o. Therefore, planned LLEO-SCNs generally have a 2π structure, and

inclinations less than 70^o.

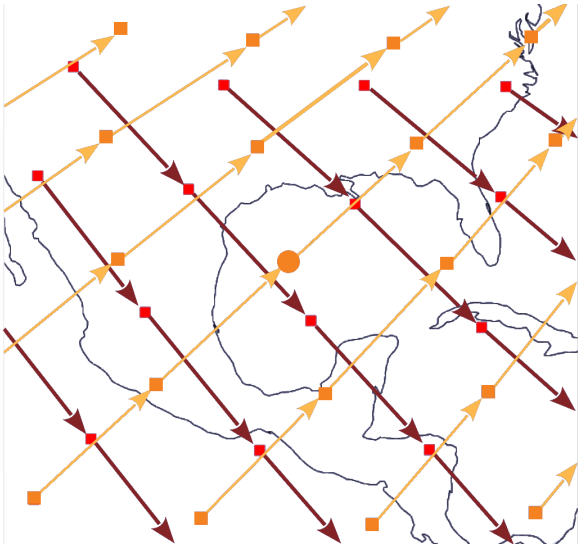


Figure 5: Movement in a 2π constellation

An example of the motion of satellites in a 2π structure is provided in Figure 5. The orange points represent satellites moving from south to north (indicated by the orange arrows), and the red points represent satellites moving from north to south (indicated by red arrows). The static +grid links are shown in yellow, between the orange nodes which all have low relative motion. Satellites moving south (red) will have high relative motion to the north moving satellites. The two groups (north moving and south moving) are said to be in different phases, and creating and maintaining a link (green, Figure 6) between two phases is difficult. There are two methods of communicating between satellites. The first is radio, which is likely to consist of a directional antenna on a gimbal. The second is optical (laser) based, and also involves a transceiver mounted on a gimbal. An optical system is preferred because it provides higher bandwidth for less power and mass than a radio based system, at the cost of greater technical difficulty [20]. Regardless of the transmission medium, two way (duplex) communication requires two satellites, each with a transceiver pointed at the other satellite. Depending on how tightly each transceivers radio or optical beam is focused, keeping two transceivers pointed at one another can be very difficult. In an optical system for example, imagine the pointing accuracy needed to keep a laser focused on a small, dresser sized satellite, hundreds of kilometers away. For satellites in the same phase

with low relative motion, this is challenging, but not impossible. However, for satellite in different phases with high relative motion (1000's of Kph), accurately tracking each other's transceivers is challenging. Despite the technical difficulty, creating links between phases is desirable because it keeps communication paths short in cases where two ground terminals are connected to satellites of different phases. An important assumption of ISLs is pairing; a link requires a transmitter+receiver pair from both of the satellites. For example in Figure 6, if each satellite can only support five links, node B can only talk to the cross phase node A and vice versa. This exclusivity makes it difficult to choose when A should switch from B to another red node. Unless all nodes in the network switch cross phase links in lockstep, whatever red node A tries to switch to will probably be busy. In addition, it may take several seconds to establish any new link, because the transceivers for each link will have to slew around and lock on to one another. The large link churn resulting from cross phase communications has the potential to cause many undesirable routing updates, which lead to network overhead and more time spent recalculating shortest paths.

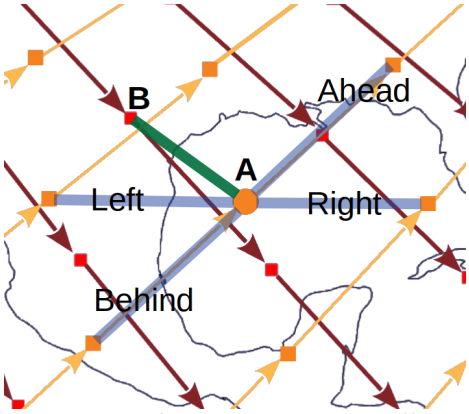


Figure 6: Linking in a 2π constellation

We do not include cross phase links, because it is unlikely to be used in LLEO-SCNs due to its difficulty. This results in two semi-independent networks as seen by a user on the ground, particularly at mid latitudes. At the highest and lowest latitudes in each orbit a satellite will switch phases (northerly phase to southerly phase at the northern most point in a orbit). Therefore cross phase communication paths exist along the north and south rims of the constellation, as shown in Figure 7. For example, a path between A and B will

be most direct by traveling along the rim of the constellation, and in doing so may cross between a north and south phase. However the most direct path between A and C will not pass by a rim of the constellation, and so should only travel along satellites in the same phase. In order to assure that a close to optimal path exists between any two ground points (reachable by the constellation), at least one satellite from each phase must be directly reachable by a ground station at all times. If this condition is not met the following could occur: point A wishes to communicate with point C, however A can currently only reach a north phase satellite, and B can only reach a south phase satellite. In this case, the two nodes that wish to communicate are physically close, but very distant over the network topology. One possible solution is to use ground stations as relays to switch satellite phases without passing one of the constellation rims.

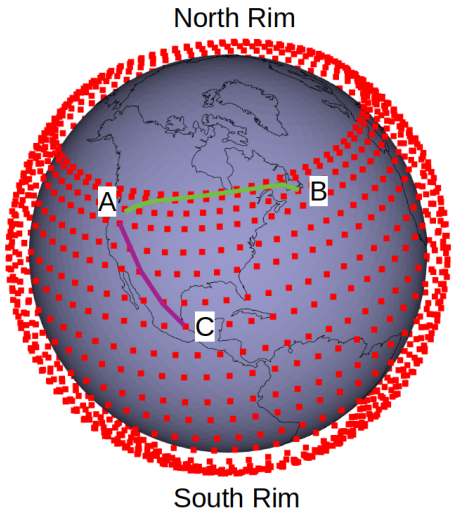


Figure 7: Cross phase communications in a 72:22 50^o constellation

Related Work

There is a large body of work, research, patents, etc. that has been growing ever since the first satellites were put in orbit. For example, the first commercial communications satellite, Intelsat 1, was launched to a geosynchronous orbit in 1965 [21]. Later satellites launched by Intelsat were used for packet based communications as part of the early internet, and spawned research into satellite networking. One example of such work explores a

"contention-based demand assignment protocol" to be used with geosynchronous satellites, submitted in 1977 [22]. A more recent 2002 article, *Topological Design, Routing, and Handover in Satellite Networks* [23], provides a detailed overview of LEO constellations. These details include satellite network topologies, satellite linking methods, hand-over methods, and routing to name a few. However, at the time of the article's publication, the most significant operating satellite network was the Iridium constellation (about 100 satellites). Accordingly, the article assumes a similar scale of network in its discussions. In fact, many papers ([24], [25], [26]) that introduce new routing algorithms and optimizations for satellite networks use an Iridium-like constellation for their simulations (close to 90^o polar orbits, and static +grid links). The currently-in-development satellite networks that SILLEO-SCNS is designed to simulate are larger in scale (1,000 to 10,000 satellites), take on more complex structures, and use different inter-satellite linking strategies. Therefore, much of the research and simulation efforts older than about 10 years may not apply readily to current large LEO networks. A detailed discussion of this appears in *Gearing up for the 21st Century Space Race* [9]. As implied by the paper's title, there is a great deal of excitement in both the commercial and academic worlds surrounding the development of constellations like Starlink, OneWeb, and Project Kuiper.

The first major related work was published in December 2019: *Network Topology Design at 27,000 Km/Hour* [19]. The authors explore a novel inter-satellite linking strategy that greatly improves the theoretical throughput of the Starlink and Project Kuiper constellations. Rather than use a +grid or other simple linking strategy, the authors came up with a method to search the set of all possible stable inter-satellite links to determine an optimal subset. In addition, they introduced a method of choosing different optimal linking strategies depending on a satellite's position in its orbit. Although not animated, the authors also provide a 3D visualization of their new linking strategies. The code developed for link optimization is public. The second major related work is an ongoing project, with the latest update in November 2019: *Using Ground Relays for Low-Latency Wide-Area Routing in Megaconstellations* [27]. The author performs extensive analysis of the in development Starlink network, with a focus on latency. In particular, the November 2019 paper looks

at the performance of the Starlink network in a case where there are no inter-satellite links. The author demonstrated that by using ground stations or user terminals as relays, the network can still provide low-latency communications. The paper references videos, in which the author demonstrates his simulation and visualization running on a laptop. A game engine is used to animate the motion of Earth, satellites, and their links, However, the code for the simulation has not been publicly released.

Goals of this Thesis

As an engineering effort, the primary goal of this project is the development of a tool capable of simulating an arbitrarily structured satellite network/constellation, applying an inter-satellite linking method, and generating a representative network graph. Once the network graph is generated, the tool provides some limited analysis capability: measuring the characteristics of a set of user defined *ground - satellite network - ground* connections. The measured characteristics for connections include: how long paths last, path length, and path change over time. In addition, we make a significant effort to write high performance and efficient code that provides a quality user experience. The secondary goal of the project is to perform analysis of the differences in routing performance for the three linking methods implemented, and differences in routing performance between different sizes of satellite constellations: The three linking methods implemented are:

- Ideal - all possible connections
- +grid - 4 connections per satellite to close neighbors, used in current satellite networks
- limited +grid / sparse - 2 connections per satellite, subset of +grid

To summarize, this thesis seeks to: create and simulate large satellite constellations, create networks from the constellations using various linking methods, and analyze the network's routing performance as a proof of concept.

CHAPTER III:
SIMULATOR

In this chapter we first discuss the requirements for our simulator, before we describe design aspects and its use. The implementation of the simulator in Python is described in a high level overview, and then broken down into detailed explanations of each class. The requirements summarized in Table 1 provide an overview of the developed tool's capabilities and its rationale.

Simulator Requirements

The key differentiating feature of a LEO network is rapid orbital motion of satellites relative to the ground and other satellites. This movement needs to be accurately captured in simulation. In addition, LEO networks may range in size from 10s to 1000s of individual satellites, arranged in different constellation structures. Requiring a user to manually define the orbit of each satellite would make setting up a simulation extremely tedious. Therefore, simply providing the number of satellites and the constellation structure (altitude, number of orbital planes, inclination, etc.) should be enough for a simulator to automatically define the orbit of every satellite.

Turning a group of satellites into a communications network requires linking them. However, there are constraints that limit the possible connections, the bulk of the Earth in particular. Satellite communication is radio or optical based, and is limited to line-of-sight. Therefore satellite-to-satellite links cannot pass through the Earth, and ground stations on the surface of the Earth will have a limited view of the satellite constellation. Hence a simulation needs to know where the surface of earth is relative to the satellite network. The sheer number of *possible* line-of-sight links scales with the size of the constellation, however individual satellites will only support a finite number of concurrent links. Out of all possible links, the subset used by a given constellation is determined by some linking method or network design. A simulator should be flexible, allowing a researcher to easily switch between network designs and implement new ones. Customized network designs

created using the simulator should be easy to export in a standard format: Graph Markup Language (GML).

