Interactive Satellite Megaconstellation Simulator

Clara Martins, Daniel Monteiro and Gonçalo Pascoal
M.Sc. Informatics and Computing Engineering
Faculty of Engineering, University of Porto
Porto, Portugal
{up201806528, up201806185, up201806332}@edu.fe.up.pt

***Abstract*—The study of cooperative groups of connected satellites, known as satellite constellations, is becoming increasingly relevant due to the transition to low-earth orbit (LEO) techniques. We propose a system to efficiently simulate these megaconstellations. Orbital parameters are used to model satellite motion, and different strategies can be employed to control how inter-satellite links are established. In addition, satellite failures can be automatically simulated or manually induced by the user. The core simulation is supported by a 3D interactive visualization tool and a real-time plotting module. We review the behavior of existing constellations and explore ways to improve network performance and reliability by adjusting the connection strategies. From our descriptive scenarios, we concluded that the Starlink constellation benefits east-west paths, while the OneWeb constellation favors north-south routes due to its near-polar orbits. Using the grid link strategy generally led to effective, consistent networks which seemed reasonably resistant to failures, while the nearest-neighbor approach delivered unpredictable, highly variable results.**

***Index Terms*—interactive visualization, modeling, satellite megaconstellations, simulation**

I. Introduction

Satellite constellations are groups of interconnected artificial satellites. These satellites can be employed for a variety of purposes, such as facilitating navigation and communication, performing Earth-observation tasks, or providing internet access. Using multiple interconnected satellites improves service coverage, with a large enough constellation providing near-global or worldwide coverage. Megaconstellations refer to networks consisting of a large number of interlinked satellites, mainly intended to provide internet services to users with poor internet connectivity.

The growing affordability of consumer-grade satellite communication terminals has caused a change in the telecommunications industry, causing it to go through a transitional phase. It is shifting away from geostationary satellites towards low earth orbit (LEO) constellation architectures. As LEO constellations fill the low-earth orbits, it is crucial to consider the reliability and efficiency of such constellations, as inadequate management of this resource might cause harmful effects [1]. To make things worse, satellites in LEO cover a much smaller area on the Earth's surface than those on geostationary orbits. Therefore, achieving the same coverage requires a much larger number of satellites. Consequently, studying these constellations and their capabilities and limitations is fundamental.

With this in mind, we aim to develop a system that can efficiently simulate large satellite constellations. In addition to exploring different strategies for establishing links between satellites, we wish to examine the impact of satellite failures in an effort to improve the connectivity and performance of the resulting networks. Employing simulation is advantageous in this case since it allows us to evaluate hypotheses in a controlled manner and study scenarios in a safe, virtual environment while still using data from real-world constellations. We also plan to implement a three-dimensional interactive visualization component and real-time plotting of relevant metrics.

The rest of this paper is divided into four major sections. Section II outlines the project's methodology, formalizes the problem, explains how it can be modeled, and describes our system's architecture and components. Afterward, in Section III, we present the experiments conducted using our simulation environment and discuss the implications of their results. Section IV discusses other tools and simulators. Lastly, Section V summarizes our findings and discusses future directions for research.

II. Methodology

In this section, we formally characterize the simulation problem to tackle, present the chosen modeling approach, and explain the architecture and central components of the developed system.

A. Problem Formalization

A satellite constellation is composed of several orbital planes, set at an equal inclination and spaced equidistantly. Satellites are evenly distributed between the orbital planes and are spaced equidistantly within the same plane. Satellites establish links between themselves to transmit data across the globe, with individual satellites having a fixed maximum number of active connections. Malfunctioning or failed satellites cannot be used for communications and could thus create vulnerable points in the network, hindering performance and connectivity.

Satellites move in fixed, circular orbits around Earth. The Keplerian orbital elements [2] are a suitable set of parameters to define these orbits, as they abstract away details concerning underlying gravitational forces, focusing purely on the characteristics of the orbits themselves. The six orbital elements are:

- Eccentricity ($e \in \mathbb{R}_0^+$), which is associated with the shape of the orbit. Higher values result in increasingly elliptical

orbits, whereas a value of 0 denotes a perfectly circular orbit;
- Semimajor axis ($a \in \mathbb{R}_0^+$), corresponding to the sum of the periapsis and apoapsis distances divided by two (distance between the centers of the two bodies at the closest and farthest points of the orbit, respectively). For circular satellite orbits, it is equal to the Earth's radius plus the orbiting altitude;
- Inclination ($i \in [0, \frac{\pi}{2}]$), which is the angle between the orbital plane and a reference plane (such as the Earth's equatorial plane) and is the same for every orbital plane;
- Longitude of the ascending node ($\Omega \in [0, 2\pi]$), corresponding to the horizontal orientation of the orbit with respect to a reference direction, which is specific per orbital plane. The term "ascending node" refers to the point in the orbit where the celestial body transitions from the southern to the northern hemisphere;
- Argument of periapsis ($\omega \in [0, 2\pi]$), corresponding to the initial orientation of each satellite within the orbital plane. Since satellites from the same orbital plane are evenly spaced, the difference between the argument of periapsis of consecutive satellites is the same;
- True anomaly ($\nu(t) \in [0, 2\pi]$), an angle that corresponds to the position of the satellite at time t relative to its starting position.

