FLoRaSat 2: Simulating Cross-Linked Direct-to-Satellite IoT LEO Constellations

Alexander Y. Choquenaira-Florez*, Robin Ohs†, Juan A. Fraire*‡, Hervé Rivano*
*Inria, INSA Lyon, CITI, UR3720, 69621 Villeurbanne, France
†Saarland University, Saarland Informatics Campus, 66123 Saarbrücken, Germany
‡CONICET - Universidad Nacional de Córdoba, Córdoba, Argentina

***Abstract*—Direct-to-Satellite IoT (DTS-IOT) represents a promising solution for data transmission in remote regions where terrestrial infrastructure deployment is unfeasible. In DTS-IOT scenarios, Low-Earth Orbit (LEO) satellites function as in-orbit gateways. Addressing the need for practical simulation tools, we present FLORASAT 2, an open-source, event-driven, end-to-end simulation tool leveraging OMNET++. The original FLORASAT met many DTS-IOT Medium Access Control (MAC) requirements. Still, this enhanced version introduces advanced Inter-Satellite Link (ISL) communication modules, including a helper for constellation creation, dynamic ISL topology control, routing, and analytics, facilitating the thorough evaluation of constellation-grade DTS-IOT networks. These new features allow detailed simulation scenario configuration, flexible support for developing and including diverse routing algorithms, and the tooling to perform automated data analysis from parametric simulations. Overall, the simulator enables the analysis of complex behaviors in DTS-IOT environments, optimizing performance and enhancing connectivity and efficiency in large-scale satellite IoT constellation networks.**

***Index Terms*—Satellite Communication, LEO satellite, LORAWAN, LORA, IoT, Simulation, OMNET++**

I. INTRODUCTION

The growing number of space projects [1], including innovative, cost-effective launch solutions and affordable nanosatellites such as *CubeSats*, has revamped the popularity of satellite communications. This opens up a possibility for telecommunication operators to expand IoT coverage and offer potential constant worldwide services [2]. Additionally, it also impacts the Internet of Things (IoT) community, which is continually exploring diversification in its applications [3].

Low-Earth Orbit (LEO) satellites, such as *CubeSats*, are at the heart of the emerging space age since their unique characteristics can play a critical role in the new IoT era. These features include short orbital periods of 90 minutes at an altitude of 500 km and cost-effective and rapid deployment capabilities. Moreover, cost-effective satellites can act as mobile IoT gateways, providing connectivity to a geographic area for 6–12 minutes, up to 2–4 times daily. The IoT devices can directly communicate with the gateway onboard LEO satellites using technologies with low-power and long-range, such as NB-IOT and LORA. These advantages can unlock applications in less accessible regions (e.g., oceans, deserts, poles, and other remote areas) where IoT gateways may not be economically justified [4].

This type of application is known as Direct-to-Satellite IoT (DTS-IOT), which, besides its benefits, addresses critical and challenging problems [5] such as transmission distances and channel dynamics, among others. The core network of the DTS-IOT system extends into the space domain. It must operate over low-bandwidth inter-satellite links (ISL), which also suffer from periodic changes due to the orbital dynamics.

In response to these difficulties, FLORASAT (Framework for LoRa-based Satellite IoT) was initially introduced in [6] as an open-source, event-driven simulator. It replicates the behavior of LoRa/LoRaWAN-based IoT systems and their interactions with satellite networks. The authors demonstrated the efficiency and benefits of FLORASAT on studying and analyzing the performance of LORAWAN DTS-IOT networks. The FLORASAT simulator leverages the well-known OMNET++ framework with its INET extension for Internet stack and a LORAWAN protocol implementation adapted from the open-source simulator FLORA [7]. FLORASAT is a unique, comprehensive, and efficient solution for satellite network simulations. However, FLORASAT was strictly focused on the device-to-satellite link, ignoring the effect of the packet forwarding across the cross-linked LEO constellation.

To ensure the continued growth of the simulator and address the challenges of designing and simulating cross-linked satellite constellations, this work presents new features that have been integrated into the FLORASAT simulator, significantly enhancing its capabilities in this domain. The scope of these improvements is relevant for studying next-generation constellations, such as Starlink [8] and Iridium [9], which utilize ISLs to achieve global coverage and low-latency communication. Moreover, these advancements enable a detailed analysis, seamless integration, and graphic comprehensive comparison of routing algorithms' performance within constellations. The contributions of this paper are:

- **Development of FLoRaSat 2** (Section III): We introduce an enhanced open-source simulation tool featuring advanced modules for creating and analyzing cross-linked DTS-IOT constellations, including constellation design, topology control, routing, and event management.
- **Introduction of the Routing Sandbox** (Section IV): We present a new flexible platform supporting the implementation and evaluation of diverse routing algorithms tailored for large-scale satellite DTS-IOT networks, enabling seamless integration and testing in FLORASAT 2.

- **Evaluation of Starlink and Iridium** (Section V): We use FLoRaSat 2 analytics and visualization capabilities to evaluate two use cases. These cases assess routing algorithms and validate FLoRaSat 2's capabilities in simulating and benchmarking Starlink and Iridium topologies.