Mentally visualizing a large, dynamic satellite constellation is not intuitive and makes it difficult to understand network designs. Therefore a network simulator which provides an interactive 3D visualization of Earth and the satellite constellation is desirable. In addition, a GUI for controlling the simulation lends itself to improving the user experience. Finally, a feature laden simulator that runs poorly and has no documentation is unlikely to be helpful to anyone. It follows that the simulator code should be well designed and efficient, while having good documentation. Table 1 provides a summary of general simulator requirements, and Table 2 provides tool-specific requirements.

Table 1: Simulator Requirements

Requirement	**Rational**
1. Simulate the orbital motion of a arbitrarily structured LEO constellation	Network dynamics caused by orbital motion of satellites is the primary factor differentiating the study of a satellite network from well understood static networks.
2. Simulate Earth	Measuring the latency between pairs of cities or ground stations requires knowing their position on Earth in relation to the satellite network. The Earth's rotation is a factor of network dynamics, and its bulk limits the field of fiew of satellites.
3. Generate network links from a user defined strategy	Without links between satellites and ground stations (and possibly other satellites), there is no network.
4. Generate 3D visualization of the simulation	Understanding the structure of a simulation, and detecting errors such as a physically impossible orbit, are enabled by a 3D visualization. In addition, it greatly aids in presenting the research.

Table 2: Tool-Specific Requirements

Requirement	**Rational**
1. Provide a user interface for controlling simulations	A (good) user interface should increase the useability and value of the tool, in addition to reducing user error.
2. Efficient, high performance code implementation	Slow, innefficient code results in a poor user experience. Therefore an effort is made to write high performance code.
3. Output network graphs in a common format	To make the tool useful for future work it should be able to output a network graph at some user defined simulation timestep, in a common format like GML (Graph Markup Language).
4. Provide documented and modular code	To be of potential value to the satellite networking research community, the code must be understandable and expandable for future projects. An install and user guide are considered appropriate documentation.

Functional Overview

This simulation tool is developed in Python3 and split in to three classes. The primary class is named Constellation and implements orbital motion, ground station motion, network design (linking methods) and data structures for nodes and links. Animation and visualization is handled by the second class, Simulation. Third, the GUI class provides simulation controls and options for a user to quickly get started with the tool. The block diagram in Figure 8 illustrates the class hierarchy. In order to keep the GUI responsive it runs as a separate process and communicates with the simulation process via a Python multiprocessing pipe.

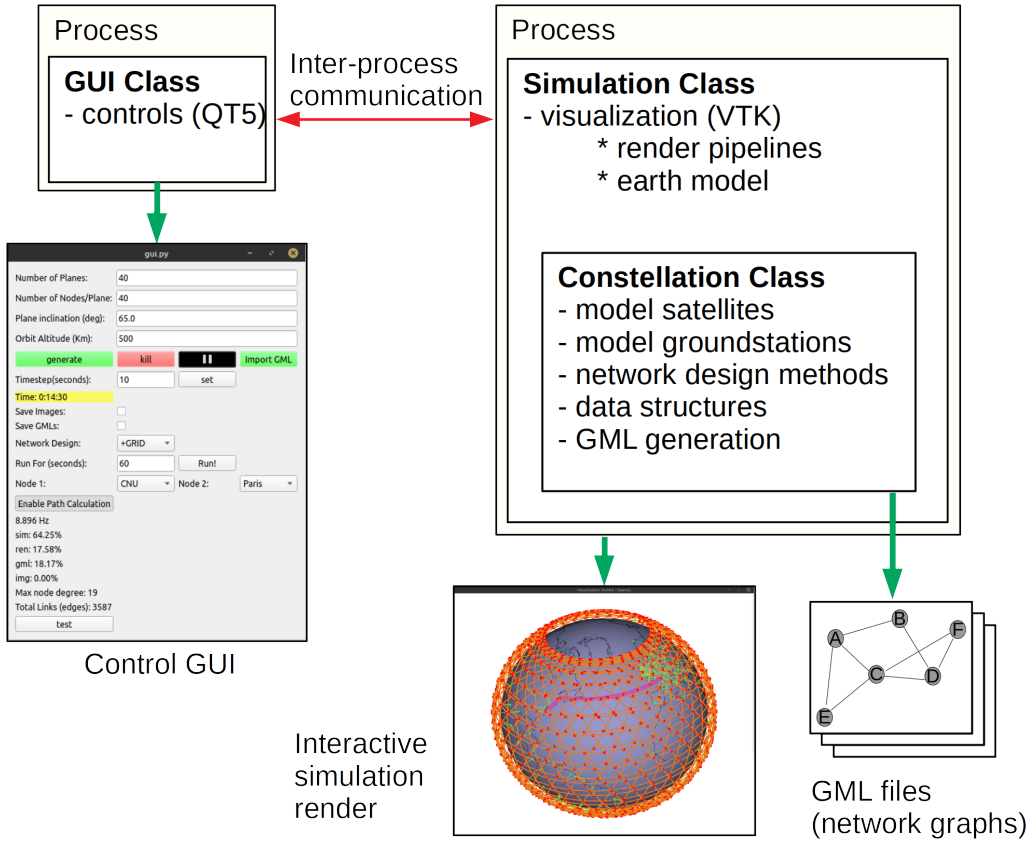


Figure 8: Simulation tool classes block diagram

Constellation Class

The Constellation class is the core of the simulator and contains all the functionality needed to generate satellites, ground stations, links, routing, and export GML files. A user passes arguments for the number of orbital planes, satellites per plane, orbit radius, and inclination. While generating satellite initial positions, default settings assume that circular orbital planes are evenly spaced around the equatorial plane, and that the spacing of satellites within each plane is uniform. However, initialization keyword arguments can be passed to set a non-zero eccentricity, and an arbitrary arc for the ascending nodes. Once initialized, a Constellation object has various functions for adding ground stations, setting the model to a particular time, exporting position data, applying network designs (linking methods) and exporting GML files. It is possible to invoke and run a Constellation object with just a python interpreter window, or a custom script, however visualization requires the Simulation class.

Coordinate System

A 3D cartesian coordinate system is used for the simulation, with the origin located at Earth's center of mass, outlined in Figure 9. In the simulator, Earth's surface is modeled as a sphere with a radius of 6,371,000 meters. The XY plane aligns with Earth's equator, and Earth's rotation axis aligns with the Z-axis. However, the coordinate system does not rotate with the Earth, making it an inertial coordinate system. Upon simulation initialization (t=0), Earth's prime meridian (0^o longitude) aligns with the positive X-axis. As the simulation progresses, Earth rotates once per 24 hours counterclockwise (looking down from above) around the Z-axis. Satellites orbit around the origin, where the XY-plane is the reference plane, and the positive X-axis is the reference direction. In a zero inclination orbit, satellites move counterclockwise around the Z-axis. Increasing inclination rotates the orbit counterclockwise around the X-axis.

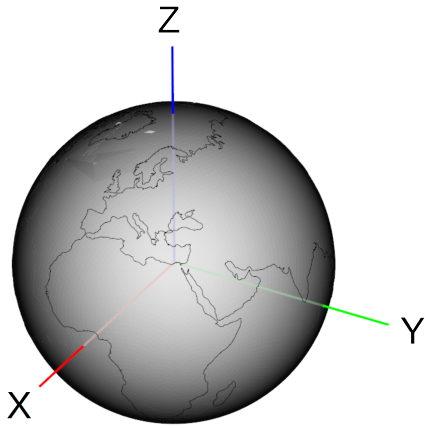


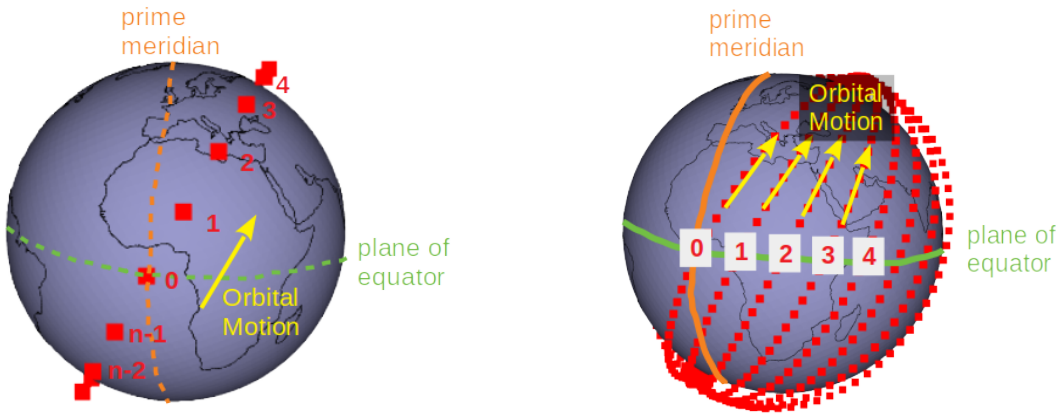
Figure 9: Earth relative to coordinate system at simulation time=0

Satellite Simulation

Satellite positions are the result of a two-body problem, solved by a Kepler Ellipse. The Ellipse solver method is described in [16], and implemented in the Python PyAstronomy [28] library. All the satellites in an orbital plane share the same Kepler Ellipse solver, and each satellite has a time-offset value that is used when calculating its position. The first satellite has an offset of 0, the second an offset of $\frac{orbitalPeriod}{satellitesPerPlane}$ seconds, and so on, such that the satellites within a plane are evenly spaced. When the simulation updates satellite positions

for a new time, the Ellipse solver for each plane is given the time and time-offset for each satellite, and returns their new position. Every time the ellipse solver for a particular plane is used, the returned position is calculated using the same parameters that were used during the simulation initialization (semi-major axis, eccentricity, inclination, and argument of the ascending node). This is important when we consider rounding error accumulation as the simulation progresses. If the next-timestep position was always calculated from the current-timestep position, rounding errors would accumulate, and cause errors in satellite position.