Figure 1 presents a diagram explaining the different orbital elements of a celestial body in circular orbit.

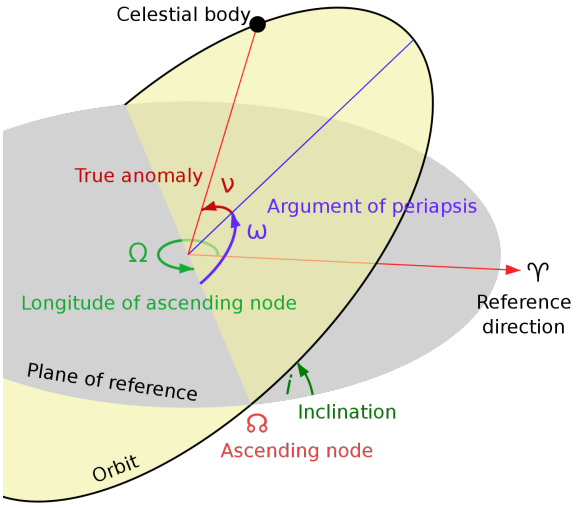


Fig. 1: Orbital elements of a celestial body [3]

By specifying the number of orbital planes, the number of satellites in each plane, the orbiting altitude, and inclination, we can precisely define the orbit of each satellite using the Keplerian elements.

Another common way of describing satellite constellations, initially proposed by John Walker [4], uses the notation **i: t/p/f**, where:

- **i** corresponds to the inclination;
- **t** refers to the total number of satellites in the constellation;
- **p** refers to the number of orbital planes;
- **f** relates to the offset between corresponding satellites in adjacent orbital planes, calculated using the formula $360 \times f/t$. This phase difference has a vital role in preventing collisions between satellites, but information regarding the specific phasing used in particular constellations is challenging to find.

The links established between satellites form an undirected graph, where edge weights correspond to the distances between connected satellites. We aim to explore ways of maximizing the connectivity of this graph and minimizing round-trip delays between various locations across the planet.

B. Modeling Approach

The problem of simulating large satellite constellations can be represented by a stochastic (due to the randomness associated with satellite failures), dynamic (since the system is examined at multiple points in time) model. The movement of the satellites can be described as a continuous process, but the links established between them can be modeled in a discrete manner.

We initially planned to adopt a decentralized, agent-based approach but later recognized that this would incur a substantial amount of additional complexity, making it inviable given the project's scope. As such, links between satellites are established through a central control system rather than through inter-satellite communication and negotiation. Viewed through a decision-support lens, we consider descriptive models to be the most applicable (and, to a lesser extent, speculative models).

Satellites can exist in two states: active or inactive. These states only control their ability to establish connections, not their movement. The input variables of the system include:

- Orbiting altitude;
- Number of orbital planes;
- Satellites per orbital plane;
- Phasing between planes;
- Maximum links per satellite;
- Satellite failure probabilities;
- Connection strategy.

The output variables are the positions of the satellites and the links established between them at each instant. Since we plan to model real-world constellations, we treat the variables describing the structure of the constellation as uncontrollable and focus on tuning the parameters related to the link strategy.

C. Architecture Overview

The simulation system we developed consists of three major components: one for modeling and simulation of the satellite constellation, one for real-time interactive visualization, and another for plotting and statistics. Figure 2 shows a high-level diagram of the simulator architecture, including its main elements and the possible interactions between them.

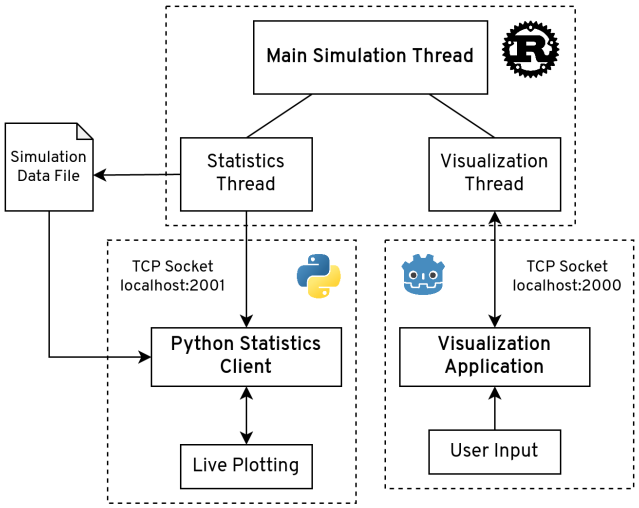


Fig. 2: Architectural diagram of the simulation system

Within the main application, in addition to the simulation thread, there are two auxiliary threads for handling interactions between the backend and supporting applications, such as the visualization and statistics components. Communication is performed through JSON messages sent using local TCP sockets. It is also possible to save simulation data to a file that the statistics component can parse. Note that visualization is only available when the simulation is running in real time.

The following subsections describe the functionality of each component in greater depth.

D. Simulation Component

When planning the course of the project and determining the set of languages and technologies to employ, we deemed existing simulation frameworks unsuitable for our goals, primarily due to the lack of support for interactive three-dimensional visualization. As such, our simulator was developed from the ground up using general-purpose libraries and tools.