The remainder of this paper is structured as follows. Section II presents an overview of FLoRaSat's unique characteristics. Section III introduces the novel developments and advancements in FLoRaSat 2, including its extended architecture supporting constellations, explaining the new modules. Section IV presents the novel routing sandbox in FLoRaSat 2. Section V analyzes two use case scenarios and analyzes their results. Finally, Section VI summarizes the conclusions and future research directions.

II. FLoRaSat Overview

One of the main challenges in the DtS-IoT domain is its evaluation since real in-orbit deployments are expensive and mostly inaccessible. As a result, the need for a realistic simulator that implements IoT and satellite dynamics is evident. To this end, FLoRaSat (Framework for LoRa-based Satellite IoT) was initially introduced in [6] as an open-source, event-driven simulator that replicates the behavior of LoRa/LoRaWAN-based IoT systems and their interactions. It was designed to address the requirements for realistic simulation tools in DtS-IoT application environments. These include (i) an accelerated and effective simulation to analyze LEO orbits over long periods and (ii) a realistic implementation of the physical and medium access layers, accompanied by detailed modeling of the end-to-end network layer.

Related Tooling: In this context, FLoRaSat is a unique tool. *STK* [10], a widely used licensed software, offers robust analytical capabilities but precise complementary development to implement routing functionality. Another appealing alternative, *Hypatia* [11], is an open-source tool software based on NS-3 [12], that provides a framework for easy and quick implementation of constellations comparable to FLoRaSat. However, it lacks support on DtS-IoT scenarios. On the other hand, *OpenSAND* is an open-source software tool designed to provide an easy and flexible way to emulate an end-to-end satellite communication system [13]. However, it does not provide a comprehensive graphical interface that allows visual inspection of the entire constellation or packet tracking in flight. Another potential solution is the General Mission Analysis Tool *(GMAT)* [14], an open-source platform independently developed by NASA in collaboration with the industry. *GMAT* only supports design and trajectory satellite optimization. Similarly, *Orekit*, written in Java, is an accurate and efficient tool for the design of space flight dynamics application [15]. While Orekit is an excellent choice for space-related software development, its API is comparably low-level. Moreover, it does not provide ISL contact computation, benchmark, or constellation out-of-the-box evaluation capabilities as FLoRaSat. Finally, *SILLEO-SCNS* appears as a noteworthy open-source option capable of simulating and analyzing satellite constellations, including ISL connectivity. However, its analytical capabilities are limited. Table I summarizes the features and capabilities of all the identified related tooling.

A. Core Features in Original Version

FLoRaSat leverages the well-established OMNeT++ framework with the INET extension, which provides internet stack services and incorporates the open-source FLoRa implementation. The authors also customized the physical layer (LoRa) and the link layer (LoRaWAN) to the DtS-IoT configuration. Next, we provide an overview of the core features of the original version.

- **Orbital Mechanics:** The system uses Contact Plan Designer [17] to create an orbital processing pipeline and respond to the changes in the orbital dynamics.
- **Beacon-Based Radio:** Beacons can play a critical role in announcing satellite presence [18]. FLoRaSat follows the LoRaWAN standards by implementing Class B while retaining legacy Class A support for compatibility.
- **Channel Model:** Inspired by [19], FLoRaSat employs a simple channel model for attenuation, interference, and the capture effect at the receiver. These factors significantly affect uplink and downlink performance.

The early version of FLoRaSat [6] was focused on the LoRa/LoRaWAN MAC layer between the satellite and the devices on the ground. Only a simplistic "right-up and down-left" hard-coded routing method was implemented for managing uplink and downlink traffic. This was insufficient to address the analysis demands of emerging multi-satellite DtS-IoT constellations with data exchange in orbit. The remainder of this paper presents FLoRaSat 2, which addresses the crucial aspect of constellation analysis in DtS-IoT.

III. FLoRaSat 2: Constellation Modules

This section introduces new and additional features developed to enhance its capabilities and address critical challenges in designing simulation scenarios based on cross-linked satellite constellations. We have developed several new modules to enhance the simulator's capabilities and address challenges in modeling more sophisticated scenarios.

These additions expand FLoRaSat's functionality by incorporating modules that simulate cross-link capabilities. Each new module enables more accurate environmental representation, especially modeling inter-satellite communications, for a detailed analysis and interpretation. The code developed for this work is publicly available[1]. Next, we will describe the new major modules in detail, outlining their functionality, parameters, and outputs. The routing sandbox is analyzed separately in Section IV.

A. Constellation Creator

The *Constellation* module configures the necessary orbital parameters for each satellite. The module enables the definition of complex satellite constellations, such as Walker-Delta and

[1]GitLab repository: https://gitlab.inria.fr/jfraire/florasat/-/tree/FLoRaSat_v2

TABLE I
COMPARISON OF TOOLS BASED ON FUNCTIONALITIES

Simulator	Orbital Propagator	Packet Simulation	DTS-IOT	Open Source	Constellation-Grade
STK [10]	✓	×	×	×	✓
Hypatia [11]	✓	✓	×	✓	✓
OpenSAND [13]	✓	×	×	✓	✓
GMAT [14]	✓	×	×	✓	×
Orekit [15]	✓	×	×	✓	×
SILLEO-SCNS [16]	✓	×	×	✓	✓
FLORASAT [6]	✓	✓	✓	✓	×
FLORASAT 2 (this paper)	✓	✓	✓	✓	✓

Walker-Star, directly from high-level Walker notation, reducing the risk of errors associated with manual configuration. It also allows users to implement complex constellations, such as OneWeb [20], [21] or Starlink systems.