Satellite nodes are given unique ID numbers upon initialization, explained in Figure 10. ID numbers are based off three values, the satellite offset number, plane number, and the (constant value) number of satellites per plane. As shown in Figure 10a, numbering starts at zero, with the satellite immediately following the plane's ascending node, at simulation time=0. Satellite offset numbers increase in the direction of orbital motion, with a maximum of one less than the number of satellites per plane. Orbital plane numbering is similar, with plane zero always having an ascending node offset of zero (in other words, the ascending node of plane zero lies at zero latitude, zero longitude at simulation time=0). IDs for satellites are always positive integers, beginning at 0, and are always assigned upon constellation initialization at time= 0. Although other numbering schemes would be more human friendly, like [*plane-number, sat-number*] they would bring difficulties when creating links. Since network links are described by a pair of endpoints, the endpoint names should be unique single values, not a combination of multiple values *(plane-number, sat-number)*. In addition, using single integers as identifiers allows for efficient code. For example, comparing satellites is a fast integer comparison, and the satellites-array index of a satellite matches its ID number. This means that given a satellite ID, the data for that satellite can be directly accessed (ID=array-index). If ID did not relate to array index, the array would first have to be searched to find the index of a particular satellite ID, which would add massive overhead and slow down the simulator considerably. Satellite IDs can be calculated using the formula in Figure 10.



(a) Satellite Offset numbers

(b) Orbital plane numbers

```
Satellite_ID = (plane_number * satellites_per_plane) + offset_number
```

Figure 10: Numbering scheme for satellites

The importance of using a memory efficient and easily iterable data structure to store satellite data is critical to simulation performance. As the number of satellites in the constellation becomes very large ($n = 10000$), calculating the set of all valid links for an ideal network design becomes a n^2 hard problem. In order to enable efficient iteration and maintain a small memory footprint, satellite data is stored in a Numpy array with the data structure shown in Figure 11. Note that position data is stored as 32-bit signed integers representing meters. This data type limits the max extent of the model to $\pm(2^{31} - 1)$ meters, which is about $5.5x$ the distance from Earth to the moon and should be more than enough for simulating low Earth orbit satellites. Also note the data type limits position accuracy to the nearest meter, which again, should be sufficient for a networking simulation. Even though the satellite ID can be calculated from the plane number and offset number, it is stored redundantly. This minimizes the amount of calculation that has to be done in repetitive operations. If the ID value for a satellite had to be calculated from offset and plane numbers every time it was needed, it would involve some math and multiple memory accesses, rather than a single memory access and no math if the ID is already available.

```
# satellites are positive integers
Satellite_ID = (plane_number * number_of_satellites_per_plane) + offset_number


# Satellites array
SATELLITE_DTYPE = np.dtype([
  ('ID', np.int16),            # ID number, unique
  ('plane_number', np.int16),  # which orbital plane the satellite is in
  ('offset_number', np.int16), # satellite number local to a plane
  ('time_offset', np.float32), # time offset for Kepler Ellipse solver
  ('x', np.int32),             # x position in meters
  ('y', np.int32),             # y position in meters
  ('z', np.int32)])            # z position in meters
```

Figure 11: Satellite array structure

Ground Station Simulation

In addition to satellites, we need ground stations to calculate end-to-end paths through the network. ground stations are network nodes that are fixed to the surface of the Earth and rotate with it. This includes static facilities used by the constellation operator and mobile user terminals, like ships and cars. Even in the case of a very mobile user terminal, like an airplane traveling at 500+ Kph relative to the ground, the movement is negligible compared to the movement of satellites (27,000 Kph ground-relative). Therefore, the only motion applied to ground stations is the rotation of the Earth. ground stations are added to the model in two steps: First the Constellation class has an *addGroundpoint()* function that takes only a latitude, longitude and altitude. The spherical coordinates are converted to 3D cartesian, and then a new entry in the ground stations-array (see Figure 12) is created. ground stations have negative ID numbers (opposed to positive numbers for satellites), starting at -1 for the first ground station added, and decreasing thereafter. The initial position and current position are stored for every ground station to prevent the accumulation of integer rounding errors. When the simulation time is updated, the difference between the previous time and the new time is used to calculate how many degrees the Earth has

rotated in the elapsed time. The calculated angle is divided by 360 to get the remainder, or the angular offset the Earth has relative to its initial position. Then, a rotation matrix for that angle is calculated and applied to the initial position of each ground station to get its new current position. If the new position of a ground station (3 32-bit integers) was calculated based on the current position, integer rounding errors would accumulate every iteration and cause the ground station to drift.

An ID (negative integers for ground stations) is not very human friendly. Therefore the second step to working with ground stations in the model occurs in the Simulation class. Upon initializing a simulation, a text file (specified by the user) of ground station names and coordinates is parsed. Place-names of ground stations from the imported file (like 'CNU' for location 37.0639, -76.4947) are added to exported GML files, and the GUI.

```
# A global ground station counter is initialized to -1, and
# decremented after a ground station is added. ground station ID
# is set to the value of the counter upon creation.
# Thus, ground stations always have negative IDs, and their
# array index is -(ID+1)

# ground stations array
GROUNDPOINT_DTYPE = np.dtype([
  ('ID', np.int16),     # ID number, unique, = array index
  ('init_x', np.int32), # initial x position in meters
  ('init_y', np.int32), # initial y position in meters
  ('init_z', np.int32), # initial z position in meters
  ('x', np.int32),      # x position in meters
  ('y', np.int32),      # y position in meters
  ('z', np.int32)])     # z position in meters
```

Figure 12: Ground station array structure

Three network designs are implemented: ideal, +grid, and sparse. Link data is stored in a numpy array with the structure shown in Figure 13, similar to satellites and ground stations. Each array index, or 'link', is 8 bytes, storing two endpoint IDs and the link distance. The array is initialized to a default length of 10 million. At this length, the array takes up 80 Megabytes of memory, but should be large enough for extremely large constellations, even when using an ideal network design. The array is not dynamically allocated, because it would be computationally expensive. If we wanted to increase and decrease the size of the link array dynamically, a deep copy would need to be performed every iteration. In other words to make the array one index larger, memory for a new (one-larger) array would first be allocated, then the old array plus the new index would be copied to the new memory, and finally the old array deallocated. A deep copy of a large million+ index link array would take significant time, and need to be performed every time the array was resized. In the case of the simulator, 80 Megabytes of memory is tiny for modern computers which typically sport 4-8 Gigabytes of RAM. However, spending the time to dynamically resize an array is expensive, and would take up a significant fraction of the total computation run every simulation timestep. In an ideal network design, where the set of valid links is recalculated every timestep, we simply overwrite the link array, and keep track of the last-link index. In a +grid (or sparse) network design, satellite-satellite links are static and only added to the link array upon simulation initialization. Every time step for a +grid design, the program simply iterates through all the valid satellite-satellite links in the link array and updates their positions as the satellites move. Then, all of the satellite-ground links are calculated and added after the satellite-satellite links, overwriting themselves every time-step.

```
# link array
LINK_DTYPE = np.dtype([
  ('node_1', np.int16),    # an endpoint of the link
  ('node_2', np.int16),    # the other endpoint of the link
  ('distance', np.int32)]) # distance of the link in meters
```

Figure 13: Link array structure

The ideal network design calculates all physically possible links at every time step, assuming satellites and ground stations can support an unlimited number of connections. Links between satellites and ground stations are valid as long as the satellite is above a specified elevation above the horizon. The default elevation is 40^o, but can be changed by the user. Inter satellite links are valid as long as they pass at least a minimum user defined distance above Earth's surface or "minimum communications altitude". For very large constellations at high altitudes, the number of valid links in the network can quickly pass into the millions. Maximum link length is calculated by solving the side-side-angle right triangle shown in Figure 14. A line (green) intersecting a satellite, and tangent to the circle defined by the minimum communications altitude is calculated. Second, the side-side-angle triangle is solved using the radius of the satellites orbit (red), min communications altitude (red), and right angle. The final max link distance is twice the length of the resulting line segment (green). For example in a network where a satellite can see $\frac{1}{10}$ of all other satellites, the number of inter satellite links will be on the order of $\frac{n^2}{10}$ (10 million for $n = 10000$). Ideal links are calculated by calling the *calculateIdealLinks()* function in the constellation class. Although not likely to be used in a real system, the ideal network design allows for calculating the shortest possible paths for a given constellation. In addition, the ability to export a GML file containing all possible links allows a user to modify or optimize the network with an external program, and then reimport the GML to see how the network evolves.

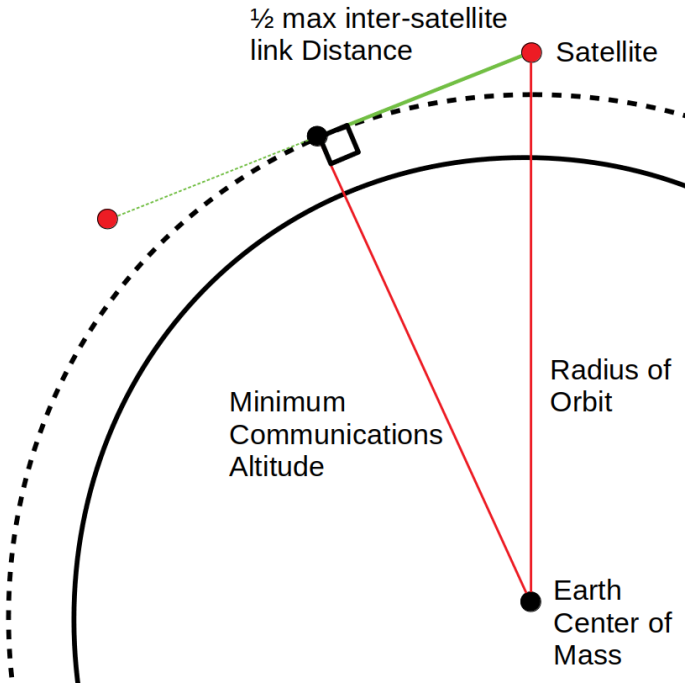


Figure 14: Definition of triangle used to calculate max ISL length

The +grid network design results in 4 inter-satellite connections per satellite that remain active for the duration of the simulation (see Figures 4a and 4b). Unlike the ideal network design, +grid satellite-satellite links are not recalculated every time-step; only the distances are updated. Links are initialized with calls to the *calculatePlusGridLinks()* function in the constellation class. The +grid links are initialized by passing a *initialize=True* keyword argument and updated by calling the function without the argument.

Calculating links for the sparse network design (see Figure 4d) is done by calling the same function as +grid: *calculatePlusGridLinks()*. A keyword argument, *crosslink-interpolation*, allows the user to limit the number of satellites with cross-plane links. The argument has a default value of 1, meaning 1 out of every 1 satellites has cross-plane links, or a +grid network design. Setting the argument to 2 results in only half of the satellite have cross-plane links, 3 equals one-third, etc. To achieve the sparse network design with no cross-plane links, *crosslink-interpolation* is set to the number of satellites per plane +1.