We originally began developing this component using Python but quickly ran into performance constraints that led us to re-implement the project using a compiled language, specifically Rust. Rust is a low-level, strongly-typed language focused on speed, safety, and concurrency. Designed for systems programming and performance-critical applications, it constitutes an alternative to languages such as C/C++, with the main advantage of improved memory management.

1) Data Structures and Simulation Loop: The core of our application revolves around a few key data structures which implement our model and provide useful methods for collecting statistics and updating the simulation state.

Listing 1 contains the Rust definition of the data structures representing individual satellites and orbital planes. Most orbital parameters are present in the `OrbitalPlane` struct to reduce repetition. Since we are analyzing circular orbits, orbital and angular speeds are calculated once from the other parameters. In our initial implementation, the position of a satellite was calculated on demand using the current simulation time. However, precalculating satellite positions at each time step and caching this information led to significant performance improvements.

```rust
pub struct Satellite {
    id: usize,
    orbital_plane: Arc<OrbitalPlane>,
    arg_periapsis: f64,
    position: Vector3<f64>,
    status: bool,
}

pub struct OrbitalPlane {
    id: usize,
    semimajor_axis: f64,
    inclination: f64,
    longitude: f64,
    // Calculated fields
    orbital_speed: f64,
    angular_speed: f64,
}
```

Listing 1: Definition of the `Satellite` and `OrbitalPlane` data structures

The Model data structure encapsulates the parameters of a satellite constellation, while other essential parameters for controlling the simulation, such as the time step, simulation speed, and satellite failure probability, are present in the Simulation struct (see Listing 2). This data structure also contains information regarding the connection strategy and graph topology.

```rust
pub struct Model {
    orbital_planes: Vec<Arc<OrbitalPlane>>,
    satellites: Vec<Satellite>,
    t: f64,
    max_connections: usize,
}

pub struct Simulation {
    model: Model,
    time_step: f64,
    simulation_speed: f64,
    connection_refresh_interval: f64,
    recurrent_failure_probability: f64,
    topology: ConnectionGraph,
    strategy: Box<dyn ConnectionStrategy>,
    // ... (some fields omitted)
}
```

Listing 2: Definition of the `Model` and `Simulation` data structures

The state of the simulation is updated at regular intervals, determined by the `update_frequency` parameter. The `simulation_speed` parameter indicates how quickly the simulation runs compared to real-time. For example, when set to a value of 5, each simulation second represents five actual seconds. Each update increments the time t by a fixed time step (calculated from the simulation speed and update frequency) and recalculates the position of each satellite. At intervals determined by the `connection_refresh_interval` parameter, the connection graph is refreshed, and potential satellite failures are simulated according to the specified failure probabilities. Listing 3 contains the code for the `step` method, which advances the simulation by a single time step.

```rust
pub fn step(&mut self) {
    self.model.increment_t(self.time_step);
    if self.t() >= self.last_update_timestamp +
    ↪ self.connection_refresh_interval {
        // Simulate potential satellite failures
        if self.recurrent_failure_probability > 0.0 {
            for sat in self.model.satellites_mut() {
                if sat.status() && self.rng.gen::<f64>() <
    ↪ self.recurrent_failure_probability {
                    sat.set_status(false);
                }
            }
        }

        self.update_connections();
    }
}
```

Listing 3: Method to advance the simulation by a single time step

To calculate the position of a satellite at a given instant, we first determine its angle relative to the start of its orbit by multiplying its angular speed by the current time. Using this value along with the orbital plane's inclination and the argument of periapsis, we then perform a series of rotations on the vector (a, 0, 0) (where a is the semimajor axis and the origin of our coordinate system corresponds to the center of the Earth).

2) Satellite Connection Strategies: The connection strategy chosen for a given scenario is responsible for establishing and updating the links between satellites over the course of the simulation. To facilitate the inclusion of new approaches for specifying the network topology, this functionality was implemented through the `ConnectionStrategy` trait. Any struct that implements this trait can be used to obtain a connection graph (Rust traits function similarly to interfaces from other languages). Our simulator provides two strategies: grid and nearest-neighbor.

When employing a grid strategy, each satellite will establish links with the two adjacent satellites within its orbital plane and corresponding satellites from the two neighboring orbital planes, resulting in a cross-like configuration. Linking satellites this way usually leads to a reasonably robust mesh network topology. We can also specify an offset value corresponding to the number of satellites to skip when connecting to neighboring planes (for example, an offset of 1 will cause satellites to connect to the satellite after the one directly to their right). These offsets can sometimes create more efficient communication paths, leading to improved performance.

The nearest-neighbor strategy links each satellite to the four closest active satellites at each update step. Despite creating short paths between satellites, this method can occasionally lead to erratic changes in the topology and loss of connectivity.

To confirm that the generated topologies are valid, we also conduct line-of-sight tests between linked satellites, ensuring that no communication paths are blocked by the Earth.

3) Configuration Files: The simulation parameters are obtained from TOML configuration files to facilitate the definition of new scenarios without modifying or extending existing code. The configuration files can include three main sections: constellation, simulation, and strategy parameters.