The satellite positions generated by this module were validated against reference values extracted from *STK* [10], a widely used tool in research and industry. Comparison showed minimal differences in latitude and longitude values, demonstrating the reliability of the module's results. The parameters required for the constellation module are detailed in Table II and obtained through a configuration file at the beginning of the simulation, where we can highlight the interplane spacing, which defines the relative offset in the argument of latitude between neighboring satellites in adjacent orbital planes [22].

B. Topology Control

On the other side, the *Topology* module is responsible for managing communication channels between satellite modules and ground modules. Additionally, this module is responsible for periodically updating the inter-plane ISLs and GSL to account for changing distances caused by movements. The parameters considered for the topology control can be detailed in Table II. We can highlight the minimum elevation, which is the minimum angle required to establish a connection between ground stations and satellites [23]; data rate, the amount of data transmitted per unit time; Walker type that indicates the type of Walker constellation, such as *Star* or *Delta*.

C. Event Manager

Due to the harsh environment and the lack of repair capabilities, we have implemented an *Event Manager* module that temporarily or permanently disables inter-satellite links. The idea is to test routing algorithms for their adaptability to failures, which complements FLORASAT's ability to test congestion handling mechanisms. This is done according to a scenario file loaded by the simulator during startup.

D. Visualization and Metrics

We developed FLORASAT-CLI (Command Line Interface), a functionality to analyze simulation results through metrics and visualizations efficiently. FLORASAT uses C++ as its primary programming language, which is unsuitable for performing statistical and graphical analyses. Alternatively, Python offers a wide range of specialized packages for data analysis, e.g., *Pandas, Matplotlib, and Scipy, among others.* This module serializes the collected results into well-defined CSV files for Python use.

However, the pre-processing step requires managing large amounts of data to calculate output metrics. We use a *Rust* implementation to handle this task efficiently, which reduces runtime and avoids memory limit problems. The *Rust* code is built as a cdylib library, which allows a direct import in Python code. Finally, FLORASAT-CLI supports the creation of comparative graphs and charts for metrics detailed previously. All the plots in the use case analyses presented in Section V are obtained from FLORASAT-CLI.

Due to its flexible architecture, it is essential to highlight that additional graphs, metrics, or output formats can be easily incorporated into the module.

E. Extended Architecture

This subsection overviews the redesigned architecture, emphasizing the integration of new modules to enhance modularity, scalability, and efficiency in complex scenarios.

The initial architecture adopted a modular design, with an OMNET++ module reflecting each component. As new modules were introduced, the architecture was extended, necessitating a thorough explanation of the behavior and interaction between all modules. Figure 1 presents the new architecture, where each block represents a module of FLORASAT with its main components. Next, we will explore each component's features and interactions, providing insights into their roles.

As core components, the *Device* and *Satellite* modules are primarily responsible for implementing LORA and LORAWAN operations within satellite and ground devices. To ensure seamless communication, these tasks include multiplexing and retransmitting LORAWAN frames between satellites and ground devices. These modules also manage mobility operations, which are crucial in simulation scenarios. On the other hand, the constellation module considers the parameters detailed in Table II to create each element of the constellation accurately. Closely related, the routing sandbox (described in the following section) utilizes the specified routing algorithm to establish connections across the network. Finally, given that the statistics module requires a high computational cost, it is not directly integrated into FLORASAT because it can significantly influence the simulation speed (simulated seconds per real second). Managing this last module separately allows

TABLE II
MODULE PARAMETERS AND METRICS

Constellation Creator	Topology Control	Statistics
Parameters	**Parameters**	**Metrics**
• Number of gateways • Index • Initial mean anomaly • RAAN • Altitude • Eccentricity • Orbit inclination • Plane Count • Interplane Spacing	• Minimum Elevation • Data Rate (ISL, Ground) • Delay (ISL, Ground) • Update Interval • Walker Type • ISL Status • Interplane Spacing • Plane Count	• Hopcount • Distance • Delivery Ratio • Packet Loss • Throughput • Drop heatmap • E2E Delay • Avg. queue size • Protocol Overhead • Failure comparison

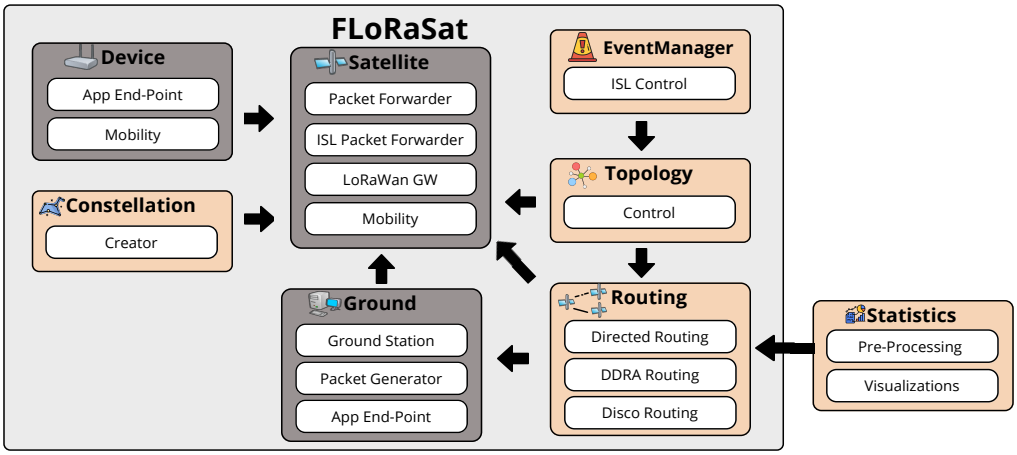

Fig. 1. The extended architecture of FLoRaSat. Orange modules were introduced in FLoRaSat 2.

the simulator to maintain an efficient and flexible simulation environment.