Adding Network Designs

This simulator is built around the assumption that users may want to add custom network designs, like extended cross-plane links in a +grid (see Figure 4c). Therefore, adding new network designs to the model is fairly straightforward. For basic functionality, one of the existing network design functions in the Constellation class like *calculatePlusGridLinks()* can be used as a template and added as a new function to the class. A simulator can then call the new function and export GML files, but users will need to modify the Simulation class to enable visualization. This is also simple: add the call to the new network design function to the *updateModel()* function in the Simulation class.

Exporting Data

The simulation can be configured to export a GML file every time-step of simulation that contains all the data necessary to recreate the entire satellite network and ground stations. An example of the GML file structure is shown in Figure 15 for a simulation with two satellite nodes, one ground station (CNU) and one link (edge). The properties of the simulation used to generate the graph are listed at the top of the file, including the simulation time, and network design (connectionStrategy). Note that node position data is not included in the graph, because the edge distances are already provided. As the simulated network becomes very large, the exported GML files (encoded as plain text) can exceed a million lines and become challenging to open with most text editors. The GML files are generated using the NetworkX.*write-gml()* function, and therefore can be easily imported by an external script using the NetworkX.*read-gml()* function. In addition, users can initialize a new simulation from a GML file. For example, a user runs several simulations with different constellation sizes and exports a set of GML files for each. Later, if the user wishes to restart one of the simulations from a specific timestep, they can simply import the corresponding GML file into the simulator, rather than re-running the simulation from t=0 to that point. This feature also allows a user to take a GML file exported by the simulator and modify it with an external program (for example, remove the least used links as indicated by a traffic

model). The modified GML file can be imported by the simulator, and then run forward in time to see how the modified network evolves.

```
graph [
  numPlanes "1"
  numNodesPerPlane "2"
  planeInclination "65.0"
  semiMajorAxisMeters "6871000.0"
  minCommunicationsAltitudeMeters "100000"
  minSatElevationDegrees "40"
  simulationTime "10"
  connectionStrategy "SPARSE"
  node [
    id 0
    label "0" # The ID number of the first satellite
    planeNumber "0"
    offsetNumber "0"
  ]
  node [
    id 1
    label "1" # The ID number of the second satellite
    planeNumber "0"
    offsetNumber "1"
  ]
  node [
    id 2
    label "-1" # The ID number of the ground station
    placeName "CNU"
  ]
  edge [
    source 0
    target 1
    distance 13741998
  ]
]
```

Figure 15: GML file example

Simulation Class

The Simulation class wraps the Constellation class, and sets up VTK [29] render pipelines for visualizing the satellite network. The class contains global variables for setting the color, size, and opacity of visualized objects like Earth, satellites, and links. Functions encompass initializing a new Constellation object, or importing one from a GML file, and exporting images of the rendered visualization for creating animations (GIFs). It is designed to be initialized as an independent process and communicate with its host process with a Pipe object (from Python3 multiprocessing library). Running as an independent process keeps the host process responsive to user input, which is desirable for a GUI.

Visualization

The ability to interact with a 3D visualization of a satellite network orbiting Earth is a critical feature of the simulator. It enables the debugging of network designs that otherwise are only described by a large set of position data and links. For example, a network design may have a bug that creates links that pass through the center of the Earth. A problem like this is obvious by looking at a 3D visualization, but not by looking at large sets of positions and links. The visualization in the simulator is done with the VTK library, an API for OpenGL. The Simulation class generates several VTK render pipelines upon initialization: the Earth model, satellite point cloud, ground stations, inter-satellite links, satellite-ground links, and a shortest path. Each render pipeline can have its color, opacity, etc. adjusted by editing global variables at the top of the simulator class, and the different layers (satellites, links, ground stations, etc.) can be individually enabled or disabled. The pipelines are rendered in an interactive window that updates every time-step, and can be resized by the user. When presenting data on a satellite network, it is very helpful to show an animation of the network. Therefore the Simulation class includes a function for exporting a PNG file of the rendered model every time-step, that can be combined into a GIF or other animation. Most of the diagrams in this thesis like Figure 16 below, were generated in this manner.

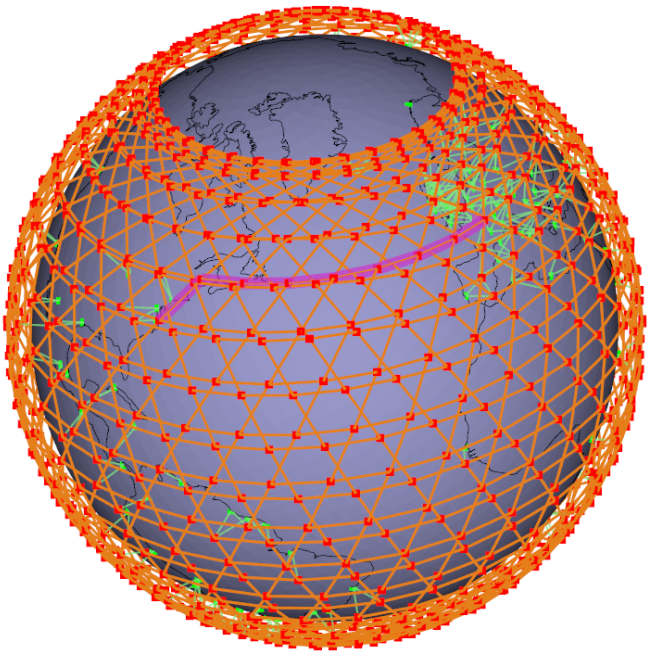


Figure 16: Example of interactive 3D render: red = satellites, green = ground stations, orange = inter-satellite links, purple = shortest path between CNU and London

GUI Class

The GUI class is based on Qt5[30], a modern, cross-platform GUI library. The interface offers controls to generate a new satellite network model, import one from a GML file, change network design, toggle image and GML export, and view performance data. The GUI also offers a function for choosing two ground stations to calculate the shortest path between, and display in the visualization. A screenshot of the GUI is provided in Figure 17.

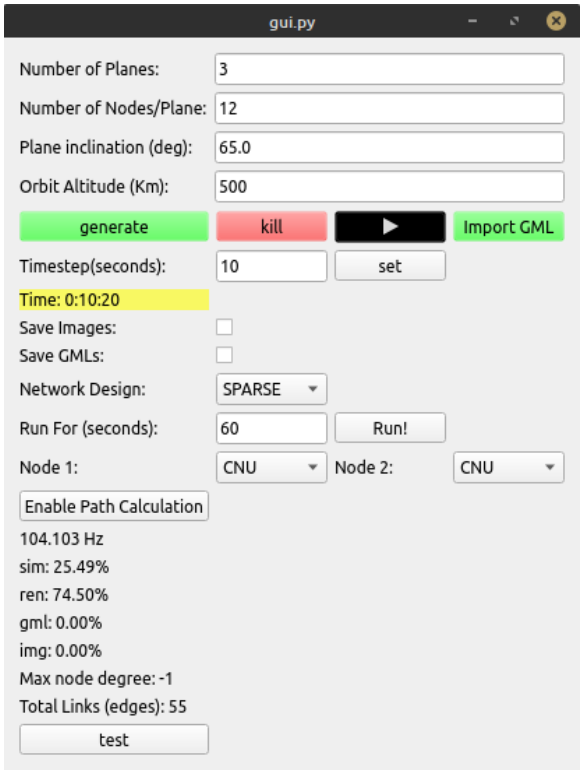


Figure 17: Example of control GUI

Performance

The actual performance of the simulation is important to its usability; if it takes hours or days to get a useful data set, users may give up or become frustrated. The following example gives an idea of the methods used to gain acceptable simulation performance.

When using the 'IDEAL' network design the distance from every satellite to every other satellite must be checked to determine the set of all valid links. This requires (in a naive implementation) iterating through the array of satellites n times for every time-step of simulation. Every iteration through the array means n memory accesses to the satellites-array, n euclidean distance calculations and n integer comparisons, resulting in a computational complexity of $O(n^2)$. The number of satellites in some proposed LEO satellite networks like Starlink, can approach 10,000 ($10000^2 = 100 \times 10^6$). Assuming the simulation is running on a recent x86 CPU at a moderate 4GHz, with standard DDR4 ram (burst size 8 64-bit words), we can make a rough estimate of the execution time. The first

step of each iteration is a memory access to get the position data for a satellite. Each index in the satellite-array described in Figure 11 is 22 bytes, and a DDR4 burst is 64 bytes, so with CPU caching, a DRAM read needs to be made about every third index. A typical ram access time is 10ns, so $10\frac{n}{3}$ ns per iteration, and $10\frac{n^2}{3}$ ns per simulation time-step is spent waiting for RAM. A 3D euclidean distance calculation, $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$, needs to be made at each index, and is made up of about 7 unique integer operations. Most CPUs have integer execution units that only take one clock, but there will also be an integer comparison each index, so we can round this to 10 CPU clocks per index. With a CPU frequency of 4GHz one clock cycle has a period of 0.25 ns, so calculation takes $0.25 * 10 * n^2$ ns per time-step. If the simulation code for ideal link calculation is translated to optimized machine code (as is done in this simulator using the Numba[31] library), with no overhead, the time needed to calculate ideal links for one timestep is $10\frac{n^2}{3} + (0.25 * 10 * n^2)$ ns. When n is 10,000 this gives 1.25 seconds, which is comparable to observed simulator performance. However, generic Python is not optimized machine code, it has lots of overhead, and it stores ints as objects that get scattered through memory (DRAM burst caching will not help). Therefore, a naive implementation that stored satellite data as a Python list of ints, takes 100 to 1000 times longer to calculate ideal links every time-step. At a factor of 100, for $n = 10000$, this would mean 125 seconds (2 minutes) per time-step, and make the simulation program quite tedious to use. As the example demonstrates, choosing data structures, variable types (integers vs floats), and using Numba to optimize functions (like ideal link calculation), is needed to meet requirement 2 (from Table 2).

Using GPU acceleration and multiprocessing was initially considered for improving performance, but was discarded. In practice, when exporting GML files and images every time-step, calculating ideal links in a large network takes only about a third of the total time. Simulation execution time becomes bottlenecked by factors like hard drive bandwidth, because writing a million+ lines of GML file every time-step is a significant operation.