Listing 4 shows an example configuration file for the *Starlink* constellation. In addition to the parameters discussed previously, we can also specify the probability that a satellite will be inactive at the start of the simulation (`starting_failure_probability`) and the likelihood that a satellite will fail at each connection update (`recurrent_failure_probability`). We can also set a fixed seed for the random number generator to obtain consistently reproducible scenarios.

```toml
name = "Starlink - Proposal 2"

[constellation]
altitude = 0.55e6
num_orbital_planes = 24
satellites_per_plane = 66
inclination = 53.0
max_connections = 4

[simulation]
simulation_speed = 5.0
connection_refresh_interval = 20.0
rng_seed = 175394
starting_failure_probability = 0.01

[strategy]
type = "grid"
offset = 1
```

Listing 4: Example TOML configuration file

E. Interactive Visualization Component

Another substantial component of our system consists of a three-dimensional interactive visualization of the satellite constellation. Developed using Godot, it is a separate application that communicates with the simulator backend through a local TCP socket bound to port 2000. Godot is mainly used as an open-source 2D and 3D game engine [5] but can also be applied to visual simulation.

A separate thread handles communication between the two parts of the system. When the TCP connection is established, it sends an `init` message containing the parameters required to set up the visualization (such as the number of satellites and orbital planes, the inclination, and the simulation speed). Afterward, at a fixed rate specified in the configuration file, an `update` message will be sent containing a snapshot of the current state of the simulation. The message will also include information about the satellite link topology if the links have been updated since the last message. Messages use JSON to avoid the need for custom parsing.

Figure 3 presents a screenshot showcasing the main elements of the visualization interface. One can pivot the camera around Earth by holding the right mouse button and zoom in and out using the scroll wheel. The top-left panel shows general information about the simulation, while framerate and settings are displayed at the top right of the screen. Note that the altitude of the satellites is to scale, but the satellites themselves are not, as this would make it extremely difficult to see or select them. Because of this, satellites can appear to collide while being, in reality, kilometers away from each other.

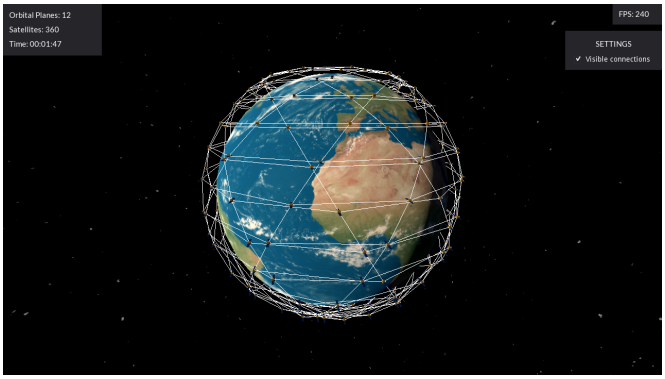


Fig. 3: Screenshot of the visualization component

Links between satellites are shown as thin white lines. For constellations with thousands of satellites, displaying these links can lead to drops in performance on less powerful hardware, so there is an option to disable them in the settings menu.

Satellites can be selected using the left mouse button, which provides additional options and information, as seen in Figure 4. A panel to the left shows information about the chosen satellite's id, position, status, and the identifiers of the satellites it is linked to. The links to other satellites will also turn red in the 3D view. There is also an option to induce a satellite failure manually. Satellite failures are displayed as a burning effect to be easily visible. The satellite's view cone is shown in green, and the orbital plane it belongs to is shown in light orange.

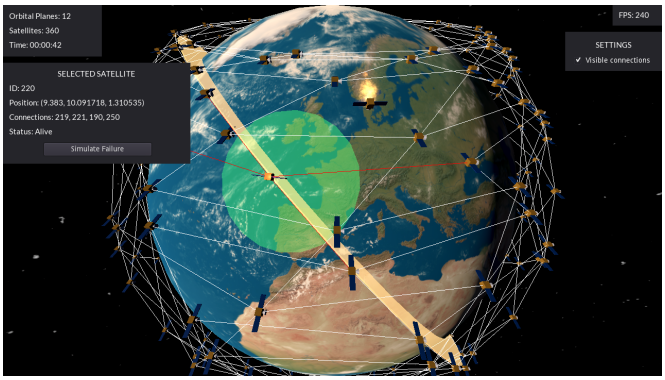


Fig. 4: Selecting a satellite

F. Real-Time Statistics Component

To assess the effectiveness of our cooperation strategies, we defined several performance metrics. These metrics are divided into two categories according to the characteristics they measure.

The first category relates to the connectivity of the underlying network graph. As previously mentioned, a constellation can be represented as an undirected graph where the vertices correspond to satellites and edges correspond to the links between them, weighted according to the distance between the pair of connected satellites. Thus, the communications distance between any pair of satellites corresponds to the length of the shortest path. These connectivity measures include:

- **Number of connected components**, corresponding to the number of connected sub-graphs. A graph is connected if there is a path from any of its nodes to any other node;
- **Number of articulation points**. Articulation points, or cut vertices, are nodes that, if removed from the graph, increase the number of connected components. Semantically, these can be seen as weaknesses in the network structure.
- **Satellite failure ratio**, corresponding to the ratio between the number of failed satellites and the total number of satellites in the constellation.
- **Graph density**, the density of the network graph, which represents the proportion between existing edges and the maximum possible edges on the graph. It is given by the formula $\frac{2E}{V(V-1)}$, where E is the number of edges, and V is the number of nodes in the graph.