IV. FLoRaSat 2: Routing Sandbox

The *Routing* sandbox is based on an abstract class to facilitate the creation and seamless integration of new routing algorithms required by satellites and ground stations to enable support for a wide variety of routing algorithms. Additionally, the module facilitates the easy and fast integration of new routing algorithms, which expands its utility for researching and developing new routing strategies.

The abstract class is called *RoutingBase* and defines functions that each novel routing algorithm must implement. An example of this structure can be seen in Figure 2. We describe each function below and the main aspects to consider when implementing new routing algorithms.

- **handlePacket:** It is responsible for the further processing of the packet and routing instructions. If the packet arrives at the destination, it should execute the appropriate steps that depend on the routing algorithm. Otherwise, the function should determine the next hop calling *sendMessage* or drop the packet if necessary and call *dropPacket*. This function is the only mandatory for possible extended routing algorithms.
- **handleTopologyChange:** The topology control calls this function after any predictable topology change. It can be used to calculate new routes and store the next hop inside a forwarding table or to reset the state.
- **preparePacket:** This function is not required for all routing algorithms since it performs additional operations on the packet. An example of an operation could be calculating the full path to the destination satellite and storing it in the packet header.
- **wrapUpPacket:** Similar to the previous function, it is not always required. This function is called before a packet is sent to a GSL. An example of using this function can be removing routing protocol-specific headers before the packet is reinserted into the regular network.
- **handlePacketDrop:** It notifies the routing module when a packet is dropped. This function can be applied to congestion control or fault recovery scenarios.
- **handleQueueSize:** This function reports the current and maximum processing queue size to the routing module. For example, it can be used as a trigger when a specific

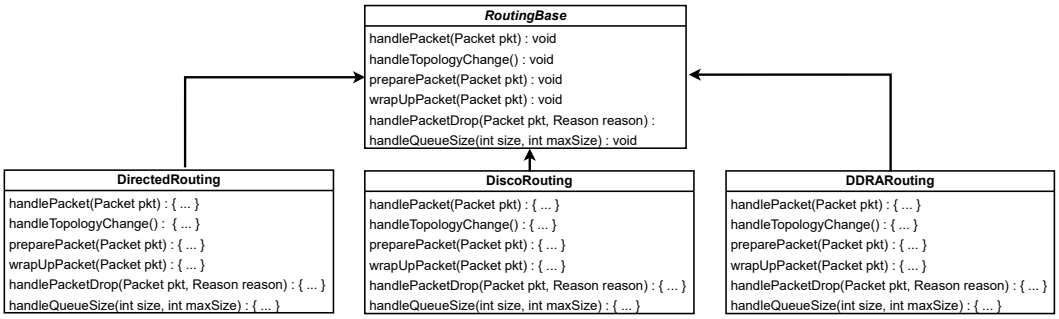

Fig. 2. Structure of the *RoutingBase* Abstract Class and implementations

queue length is reached to send congestion notifications to neighboring satellites.

Below, we list some routing algorithms implemented in the latest routing module in FLoRaSat 2: DIRECTED routing, DDRA, and DISCOROUTE.

A. Directed Routing

DIRECTED routing is a simple algorithm that employs the Manhattan distance as the heuristic to select the next hop, as suggested in [24]. The algorithm selects the next hop from an open list of potential nodes. Calculating the Manhattan distance requires specific considerations. In the case of a Walker-Star type constellation, assuming a network with v planes and w satellites per plane, the Manhattan distance between A and B can be determined using Equation 1.

$$\begin{aligned} d_1 &= |A_x - B_x| + |A_y - B_y| \\ d_2 &= 1 + A_y + |A_x - B_x| + |(w-1) - B_y| \\ d_3 &= w - A_y + |A_x - B_x| + |0 - B_y| \\ d &= \min(d_1, d_2, d_3) \end{aligned} \tag{1}$$

On the other hand, if the algorithm is applied to Walker-Delta constellations, the Equation 2 should be considered. It adds the new possible paths for horizontal cyclic edges from left to right, along with the vertical and horizontal cyclic hop.

$$\begin{aligned} d_1 &= |A_x - B_x| + |A_y - B_y| \\ d_2 &= 1 + A_y + |A_x - B_x| + |(w-1) - B_y| \\ d_3 &= w - A_y + |A_x - B_x| + |0 - B_y| \\ d_4 &= 1 + A_x + |(v-1) - B_x| + |A_y - B_y| \\ d_5 &= v - A_x + |0 - B_x| + |A_y - B_y| \\ d_6 &= v - A_x + w - A_y + |0 - B_x| + |0 - B_y| \\ d_7 &= v - A_x + 1 + A_y + |0 - B_x| + |(w-1) - B_y| \\ d_8 &= 1 + A_x + w - A_y + |(v-1) - B_x| + |0 - B_y| \\ d_9 &= 1 + A_x + 1 + A_y + |(v-1) - B_x| + |(w-1) - B_y| \\ d &= \min(d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9) \end{aligned} \tag{2}$$

B. DDRA

The Dynamic Detection Routing Algorithm (DDRA), initially proposed in [25], is a reactive algorithm that optimizes shortest path selection. The algorithm divides the satellite's period into time slices, where the constellation routing topology remains constant. Following this technique, there are two types of changes: predictable changes, which occur when a new topology is loaded in the next time slice, and sudden changes, generated by congestion or failures.