Evaluation of Routing Performance

In order to demonstrate the developed simulation tool, latency performance of the three implemented linking strategies is compared using the average latency over a set of several station pairs. The pairs are mapped geographically to the locations of large cities. Although not a perfect match to a global map of internet usage, a set of links defined by the large population centers provides a reasonable approximation for the type of links seen in an operational satellite network. The tool outputs a network graph at user defined time steps directly to NetworkX. Latency between a set of ground points is calculated as the sum of signal propagation times ($\frac{distance}{c}$ seconds) from each hop along the shortest path between said points, calculated using NetworkX. The details of the analysis are discussed in the following chapter.

CHAPTER IV:
ANALYSIS

To demonstrate the functionality of the simulation tool, we consider a number of questions involving satellite network design and uses simulation data to present possible solutions. The first section looks at routing performance for different network designs, and the second compares routing performance for different sized constellations.

Comparison of Network Designs

Network design has a large impact on routing performance, which raises the question: How do +grid, sparse, and ideal network designs compare? Routing performance is measured by three factors in this analysis: latency, hop count, and time-between-path-changes or path stability. The three characteristics are measured in simulations of all three network designs for a small set of connections:

Table 3: Set of measured paths

Name	Location	Name	Location	great circle distance
CNU	(37.06 -76.49)	London	(51.53 -0.08)	5,985 km
CNU	(37.06 -76.49)	Los Angeles	(34.01 -118.41)	3,787 km
CNU	(37.06 -76.49)	Chicago	(41.83 -87.66)	1,117 km
CNU	(37.06 -76.49)	Moscow	(55.75 37.6)	7,956 km
CNU	(37.06 -76.49)	Sydney	(-34.0 151.0)	15,733 km
CNU	(37.06 -76.49)	Cape Town	(-33.91 18.36)	12,592 km
CNU	(37.06 -76.49)	Tokyo	(35.66 139.75)	11,132 km

Every simulation time-step, the shortest path for each connection above is calculated using Dijkstra's algorithm, where link weight is the link distance. Latency is measured as $\frac{pathlength}{c}$ seconds, and path stability is measured as the time interval between changes in the shortest path. In Figure 18, data for a 40:40 constellation at an altitude of 500km and inclination

of 60^o is illustrated, while Table 4 provides statistics. An ideal network design provides the lowest possible latency across all measured paths, followed by +grid. The sparse network design has higher latency over all paths, and generally more variance. The higher latency of the sparse network design is clearest in east-west connections like CNU - Los Angeles, where the path can only move between orbital planes by relaying through a ground station. All three network designs exhibit periodic spikes in latency, usually indicating a change in one of the two satellite to groundpoints connections. In other words, the first hop and last hop in a connection usually change dramatically when a satellite goes out of range of a groundstation, and that groundstation must switch to another satellite. Between these sharp transitions, the latency tends to gradually change, as the path stretches or compresses due to satellite movement. This periodic behavior can be seen quite clearly in the +grid network design CNU-London connection, where both the minimum and maximum latency values repeat between sharp changes in latency caused by groundpoints jumping between satellites. In fact the periodic, sharp transitions in latency can be seen in all of the measured connections, because all of the ground stations see satellites moving across the sky and going in and out of range at almost the same rate (minor differences due to earth's rotation). Also, less pronounced transitions occur when satellite to satellite paths change. These smaller changes appear more in longer connections, like CNU-Tokyo, because as the number of satellites in the shortest path increases, the likelihood of that path changing also increases.

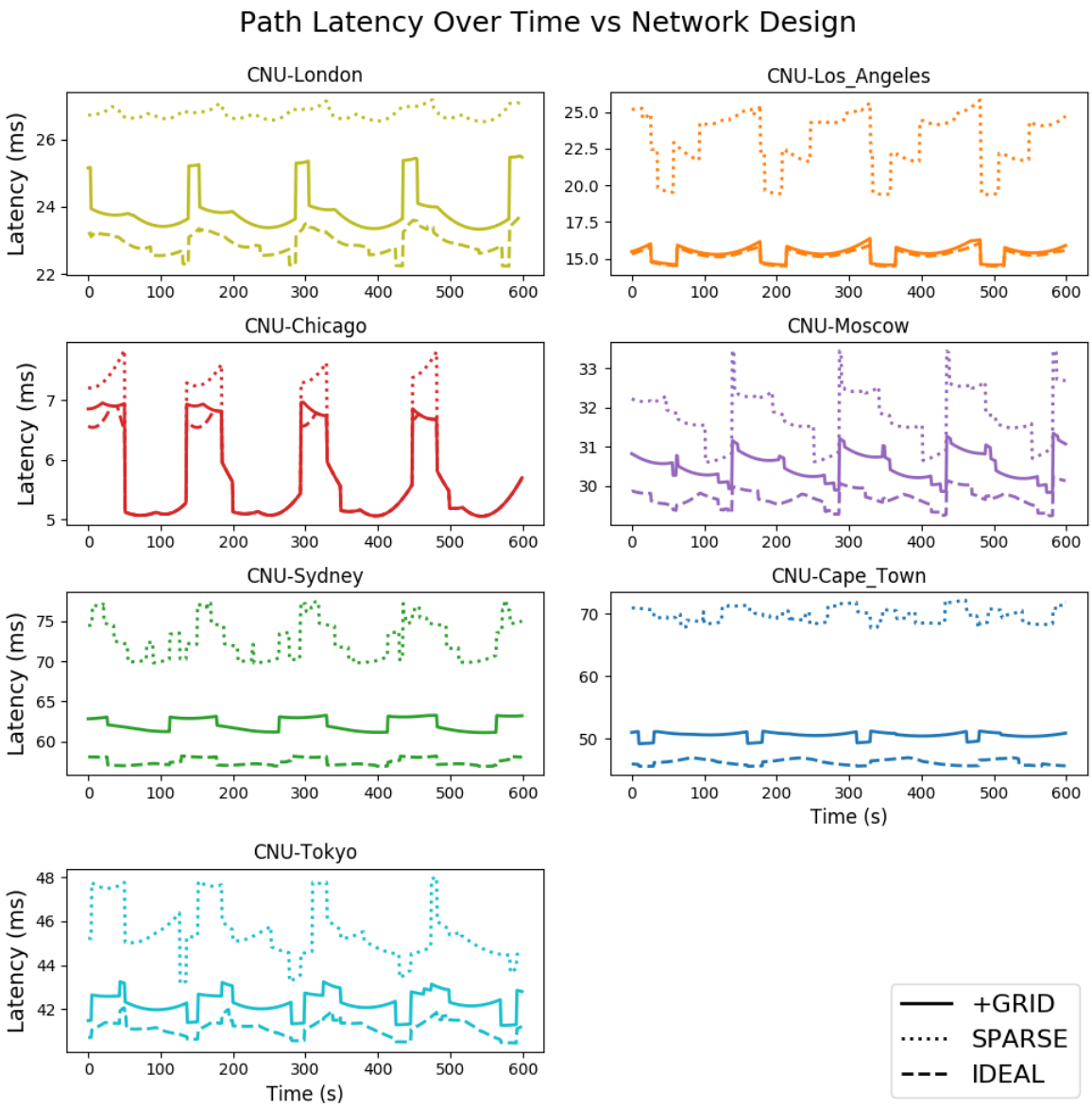


Figure 18: Comparing latency between network designs over simulation steps

Table 4: Latency statistics for 600 time-steps of one second each, all values in milliseconds

Satellite Latency, CNU-		London	L. A.	Chicago	Moscow	Sydney	C. T.	Tokyo
Average	IDEAL	22.9	15.2	5.64	29.6	57.4	46.2	41.1
	+GRID	23.8	15.3	5.68	30.5	62.1	50.5	42.2
	SPARSE	26.7	23.0	5.84	31.7	72.6	69.7	45.4
Min	IDEAL	22.2	14.4	5.04	29.2	56.9	45.6	40.4
	+GRID	23.3	14.5	5.04	29.8	61.1	49.2	41.2
	SPARSE	26.5	19.3	5.04	31.7	72.6	67.7	43.1
Max	IDEAL	23.7	16.0	6.90	30.2	58.2	47.0	42.0
	+GRID	25.5	16.3	6.96	31.3	63.2	51.2	43.2
	SPARSE	27.1	25.8	7.84	33.4	77.6	72.1	48.0
Median	IDEAL	22.8	15.2	5.19	29.6	57.2	46.2	41.1
	+GRID	23.6	15.4	5.19	30.4	61.8	50.6	42.2
	SPARSE	26.7	24.1	5.19	31.6	72.1	69.7	45.2
Great Circle, $\frac{distance}{c}$		20.0	12.6	3.7	26.5	52.5	42.0	37.1

The plots in Figure 19, illustrate hop counts for the same set of network designs, with statistics presented in Table 5. Aside from the very short CNU - Chicago path, the ideal network design results in far fewer hops. This is due to both the +grid and sparse network designs being limited to connections between adjacent satellites. In the ideal design, connections can 'skip' many adjacent satellites, resulting in lower hop counts particularly for long distance connections. In addition, periodic variations in hop count for the +grid and sparse network designs occur for the same reason as the periodic sharp transitions in latency. When ground stations switch between first or last hop satellites, the sharp change in path length often results in a change in hop count. However, in the ideal network design, periodic changes in hop count are less apparent, because the longer links can absorb the changes in path length more readily.