The second category of metrics was used to measure communication performance. As in [6], we measured latency across continents, specifically between London and three other cities: New York City in North America, Singapore in Asia, and Johannesburg in Africa. The chosen performance measures are:

- **Round trip time (RTT)**, is the time required for a message to travel from the sender to the receiver, plus the time necessary for an acknowledgment to return to the sender. This measure doesn't consider the communication's efficiency, as communicating between more distant locations is generally expected to take longer.
- **Latency to distance ratio**, corresponding to the ratio between the RTT and the haversine distance between the sender and receiver. Lower values are associated with more efficient communication routes.

These metrics are calculated by the Rust backend, with the statistics visualization being in Python. Similarly to the 3D visualization tool, these components communicate through a TCP connection using port 2001, managed by a separate thread. Statistics messages also use the JSON interchange format.

We used a version of Tarjan's algorithm to calculate the number of articulation points. As for calculating the round-trip times, a naive solution would involve routing messages from the start location to the closest available satellite and then locating a route to the satellite nearest to the destination. Creating a copy of the topology and including additional edges between satellites visible from the start and end locations (that is, considering ground communication in addition to inter-satellite communication when calculating the shortest paths) leads to more efficient and consistent communication routes.

The real-time statistics visualization component was developed using Matplotlib. Matplotlib is a widely used data visualization Python library that can generate both static and animated plots [7]. We used the `FuncAnimation` class from Matplotlib's `animation` module to perform real-time plotting,

with the visual representation being updated as new data arrives from the TCP socket.

Additionally, there is an option to view statistics from a previously conducted scenario by specifying the path to a file containing the simulation data. This mode displays individual plots sequentially in a static manner.

III. Results and Discussion

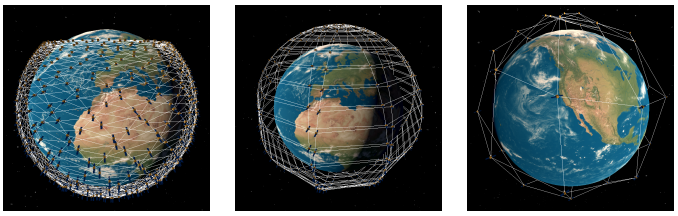
To adequately evaluate the developed system and obtain observations relevant to real-world scenarios, we utilized data from several existing LEO satellite constellations, specifically *Starlink*, *OneWeb*, and *Iridium*.

Starlink is owned by SpaceX and provides Internet services to multiple countries [8], [9]. In our simulator, we tested a group of 1584 *Starlink* satellites orbiting at 550km, at an inclination of 53º, and split across 24 orbital planes. Figure 5a shows a screenshot of this megaconstellation as seen using our visualization tool.

The OneWeb constellation (see Figure 5b) also offers Internet access and comprises 648 satellites at approximately 1200km. Satellites are distributed among 12 planes at an inclination of 86.4º (thus classified as near-polar orbits) [10]. At the time of writing, the constellation is not yet fully deployed.

The Iridium constellation (see Figure 5c) provides communications services, with 66 satellites orbiting at 781km. The six orbital planes, with 86.4º of inclination, are distributed in a Walker Star configuration, meaning that their longitude values span 180º around the Earth rather than the usual 360º for Walker Delta constellations [11].

This section defines a set of testing scenarios and presents statistical results collected from executing them on our simulator. The initial case is intended to be a purely descriptive analysis of the three constellations mentioned previously. The remaining scenarios examine the effects of adjusting different strategy parameters on latency and connectivity or analyze how satellite failures affect overall network performance.



(a) Starlink (b) OneWeb (c) Iridium

Fig. 5: Constellations as seen with the visualization component

A. Comparison of Real-World Constellations

The first scenario tested uses the default specifications of all three constellations, with the intent to compare how they perform under standard conditions. Figures 6, 7, and 8 present the evolution of the round-trip times and latency ratios over the course of the simulation for each constellation. All scenarios correspond to the simulation of two hours of orbits. The x-axis in each plot corresponds to the time in seconds since the start of the simulation. Round trip times are specified in milliseconds, while latency-to-distance ratios use seconds per meter.