DDRA employs the *DSPA* techniques introduced by [26], using the precomputed shortest path to each satellite at each time slice as the potential optimal route. This does not guarantee that packets will take the shortest path but can significantly reduce processing power consumption. Additionally, the authors propose a congestion control mechanism where each satellite periodically monitors the number of packets in its queue. The satellite reports its congestion status to neighboring nodes if it exceeds a specified threshold. The threshold is calculated as follows:

$$Threshold \leq \frac{T_{\text{MAX}}}{M} \times \frac{V_{\text{MAX}}}{L_{\text{MIN}}} \tag{3}$$

where T_{MAX} is the maximum tolerable delay normally default to 500ms, M is the average number of hops in the constellation, V_{MAX} is the maximum link transmission data rate, and L_{MIN} refers to the minimum packet length.

C. DiscoRoute

This routing algorithm [27] is specially focused on mega-constellations comprising hundreds or thousands of satellites but only applies to Walker-Delta constellations. DISCOROUTE uses the orbital elements to calculate the minimum number of ISL hops between a satellite pair. Leveraging the deterministic nature of satellite orbital parameters, this algorithm aims to optimize inter-plane hops by prioritizing transitions near the poles, where satellite pairs are closer to the equator. According to the authors, the DISCOROUTE algorithm requires shorter processing times than DDRA and offers greater scalability.

V. USE CASES

In this section, we validate the new modules presented in this work. To this end, we propose two benchmark scenarios: creating a constellation of LEO satellites, applying different routing algorithms, and comparing their performance.

Twenty globally distributed ground stations transmit traffic patterns through this constellation. The resulting scenario is

```
Iridium.ini
*.topologyControl.minimumElevation = 5.0
*.topologyControl.islDatarate = 50000000
*.topologyControl.groundlinkDatarate = 50000000

*.nrOfGW = 66
*.topologyControl.walkerType = "STAR"
*.constellationCreator.raanSpread = 180
*.constellationCreator.inclination = 86.4
*.constellationCreator.altitude = 781
*.constellationCreator.baseYear = 20
*.constellationCreator.baseDay = ${1.5, 2.0 ! repetition}
*.constellationCreator.epochYear = 20
*.constellationCreator.epochDay = 1.5
*.**.interPlaneSpacing = 5
*.**.planeCount = 6

*.packetRecorder.constName = "Iridium"
```

Fig. 3. Configuration file Iridium Constellation

```
StarLink.ini
*.topologyControl.minimumElevation = 30.0
*.topologyControl.islDatarate = 50000000
*.topologyControl.groundlinkDatarate = 50000000

*.nrOfGW = 1584
*.topologyControl.walkerType = "DELTA"
*.constellationCreator.raanSpread = 360
*.constellationCreator.inclination = 60.0
*.constellationCreator.altitude = 550
*.constellationCreator.baseYear = 20
*.constellationCreator.baseDay = ${1.5, 2.0 ! repetition}
*.constellationCreator.epochYear = 20
*.constellationCreator.epochDay = 1.5
*.**.interPlaneSpacing = 39
*.**.planeCount = 72

*.packetRecorder.constName = "StarLink"
```

Fig. 4. Configuration file for Starlink constellation

configured with a total runtime of 15 simulated minutes and a data rate of 50 Mbps. The network supports a maximum of 25 hops, each with a processing overhead of 1 millisecond. Additionally, the satellites' queue capacity is set to 40.

Simulations were executed on an Ubuntu Linux with an Intel Xeon Gold 5122 CPU @ 3.60 GHz and 64 GB of RAM. However, the simulations do not require significant memory in most cases.

The first scenario is a Walker-Star Iridium constellation composed of 66 LEO satellites. The main parameters are configured in a config file, which is then loaded by the FLORASAT simulator. The values used in this benchmark scenario are shown in Figure 3.

The second scenario tests the ability of DIRECTED and DISCOROUTE algorithms to recover from link failures. It is based on a Walker-Delta Starlink constellation composed of 1584 satellites. The main parameters of the constellation can be seen in Figure 4. In this scenario, the failure probability is set to a high value (1.%), reflecting realistic conditions since these events are relatively rare.

A. *Comparison of Routing Algorithms* DDRA *vs.* DIRECTED

After setting the simulation parameters, running the simulations will generate CSV files containing the collected runtime metrics for both algorithms. We use the FLORASAT-CLI to analyze these metrics and generate corresponding graphs, demonstrating one major strength of FLORASAT. Figure 5 presents some graphics generated using the FLORASAT-CLI with the scenario simulation detailed. Next, we will analyze each of these graphics in detail.