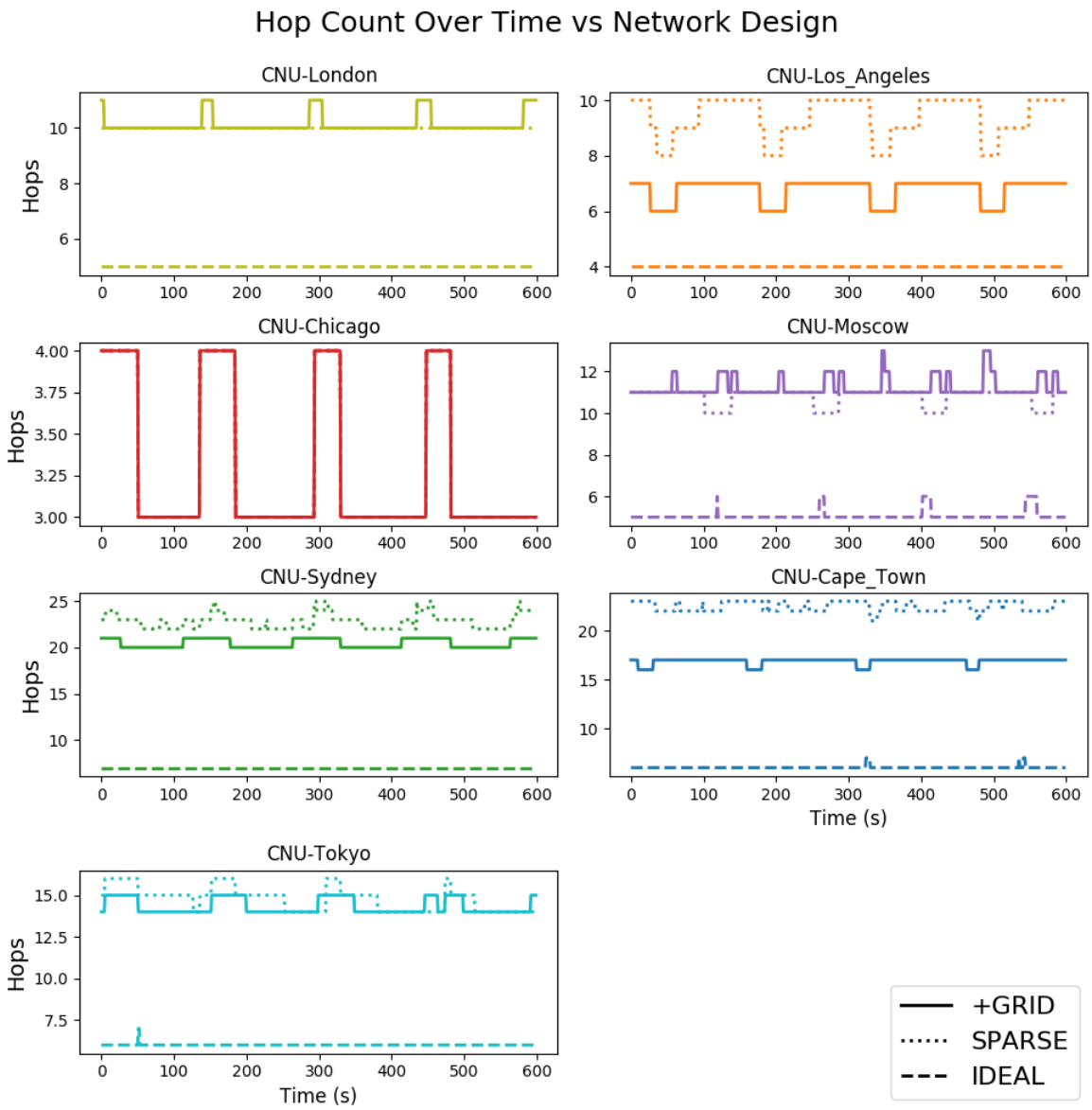


Figure 19: +grid network design

Table 5: Hop count statistics for 600 time-steps of one second each

Satellite Hops, CNU-		London	L. A.	Chicago	Moscow	Sydney	C. T.	Tokyo
Average	IDEAL	5.00	4.00	3.28	5.06	7.00	6.02	6.00
	+GRID	10.1	6.77	3.28	11.2	20.4	16.9	14.3
	SPARSE	10.0	9.39	3.28	10.8	22.8	22.5	14.8
Min	IDEAL	5	4	3	5	7	6	6
	+GRID	10	6	3	11	20	16	14
	SPARSE	10	8	3	10	22	21	14
Max	IDEAL	5	4	4	6	7	7	7
	+GRID	11	7	4	13	21	17	15
	SPARSE	10	10	4	11	25	23	16
Median	IDEAL	5	4	3	5	7	6	6
	+GRID	10	7	3	11	20	17	14
	SPARSE	10	10	3	11	23	23	15

Although the ideal network design offers lower latency and hop counts, it suffers in path stability. Figure 20 compares the average time between path changes for the three network designs, where a longer time indicates better stability. Here the +grid design offers significantly better stability than sparse and ideal, with an average of 30.6 seconds between path changes. In comparison, the average time between changes in the ideal design is 15.8 seconds, and 17.9 seconds in the sparse design.

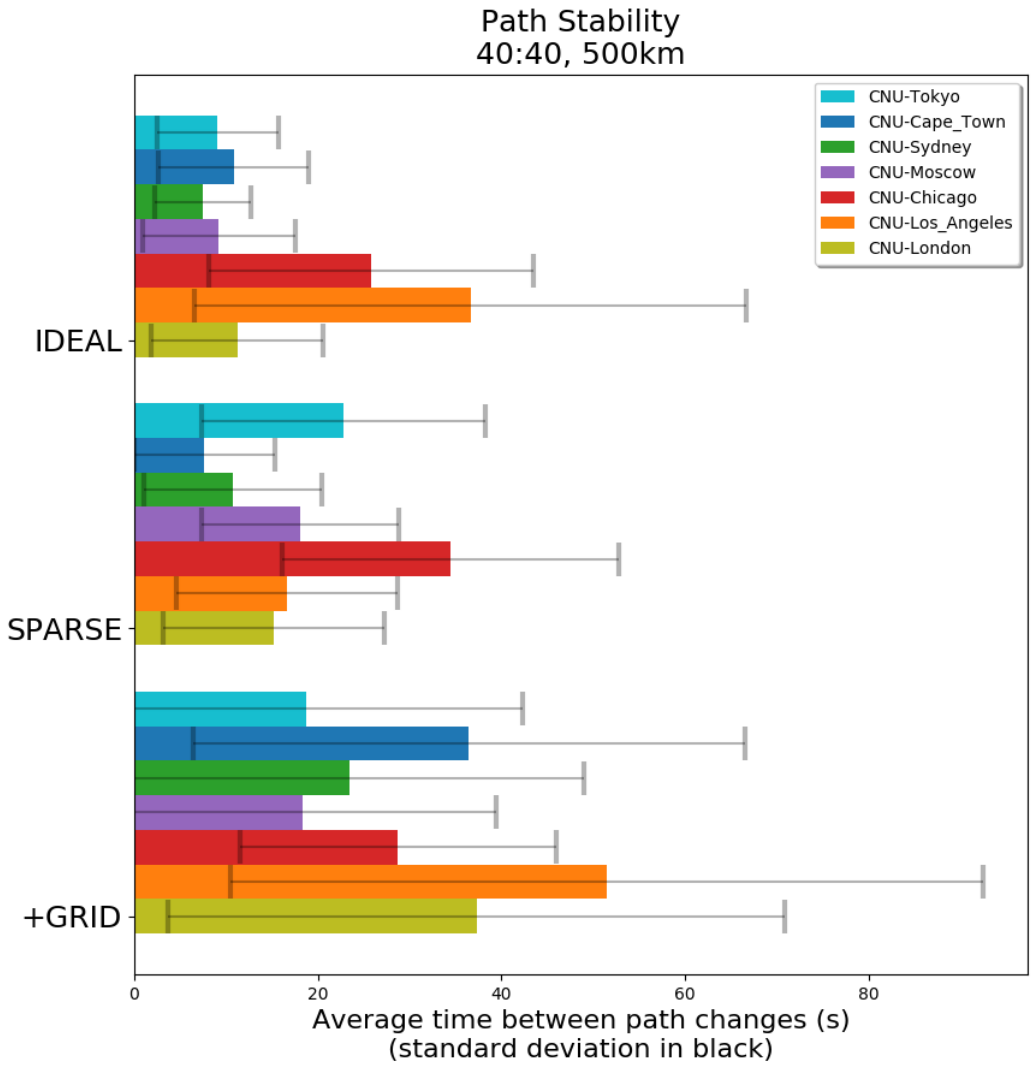


Figure 20: Path stability between network designs (higher is more stable)

Density

It may be desirable to optimize the density of a satellite network for the best *cost - (routing)performance*, all other things (altitude, inclination, etc.) being equal. By itself the question "what density gives the best cost-performance?" is vague; as in the previous section, better 'performance' is defined by lower latency, lower hop count, and longer time between path changes (path stability). Cost likely scales with constellation size, with more satellites being more expensive. Therefore a better question may be: What constellation

density provides the best compromise between cost, low latency, and path stability? To answer, we collect data from a range of network densities: 35 symmetrical (number of planes = satellites per plane) constellations between 25:25 and 60:60. Figure 21 provides an illustration of several constellation densities. All constellations have the same altitude of 600km, inclination[3] of 60^o, and are connected with a +grid network design. Each constellation is simulated for 600 seconds, with a time-step of 1 second, and records the same set of shortest paths (Table 3) discussed in the previous section.

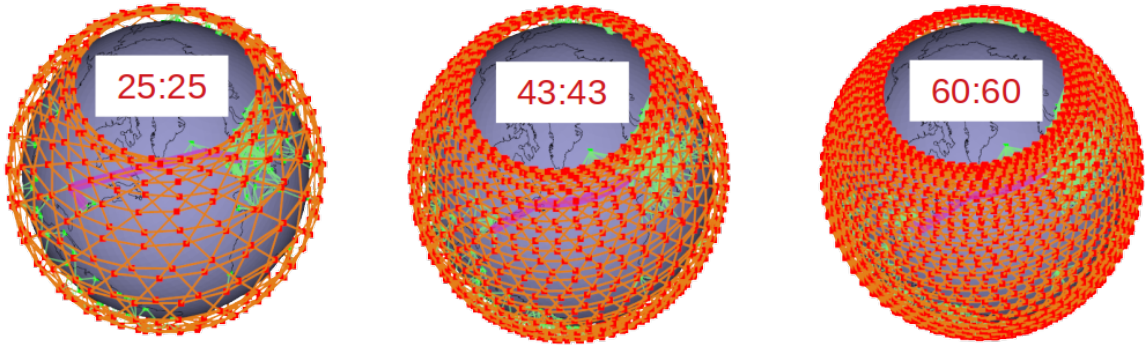


Figure 21: Range of constellation densities, +grid network design

It should be noted that generating the data described above is not a trivial operation. Simulating 35 constellations, for 600 time-steps each, and running Dijkstra's for multiple paths every time-step takes over two hours if run as a single thread. To speed up such a lengthy operation, the project includes a script, 'gen_data_parallel.py', that uses Python's multiprocessing library to parallelize the simulations and cut execution time down to about 7 minutes on a 16-thread, 4.2 GHz CPU. If the network design used in the simulation is changed to ideal, this same operation running in parallel x16 can take over five hours; running on a more common 4-thread, 3GHz CPU can stretch the execution time to over a day.

Using the CNU - London connection as an example, the average latency vs constellation density is shown in in Figure 22. Average latency decreases as density increases, but only by

[3]600km and 60^o are similar values to the in development Starlink constellation

a slight 2.5 ms from 25:25 to 60:60 constellations. Hop count (Figure 23) increases linearly with density as expected.