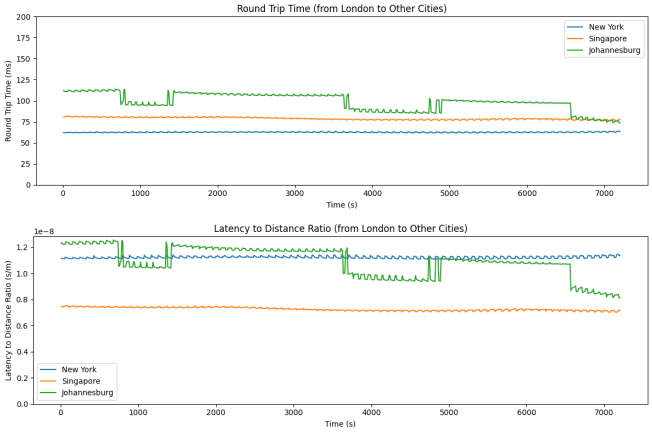


Fig. 6: *Starlink* Constellation

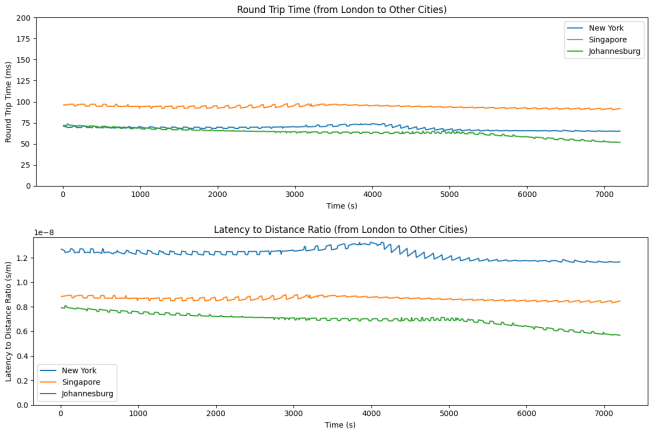


Fig. 7: *OneWeb* Constellation

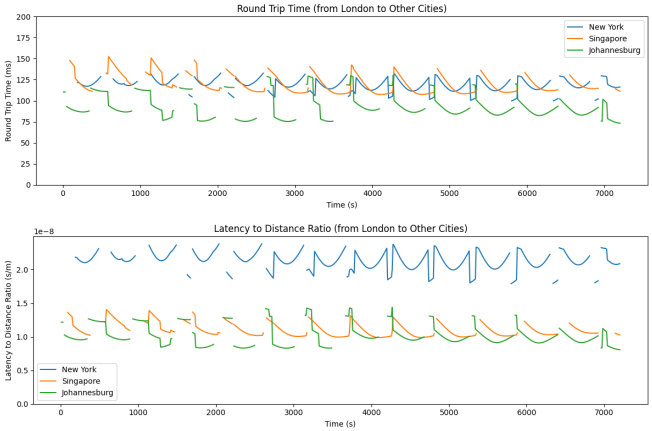


Fig. 8: *Iridium* Constellation

For the *Starlink* constellation (Figure 6), we observe remarkably consistent and short round-trip delays to New York

and Singapore (averaging 62.5 and 78.8 ms, respectively). Communications between London and Johannesburg have a bit more variance, with the average sitting at 98.8ms. These results are to be expected, given that the resulting topology favors east-west paths to the detriment of north-south routes. Also of note is the exceptional efficiency of London-Singapore paths, as observed from the latency ratio plot.

On the other hand, the near-polar orbits of the *OneWeb* constellation (Figure 7) lead to excellent Johannesburg RTTs (63.2ms) at the cost of increased New York (68.5ms) and Singapore (98.8ms) delays.

The *Iridium* constellation displayed significant variances in RTTs. We can also observe continuity breaks in the charts, which correspond to intervals with no available communication path between the two cities. We suspect this might be due to insufficient information regarding the configuration of the satellites or the fact that the grid strategy could require a different setup when applied to Walker Star constellations.

B. Grid and Nearest Neighbor Strategies

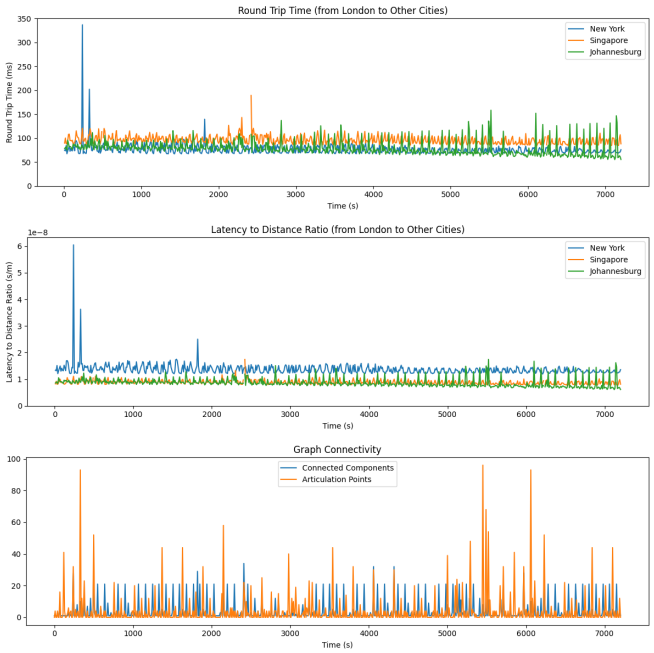


Fig. 9: *Starlink* – Nearest Neighbor Strategy

Figures 6 and 9 illustrate the impact of the different connection strategies (grid and nearest neighbor, respectively) on the *Starlink* network's performance.

As the nearest-neighbor strategy doesn't consider the overall topology and solely looks to maximize local communication speed, latency values are highly erratic, and spikes can be observed across all three routes. We can also notice losses in connectivity throughout the network as well as an abundance of articulation points. Despite this, RTTs for Johannesburg improved to an average of 77.8ms compared to the grid strategy, while the communication speed between London and New York or Singapore deteriorated. One can argue, however, that even the shortened Johannesburg RTTs do not justify the overall decline in connection stability.