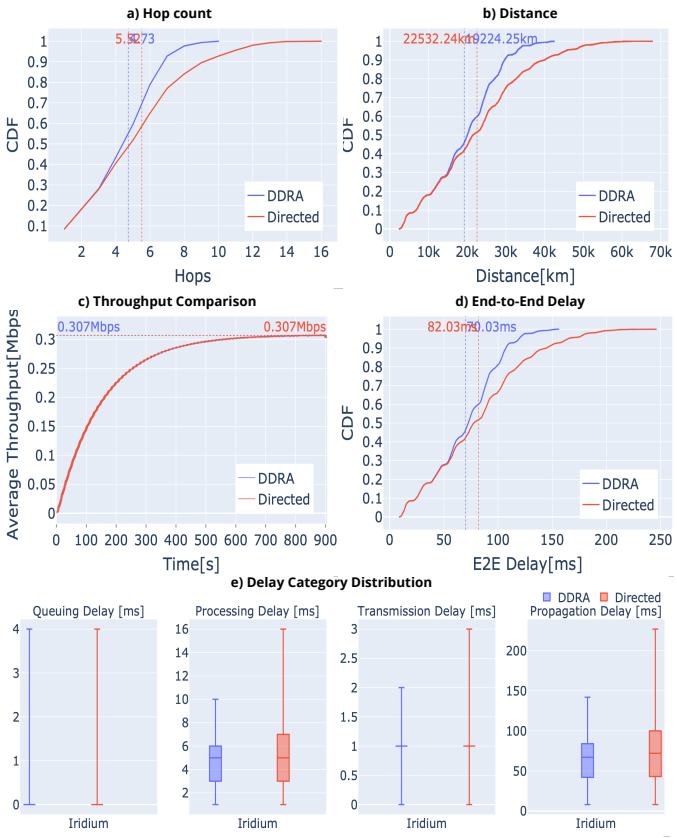

Fig. 5. Comparative graphs created by the FLORASAT-CLI for the first benchmark.

a) Hop Count and Distance: Although both algorithms exhibit a similar average number of hops, DDRA performs slightly better than DIRECTED in average hop count. We also noticed in the Figure 5-a), that the algorithms reach at most around 10 and over 16 hops for DDRA and DIRECTED respectively. This suggests that the DIRECTED algorithm may suffer packet loss due to expiration from excessively long routes. Furthermore, these results also explain why the average route length of the DIRECTED algorithm overpass DDRA Figure 5-b). Additionally, CDF grows faster in the DDRA algorithm, which indicates its capability to deliver a higher proportion of packets with fewer hops or distances.

b) Throughput Comparison: A graphical comparison of the average throughput between the two routing algorithms can be seen in Figure 5-c). DIRECTED performs equal to DDRA, indicating that both algorithms can sustain the same throughput given the defined scenario.

c) End-to-End Delay: Figure 5-d) shows the average End-to-End(E2E) delay CDF for delivered packets using DDRA and DIRECTED algorithms. It indicates that DDRA performs better than DIRECTED, which can be explained by the faster growth of CDF values for hop count and distance in Figure 5-a) and b). Additionally, Figure 5-e) presents a box plot analysis of the subcategory delays.

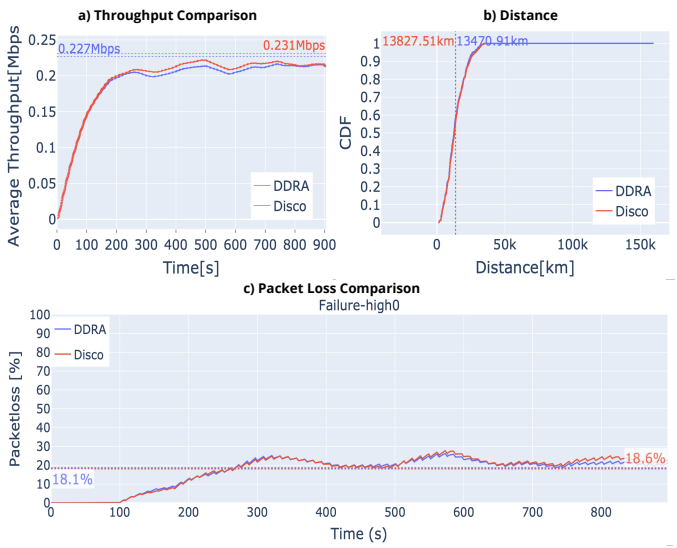

Fig. 6. Comparative graphs created by the FLORASAT-CLI for the second benchmark.

B. Comparison of Routing Algorithms DDRA *vs.* DISCOROUTE

Similar to the previous case, we use the FLORASAT-CLI to generate graphics, which we will discuss in detail next.

a) Throughput and Distance: Figure 6-a) compares the average throughput between the two algorithms. The results indicate that DDRA and DISCOROUTE have comparable throughput performance, with a slightly better performance in favor of DISCOROUTE. This indicates that both algorithms can maintain similar throughput under the defined scenario, even when Figure 6-b) shows that the average route length of DISCOROUTE algorithm overpass DDRA. However, since DISCOROUTE is not based on Dijkstra's shortest path algorithm for calculating the path but uses a much more efficient algorithm, this result favors DISCOROUTE, as the differences are only minor.