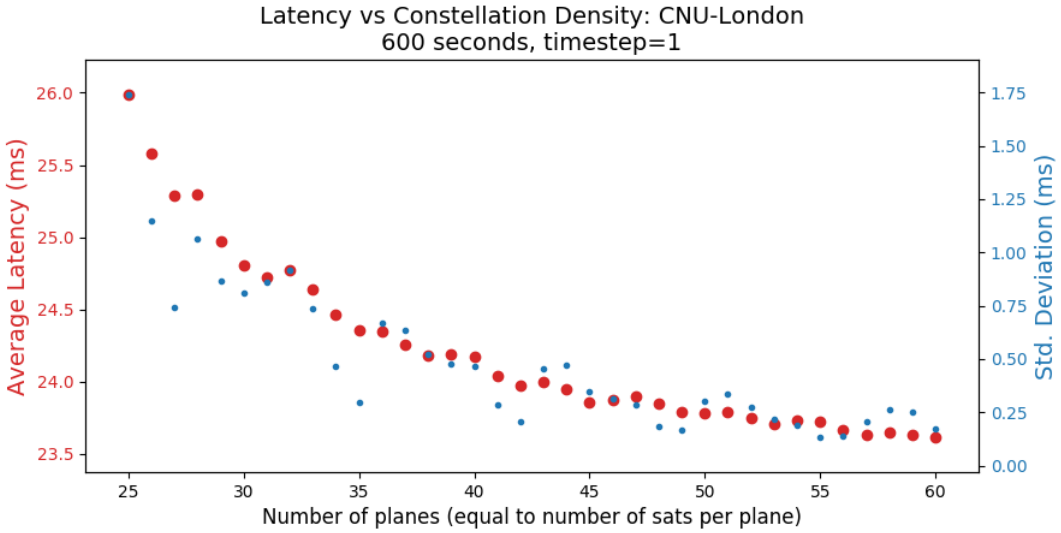


Figure 22: Latency vs Density

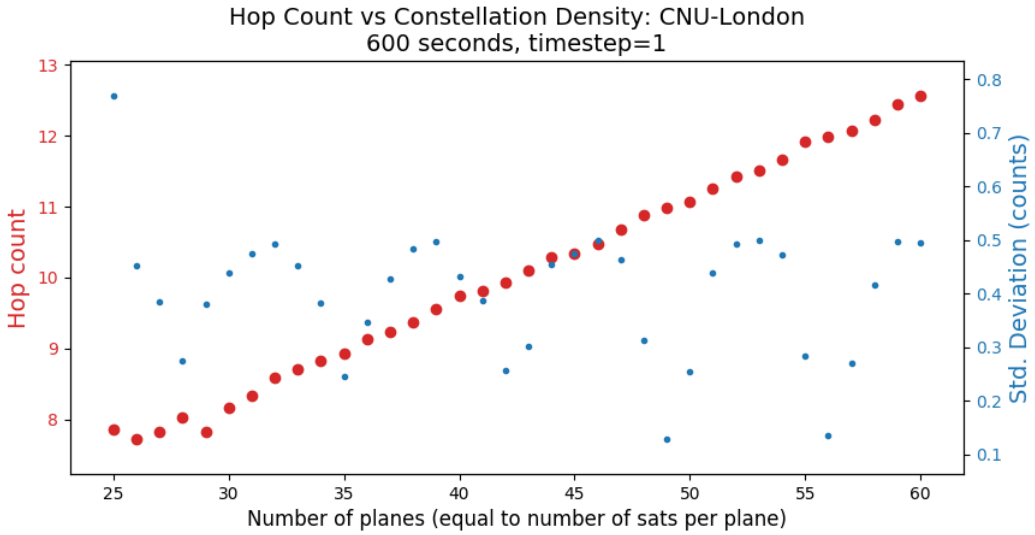


Figure 23: Hop Count vs Density

The average time between path changes vs constellation density in Figure 24 is quite interesting. Longer times between path changes, or higher path stability, is desirable in a network because it minimizes routing updates. Figure 24 shows that path stability generally decreases as density increases, but there is a local maximum at constellation size

39:39. This indicates that a 39:39 constellation with the specific simulated characteristics (600km altitude, 60^o inclination, etc.) may be a good compromise between low latency, high path stability, and low cost (larger networks being more expensive).

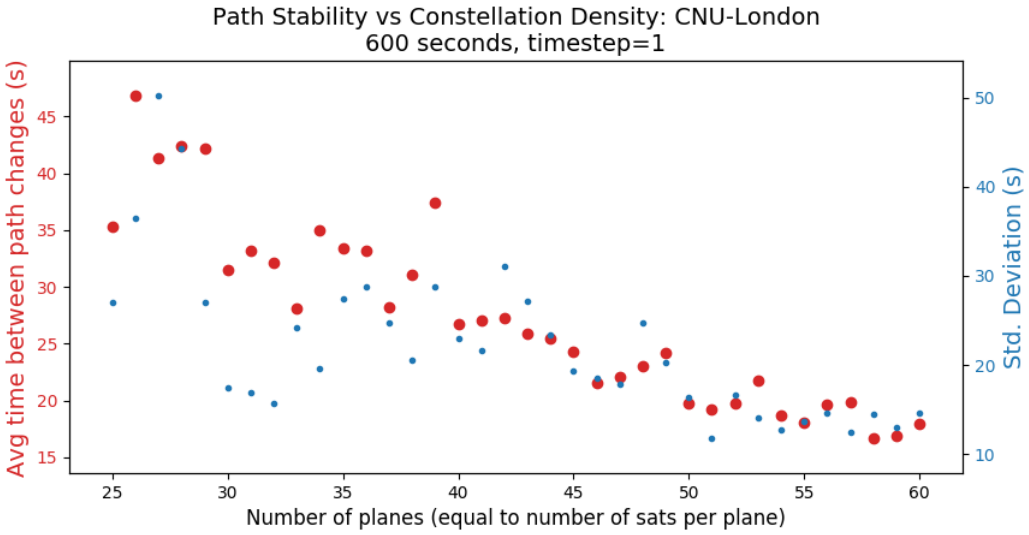


Figure 24: Path Stability vs Density

However, analyzing a single connection (CNU - London), is not enough to characterize an entire constellation with respect to density. Here we take the same simulation data, but look at a different connection: CNU - Sydney. Figures 25 and 26, graphing latency and hop count respectively, are quite similar in shape to the CNU - London connection. However, the path stability graphed in Figure 27 is quite different. Unlike the CNU - London connection there does not appear to be a decreasing trend in path stability as the density increases. Surprisingly, the path stability for CNU - Sydney at the 39:39 constellation is on the higher end of the range, similar to the CNU - London connection. However, unlike the CNU - London connection, the path stability at this density does not appear to be a local maximum. Rather, there is no readily apparent trend in path stability for the CNU - Sydney connection.