C. Grid Offset Parameter

The third experiment evaluated the effects of varying the offset parameter associated with the grid strategy. The base *Starlink* configuration uses an offset of 1 (see Figure 6), meaning that satellites will connect to the next satellite in the neighboring orbital plane rather than the satellite directly to their right.

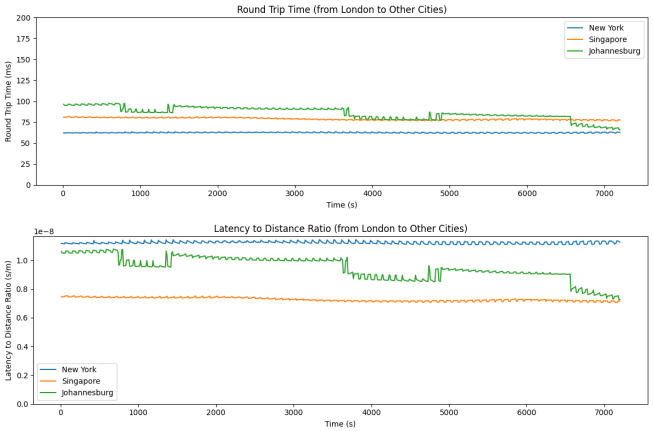


Fig. 10: *Starlink* – Grid Strategy – No Offset

In theory, we expected this to generate a more effective mesh network. Still, we observed that not including this offset improved the average RTT for communications with Johannesburg to 85.8ms, with little impact on the delays associated with the other routes (see Figure 10). Nevertheless, taking more definitive conclusions from these tests would require precise knowledge of the phasing coefficient between the orbital planes and the exact type of grid used.

D. Satellite Failure Scenarios

In this scenario, we evaluate the impact of satellite failures on the Starlink constellation, both when using the grid and nearest-neighbor strategies. We employed a recurring satellite failure probability of 0.03%, corresponding to the likelihood of any individual satellite failing at each connection update. We used the same seed for the random number generator to assess both strategies with maximum fairness.

Figure 11 shows the evolution of the percentage of failed satellites throughout the simulation. As expected, given the failure simulation method, satellite failures grew linearly with time. Note that these experiments represent an extreme scenario to aid in the analysis. In reality, we expect the number of satellite failures to be far less pronounced.