Leveraging the preprocessed routes generated through FLORASAT-CLI, we extract and analyze the shortest and longest routes packets take. Figure 7 shows the routes and hops through the satellite constellation. Projection angles were adjusted to enhance visualization. Figure 7-a) evidences DDRA routing's behavior of prioritization of the shortest path, while in Figure 7-b), the Disco behavior algorithm favors transitions near the poles. This capability enables detailed analysis across the network, providing insights into the routing algorithms and identifying potential bottlenecks.

b) Packet Loss Comparison: Finally, Figure 6-c) shows the packet loss for the previously mentioned high failure probability. Under the given scenario conditions, both algorithms demonstrate similar packet loss performance. DDRA shows a slight advantage in reducing packet loss compared to DISCOROUTE. This highlights that the failure response capabilities built into the algorithm impact overall performance. The relatively small performance differences are because the constellation is large, and the probability for DISCOROUTE calculating a path that contains failed links is still relatively low. We expect the performance gap between DDRA and DISCOROUTE to increase with higher link failure probabilities as the areal distribution of failed links also increases.

C. FLORASAT *2 Limitations*

Despite the potential of these new modules, they have some limitations. Their use requires proficiency in C++ and OMNET++, which can pose a steep learning curve for unfamiliar users. Additionally, the system requirements for large-scale simulations are demanding, potentially impacting performance and execution time. While OMNET++ is designed for serial execution inherent to discrete-event simulations, it supports multiple simulations running in parallel and allows multithreaded execution of C++ routines within individual events. Lastly, the visualization and statistics module is limited to comparing two algorithms simultaneously, restricting the scope for more complex and comprehensive evaluations.

VI. CONCLUSION AND FUTURE WORK

The main contribution of this work is the development and enhancement of FLORASAT 2, an advanced publicly available simulation tool tailored for analyzing cross-linked DTS-IOT constellations. This includes essential new modules, such as constellation creation, dynamic topology control, routing, and event management, which enable precise and flexible modeling of large-scale DTS-IOT satellite networks. Additionally, integrating a flexible routing sandbox facilitates the seamless implementation and evaluation of diverse routing algorithms tailored for constellation-grade DTS-IOT scenarios.

A standout contribution is the FLORASAT-CLI module, a robust Python-based analytics and visualization tool used to analyze two representative use cases: the evaluation of Starlink and Iridium topologies. These use cases demonstrate FLORASAT 2's unique ability to validate routing algorithms and assess complex constellation designs effectively. Together, these advancements position FLORASAT 2 as a versatile end-to-end simulation platform, providing researchers with a comprehensive resource for analyzing and optimizing satellite communication and IoT networks.

Future developments should focus on integrating MAC layer capabilities with routing algorithms to enhance the realism of simulation scenarios further and kick off the study of ISL-aware MAC techniques. This improvement allows detailed studies of the interaction between MAC protocols and network performance, supporting the exploration of adaptive MAC schemes and machine learning-driven optimizations. By incorporating these enhancements, FLORASAT 2 will solidify its role as a pivotal tool for next-generation DTS-IOT systems.

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 101008233 (MISSION) and the French National Research Agency (ANR) under the project ANR-22-CE25-0014-01.

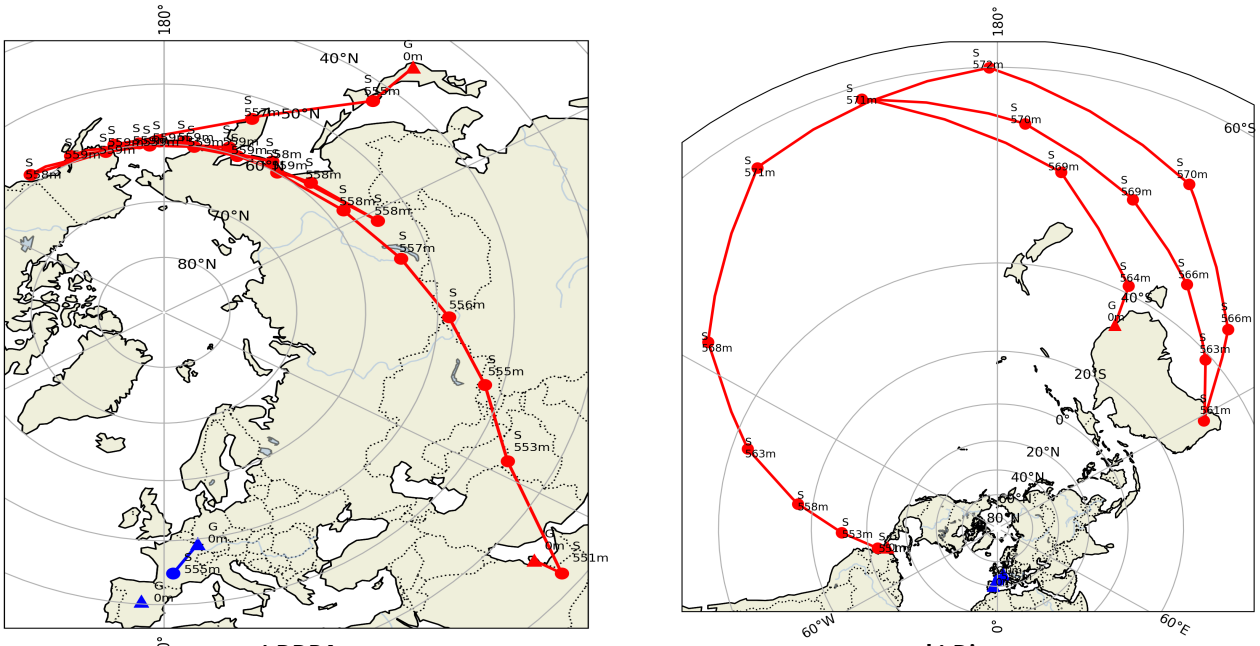

Fig. 7. Analysis of routes preprocessed by FLoRaSat-CLI

References

[1] I. e. a. Leyva-Mayorga, "Leo small-satellite constellations for 5g and beyond-5g communications," *Ieee Access*, vol. 8, 2020.