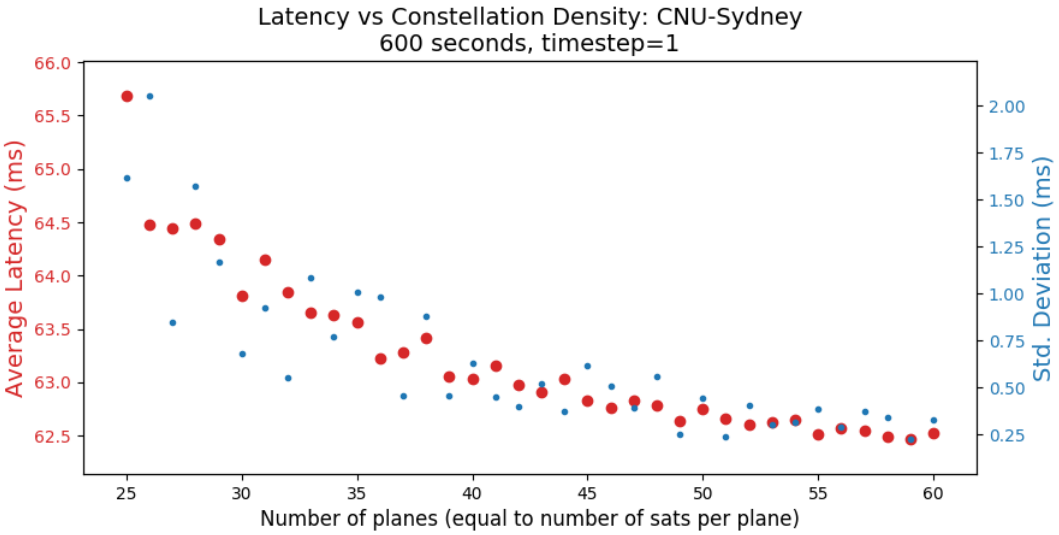


Figure 25: Latency vs Density

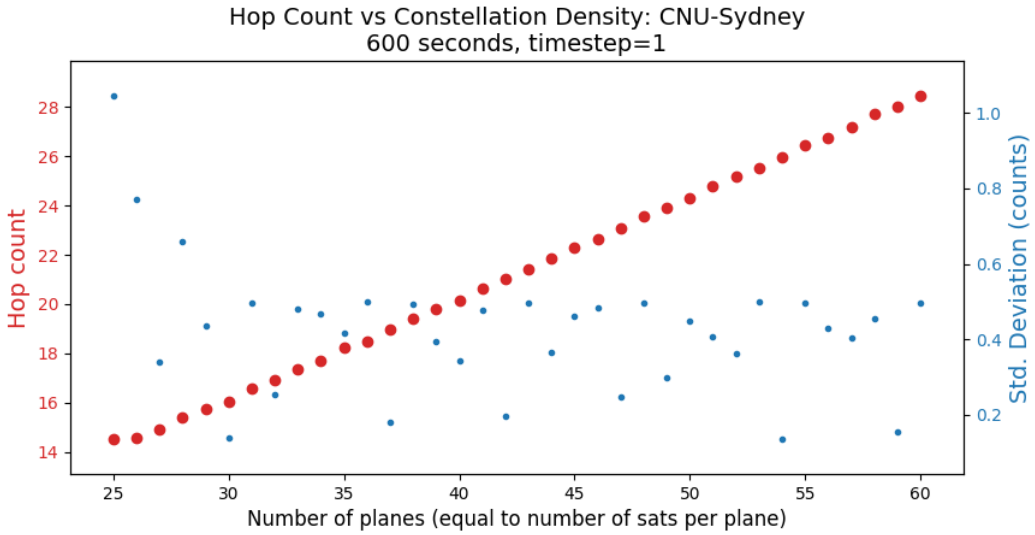


Figure 26: Hop Count vs Density

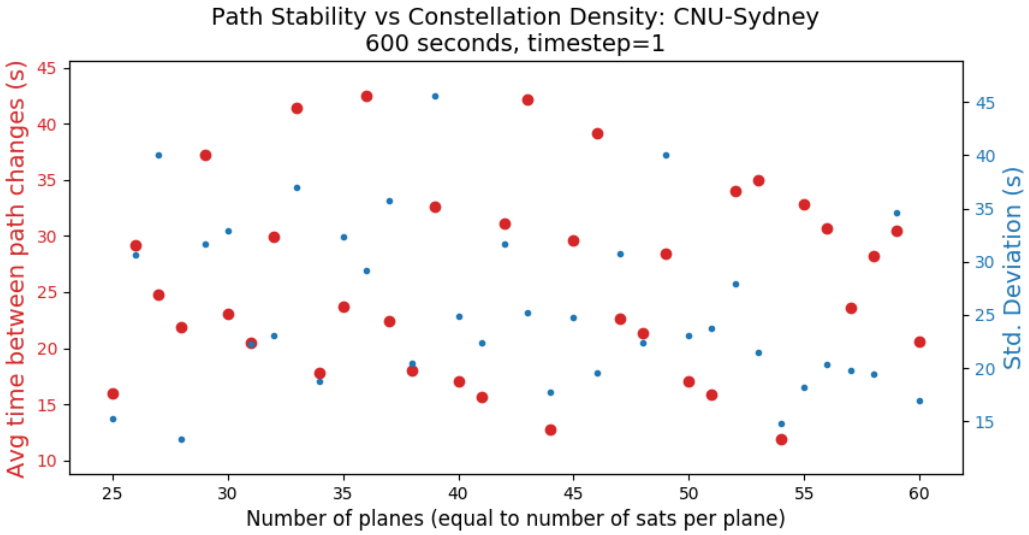


Figure 27: Path Stability vs Density

CHAPTER V:
CONCLUSION AND OUTLOOK

In conclusion, the satellite constellation and network simulation tool presented in this thesis was a successful software engineering effort aimed at contributing to the satellite network research community. It enables the simulation of an arbitrarily sized and structured satellite constellation, and can apply one of three different network designs. Ground stations or user terminals can be added to the network, and follow Earth's rotation. The simulation can be displayed in an interactive 3D model, showing network connections, satellites, groundpoints, and a simple model of the Earth. A user can set the simulation to export images and GML files of the network, and easily collect large data sets. The code has been carefully designed with performance in mind and takes advantage of JIT compilation to bring C or FORTRAN like levels of speed. Finally, the tool is documented and provides a user guide to help others. Overall the tool fulfills the requirements presented in Tables 1 & 2, and will hopefully be a useful contribution to the research community.

As demonstrated in the previous chapter, the simulator by itself is suitable for collecting large amounts of network graph data to analyze routing performance. In addition, the GUI and visualization functions allow users to quickly generate constellations and see the structure and connections evolve over time. As discussed in the introduction, a tool that provides these capabilities for satellite network research was not available until now. Therefore in terms of outlook, the simulation tool presented in this thesis has the potential to be very useful. Several research questions are analyzed in Chapter 4 that focus on comparing routing performance. However this is not the only use case for this tool.

Rather than a small list of paths between CNU and some large cities, a user could import a much larger number of groundpoints based on a population density map. Take the in-development OneWeb and Starlink constellations as an example; both appear to place emphasis on large numbers of low cost user terminals to provide broadband internet access. In this case, one could ask the question of "how does a satellite network handle areas of very high user terminal density, and what is the throughput of the network?". Using this

simulator tool, one could take a series of network graphs of a simulation that includes a large number of user terminals or groundpoints and add some properties to the satellites like queueing delay, using a tool like NetworkX or Network Simulator 3 (NS-3). The resulting model could be used to study how a satellite network copes with a population density based load or network characteristics like throughput.

Another example of further work with this simulation tool is adding new network designs. As seen in the analysis chapter, +grid and sparse suffer from high hop counts compared to an ideal network design that allows longer connections. However the ideal network is far too complex to implement in a real satellite constellation. A user could implement a new network design to improve the performance of +grid, while maintaining a realistic number of inter satellite links. Rather than always connecting to the satellites in the first adjacent planes, an extended +grid network design might implement crosslinks that connect to a given satellite's third adjacent planes. The longer links would possibly decrease hop count, and slightly improve latency in longer connections. With the developed simulator tool, adding a new network design, visualizing it, and studying changes in routing performance is fairly easy.

LITERATURE CITED

[1] Hughes Network Systems, LLC. (2020). How it works, [Online]. Available: https://www.hughesnet.com/about (visited on 07/21/2020).

[2] Viasat Inc. (2020). Viasat, [Online]. Available: https://www.viasat.com/ (visited on 07/22/2020).

[3] International Telecommunications Union. (2003). One-way transmission time (ITU-T G.114).

[4] T. Greg Ritchie, "Why low-earth orbit satellites are the new space race," Jul. 10, 2020. [Online]. Available: https://www.bloomberg.com/news/articles/2019-08-09/why-low-earth-orbit-satellites-are-the-new-space-race-quicktake (visited on 07/22/2020).

[5] Starlink. (2019). Starlink, [Online]. Available: https://www.starlink.com/ (visited on 11/12/2019).

[6] Amazon. (2020). Project kuiper, [Online]. Available: https://www.amazon.jobs/en/teams/projectkuiper (visited on 05/15/2020).

[7] OneWeb. (2019). Oneweb, [Online]. Available: https://www.oneweb.world/ (visited on 05/15/2020).

[8] International Telecommunications Union, "Measuring digital development, facts and figures 2019," 2019.

[9] D. Bhattacherjee, W. Aqeel, I. N. Bozkurt, A. Aguirre, B. Chandrasekaran, P. B. Godfrey, G. Laughlin, B. Maggs, and A. Singla, "Gearing up for the 21st century space race," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, ser. HotNets '18, Redmond, WA, USA: Association for Computing Machinery, 2018, pp. 113–119, ISBN: 9781450361200. DOI: 10.1145/3286062.3286079. [Online]. Available: https://doi.org/10.1145/3286062.3286079.

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959, ISSN: 0029-599X. DOI: 10.1007/BF01386390. [Online]. Available: https://doi.org/10.1007/BF01386390.

[11] A. Jamalipour, M. Katayama, and A. Ogawa, "Traffic characteristics of leos-based global personal communications networks," *IEEE Communications Magazine*, vol. 35, no. 2, pp. 118–122, 1997.

[12] (2020). Satellite Network Simulator 3, [Online]. Available: http://satellite-ns3.com/ (visited on 02/08/2020).

[13] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.

[14] B. S. Kempton. (2020). SImulator for Large LEO-Satellite Communication NetworkS (SILLEO-SCNS), [Online]. Available: https://github.com/Ben-Kempton/SILLEO-SCNS.

[15] (2018). GMAT (General Mission Analysis Tool), [Online]. Available: http://gmatcentral.org/ (visited on 02/08/2020).

[16] L. Markley, "Kepler equation solver," *Celestial Mechanics and Dynamical Astronomy*, vol. 63, pp. 101–111, Mar. 1995. DOI: 10.1007/BF00691917.

[17] C. Henry, "Spacex says more starlink orbits will speed service, reduce launch needs," *SpaceNews*, Sep. 7, 2019. [Online]. Available: https://spacenews.com/spacex-says-more-starlink-orbits-will-speed-service-reduce-launch-needs/ (visited on 11/13/2019).

[18] Iridium Communications Inc. (2019). Iridium global network, [Online]. Available: https://www.iridium.com/network/globalnetwork/ (visited on 11/12/2019).

[19] D. Bhattacherjee and A. Singla, "Network topology design at 27,000 km/hour," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, ser. CoNEXT '19, Orlando, Florida: Association for Computing Machinery, 2019, pp. 341–354, ISBN: 9781450369985. DOI: 10.1145/3359989.3365407. [Online]. Available: https://doi.org/10.1145/3359989.3365407.

[20] J. Buck and D. Washington, "Nasa laser communication system sets record with data transmissions to and from moon," Oct. 2013. [Online]. Available: https://www.nasa.gov/press/2013/october/nasa-laser-communication-system-sets-record-with-data-transmissions-to-and-from/.

[21] M. Wade, *Intelsat 1*, 1997-2020. [Online]. Available: http://www.astronautix.com/i/intelsat1.html.

[22] I. Jacobs, L.-N. Lee, A. Viterbi, R. Binder, R. Bressler, N.-T. Hsu, and R. Weissler, "Cpoda - a demand assignment protocol for satnet," in *Proceedings of the Fifth Symposium on Data Communications*, ser. SIGCOMM '77, Snowbird, Utah, USA: Association for Computing Machinery, 1977, pp. 2.5–2.9, ISBN: 9781450374064. DOI: 10.1145/800103.803329. [Online]. Available: https://doi.org/10.1145/800103.803329.

[23] A. Ferreira, J. Galtier, and P. Penna, "Topological design, routing, and handover in satellite networks," *Handbook of Wireless Networks and Mobile Computing*, pp. 473–493, Feb. 2002. DOI: 10.1002/0471224561.ch22.

[24] E. Ekici, I. F. Akyildiz, and M. D. Bender, "A distributed routing algorithm for datagram traffic in leo satellite networks," *IEEE/ACM Transactions on Networking*, vol. 9, no. 2, pp. 137–147, 2001.

[25] Z. Gao, Q. Guo, and P. Wang, "An adaptive routing based on an improved ant colony optimization in leo satellite networks," in *2007 International Conference on Machine Learning and Cybernetics*, vol. 2, 2007, pp. 1041–1044.

[26] E. Papapetrou and F. Pavlidou, "Distributed load-aware routing in leo satellite networks," in *IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference*, 2008, pp. 1–5.

[27] M. Handley, "Using ground relays for low-latency wide-area routing in megaconstellations," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, ser. HotNets '19, Princeton, NJ, USA: Association for Computing Machinery, 2019, pp. 125–132, ISBN: 9781450370202. DOI: 10.1145/3365609.3365859. [Online]. Available: https://doi.org/10.1145/3365609.3365859.

[28] S. Czesla, S. Schröter, C. P. Schneider, K. F. Huber, F. Pfeifer, D. T. Andreasen, and M. Zechmeister, *PyA: Python astronomy-related packages*, Jun. 2019. ascl: 1906.010.

[29] B. L. Will Schroeder Ken Martin, *The Visualization Toolkit (4th ed.)* Kitware, 2006, ISBN: 978-1-930934-19-1.

[30] Riverbank Computing Limited, *PyQt5*, 2019.

[31] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, ser. LLVM '15, Austin, Texas: Association for Computing Machinery, 2015, ISBN: 9781450340052. DOI: 10.1145/2833157.2833162. [Online]. Available: https://doi.org/10.1145/2833157.2833162.