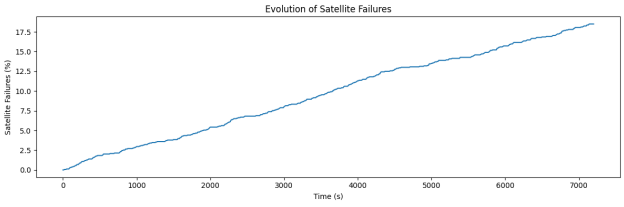


Fig. 11: Percentage of Failed Satellites over Time

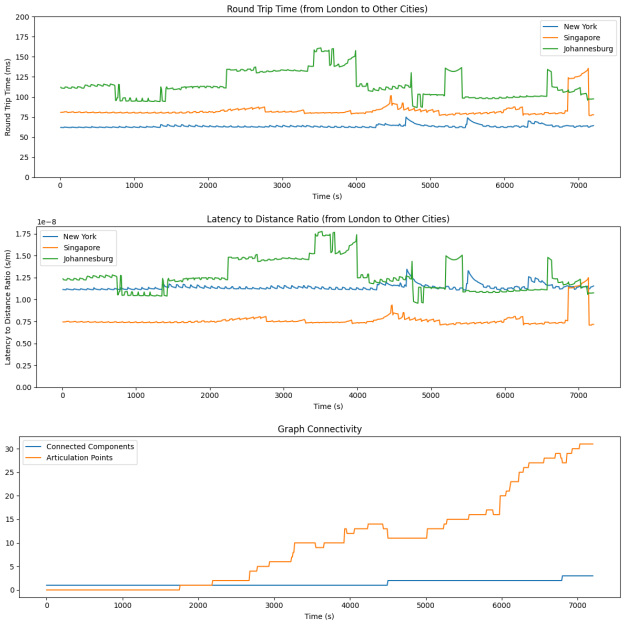


Fig. 12: *Starlink* – Grid Strategy – Satellite Failures

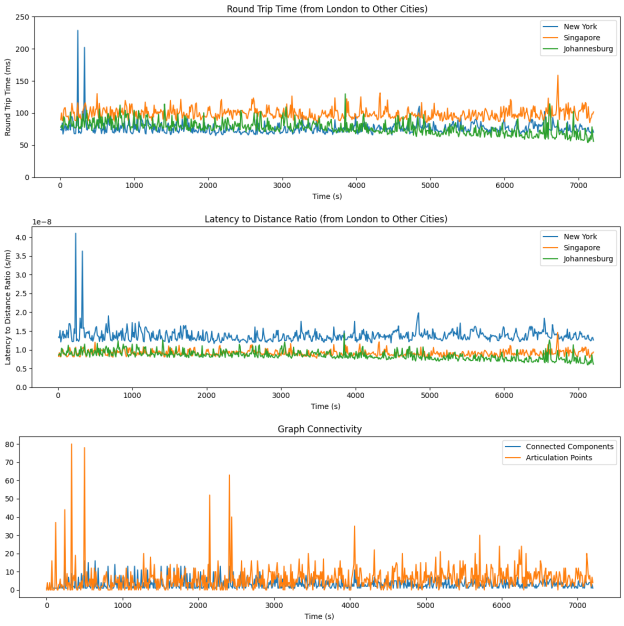


Fig. 13: *Starlink* – Nearest Neighbor Strategy – Satellite Failures

The grid strategy proved particularly robust, maintaining connectivity until around 12.8% of the satellites had failed (see Figure 12). Round-trip times for New York and Singapore remained stable throughout most of the simulation. We observed more significant variances with the Johannesburg route, suggesting that it is likely more susceptible to failures in the network.

As for the nearest-neighbor strategy, we didn't detect much of a noticeable difference between its normal behavior (see Figure 9) and its performance under failures (see Figure 13). It maintains the same erratic and inconsistent changes in latency, and the same connectivity issues arise, but average RTTs increased only slightly as the percentage of failed satellites grew.

IV. Related Work

There are several satellite megaconstellation simulators in the literature. The most well-known example of such simulations being used to assess the performance of these constellations is Handley's analysis of early iterations of SpaceX's Starlink constellation[6]. There are other notable examples of simulation-driven analysis in the literature, such as Kempton's[12] and Z. Lai et al's[13] contributions to this subject matter.

V. Conclusions and Future Work

This paper details the requirements and architecture of a system for modeling and simulating satellite megaconstellations. To complement the simulation backend, we developed real-time interactive visualization and statistics components. In addition to the simulation of the satellites' movement in orbit, our simulator can establish inter-satellite links in a centralized manner, using two different strategies, and simulate complete satellite failures. When comparing real-world constellations, we concluded that the Starlink constellation is well-suited for east-west communications paths, whereas OneWeb excels at north-south routes. Among the implemented connection strategies, the nearest neighbor proved more unstable. Besides being highly dependent on the constellation structure, it can sometimes lead to the loss of connectivity. On the other hand, despite its basic concept, the grid connection strategy proved resilient and capable of keeping the network connected even with a fairly high percentage of satellites down.

Although our base model and visualization components are sufficiently robust, we can still pinpoint ways of improving and extending them. In addition to further researching and improving the accuracy of the configuration data for simulating real-world constellations, some possible extensions include simulating communication delays, constellations containing satellites at different altitudes, and different kinds of satellite failures (such as communication problems with other satellites or ground stations instead of complete shutdown). Furthermore, exploring a decentralized multi-agent simulation angle to establish satellite connections could prove interesting. Beyond that, we could feed the topologies to a network simulator to extract more insights.

References

[1] F. Kinsella, *Mega-constellations in space: Revolutionising the satellite industry*, https://securecommunications.airbus.com/en/meet-the-experts/mega-constellations-in-space-revolutionising-satellite-industry.

[2] P. Williamson, *Keplerian Elements Tutorial [Archived]*, https://web.archive.org/web/20021014232553/http://www.amsat.org/amsat/keps/kepmodel.html, Jun. 2001.

[3] Lasunncty, *Diagram illustrating and explaining various terms in relation to orbits of celestial bodies*, https://commons.wikimedia.org/wiki/File:Orbit1.svg, Oct. 2007.

[4] G. J. Walker, "Satellite constellations," *Journal of the British Interplanetary Society*, vol. 37, pp. 559–571, 1984.

[5] J. Linietsky, A. Manzur, and Godot Contributors, *Godot engine - free and open source 2d and 3d game engine*, https://godotengine.org/, Jan. 2023.

[6] M. Handley, *Starlink revisions, Nov 2018*, https://www.youtube.com/watch?v=QEIUdMiColU, Nov. 2018.

[7] Matplotlib Development Team, *Matplotlib — Visualization with Python*, https://matplotlib.org/.

[8] Starlink, *How Starlink Works*, https://www.starlink.com/technology, 2023.

[9] J. C. McDowell, *Starlink Statistics*, https://planet4589.org/space/con/star/stats.html, Jan. 2023.

[10] D. Mohney, *OneWeb talks satellite broadband speeds, constellation configs*, https://www.spaceitbridge.com/oneweb-talks-satellite-broadband-speeds-constellation-configs.htm, Oct. 2019.

[11] W. Graham, *Iridium next-5 satellites ride to orbit on spacex falcon 9*, https://www.nasaspaceflight.com/2018/03/iridium-next-5-satellites-spacex-falcon-9/, Mar. 2018.

[12] B. S. Kempton, "A simulation tool to study routing in large broadband satellite networks," 2020, ISBN: 9798662568945. [Online]. Available: https://search.proquest.com/docview/2438605919?accountid=27868.

[13] Z. Lai, H. Li, and J. Li, "Starperf: Characterizing network performance for emerging mega-constellations," in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, 2020, pp. 1–11. DOI: 10.1109/ICNP49622.2020.9259357.