[2] M. R. Palattella and N. Accettura, "Enabling internet of everything everywhere: Lpwan with satellite backhaul," in *2018 Global Information Infrastructure and Networking Symposium (GIIS)*. Thessaloniki, Greece: IEEE, 2018, pp. 1–5.

[3] M. e. a. Centenaro, "A survey on technologies, standards and open challenges in satellite iot," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1693–1720, 2021.

[4] M. e. a. De Sanctis, "Satellite communications supporting internet of remote things," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 113–123, 2015.

[5] J. A. Fraire, S. Céspedes, and N. Accettura, "Direct-to-satellite iot-a survey of the state of the art and future research perspectives: Backhauling the iot through leo satellites," in *International Conference on Ad-Hoc Networks and Wireless*. Springer, 2019, pp. 241–258.

[6] J. A. e. a. Fraire, "Simulating lora-based direct-to-satellite iot networks with florasat," in *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. Belfast, United Kingdom: IEEE, 2022, pp. 464–470.

[7] M. e. a. Slabicki, "Adaptive configuration of lora networks for dense iot deployments," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. Taipei, Taiwan: IEEE, 2018, pp. 1–9.

[8] SpaceX and FCC, "Starlink Amendment Phase 1," https://fcc.report/IBFS/SAT-MOD-20181108-00083/1877671.pdf, 2018, [Accessed 05-11-24].

[9] M. S. C. Inc and FCC, "Iridium Amendment," https://fcc.report/IBFS/SAT-AMD-19920808-00021/1162673.pdf, 1991, [Accessed 05-11-24].

[10] Ansys Inc., "STK (Systems Tool Kit)," 2024, [Accessed 29-10-24]. [Online]. Available: https://www.ansys.com/products/missions/ansys-stk

[11] S. e. a. Kassing, "Exploring the "Internet from space" with Hypatia," in *ACM IMC*, 2020.

[12] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.

[13] Thales Group., "OpenSAND," [Accessed 15-11-24]. [Online]. Available: https://www.opensand.org/references.html

[14] NASA, "General Mission Analysis Tool (GMAT)," [Accessed 15-11-24]. [Online]. Available: https://opensource.gsfc.nasa.gov/projects/GMAT/index.php

[15] L. Maisonobe, V. Pommier, and P. Parraud, "Orekit: An open source library for operational flight dynamics applications," in *4th international conference on astrodynamics tools and techniques*. European Space Agency Paris, 2010, pp. 3–6.

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

[17] J. A. Fraire, "Introducing contact plan designer: A planning tool for dtn-based space-terrestrial networks," in *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*. Madrid, Spain: IEEE, 2017, pp. 124–127.

[18] K. Vogelgesang, J. A. Fraire, and H. Hermanns, "Uplink transmission probability functions for lora-based direct-to-satellite iot: A case study," in *2021 IEEE Global Communications Conference (GLOBECOM)*. Madrid, Spain: IEEE, 2021, pp. 01–06.

[19] J. e. a. Janhunen, "Satellite uplink transmission with terrestrial network interference," in *2015 IEEE global communications conference (GLOBECOM)*. San Diego, California: IEEE, 2015, pp. 1–6.

[20] F. OneWeb, "OneWeb Amendment Phase 1," https://fcc.report/IBFS/SAT-MPL-20200526-00062/2379565.pdf, 2020, [Accessed 05-11-24].

[21] ——, "OneWeb Amendment Phase 2," https://fcc.report/IBFS/SAT-MPL-20200526-00062/2379706.pdf, 2020, [Accessed 05-11-24].

[22] Q. e. a. Chen, "Topology virtualization and dynamics shielding method for leo satellite networks," *IEEE Communications Letters*, vol. 24, no. 2, pp. 433–437, 2020.

[23] M. e. a. Werner, "Analysis of system parameters for leo/ico-satellite communication networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 2, pp. 371–381, 1995.

[24] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artificial intelligence*, vol. 1, no. 3-4, pp. 193–204, 1970.

[25] H. Tan and L. Zhu, "A novel routing algorithm based on virtual topology snapshot in leo satellite networks," in *2014 IEEE 17th International Conference on Computational Science and Engineering*. Chengdu, China: IEEE, 2014, pp. 357–361.

[26] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: his life, work, and legacy*, 2022, pp. 287–290.

[27] G. Stock, J. A. Fraire, and H. Hermanns, "Distributed on-demand routing for leo mega-constellations: A starlink case study," in *2022 11th Advanced Satellite Multimedia Systems Conference and the 17th Signal Processing for Space Communications Workshop (ASMS/SPSC)*. Graz, Austria: IEEE, 2022, pp. 1–8.