Saarland University

Masterarbeit

Routing in Low-Earth Orbit Constellations: A Performance Comparison

Robin Ohs

Dependable Systems and Software
Saarland Informatics Campus

Betreut von: Prof. Dr. Holger Hermanns
Prof. Dr. Juan A. Fraire

Version: October 8, 2023

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken,_____ _____

(Datum/Date) (Unterschrift/Signature)

Abstract

Modern societies and economies require fast, reliable and continuous connectivity for dependable communication and business development. This necessity extends to remote areas, such as at sea or in the air, and particularly in developing countries. Connecting ships or airplanes to the fiber-optic network is not feasible, just as it is almost impossible to provide a connection for every individual via wired links. One solution are Low-Earth orbit satellite constellations, which, thanks to their low altitude and ability to exchange traffic using inter-satellite links, enable continuous and fast connections even in the most remote areas or on ships and aircraft.

Finding a suitable path from a source through several intermediate satellites to a last satellite that finally transmit the data to a destination is a non-trivial task and is referred to as routing. Routing in general, is a well-researched area, but since the topology is subject to frequent changes due to the Earth's rotation and the movement of satellites, algorithms capable of coping with these evolving topologies are needed. Numerous algorithms for routing in low-Earth orbit have been proposed in the past, but the performance classification between those algorithms is hard because there is no general methodology, including benchmark scenarios, simulation platform, and evaluation questions, that allow for a comprehensive comparison. Instead, it is common practice to create a custom simulator for the algorithm study and evaluate a handful of characteristics. This approach lacks comparability with peer algorithms because the modeled orbital behavior of the satellites, the behavior of the inter-satellite links, and the packet processing implementation are not equal, resulting in potential performance impacts.

In order to achieve a better comparability, this work investigates whether it is possible to develop a generally applicable methodology by proposing a set of benchmark scenarios with associated evaluation questions, together with the development of an extensible simulation platform that enables the creation of dynamic low-Earth orbit Walker constellations, the dynamic establishment and tear down of inter-satellite links and a comprehensive collection of statistics.

To validate that the proposed set of benchmarks, in conjunction with the developed platform, enables the desired comprehensive study of algorithm properties and performance, the description of the platform development is followed by a comparison of two routing algorithms for Walker-Star and two for Walker-Delta using the created simulator. Since it has been shown that the simulator, first, can be extended with any low-Earth orbit routing algorithm due to its modularity, and, second, allows the creation of a consistent constellation and routing topology environment, a satisfactory level of comparability is established that allows the desired, general applicable comparison. Future research can utilize the developed methodology to eliminate the need to create benchmark scenarios and develop simulators, resulting in a decrease in overall research efforts. Researchers will be able to add new algorithms as modules and focus on improving them instead of spending time creating a test environment. Furthermore, as far as the established benchmarks are considered, a minimum level of comparability between algorithms should be established.

Table of Contents

List of Figures

List of Listings

List of Tables

List of Acronyms

RTT Round Trip Time

TTL Time to Live

CRA Constellation Routing Algorithm

E2E End-to-End

MSN Manhattan Street Network

DSPA Dijkstra Shortest Path Algorithm

DDRA Dynamic Detection Routing Algorithm

DISCO Distributed On-Demand Routing for Mega-Constellations

LEO Low Earth Orbit

MEO Medium Earth Orbit

GEO Geostationary Earth Orbit

RAAN Right Ascension of the Ascending Node

ECEF Earth-centered, Earth-fixed coordinate system

LLA Latitude, Longitude and Altitude

ISL Inter-Satellite Link

GSL Ground-Satellite Link

RX Receiver

TX Transmitter

API Application Programming Interface

DES Discrete Event Simulator

GS Ground Station

CLI Command-line Interface

CDF Cumulative Distribution Function

1 Introduction

Continuous connectivity has become a crucial need for the modern society. People in all parts of the world and even in in the most remote areas strive for fast and reliable internet access to communicate and collaborate with far distant others. Connecting all these people through fiber is a seemingly impossible task. One solution for providing global connectivity is the use of satellite systems, which can provide wireless connectivity in areas without well-developed infrastructure. For decades, most satellite systems were deployed in high orbits such as Geostationary Earth Orbit (GEO) at 35786 km altitude, but because of the long distance between the satellite and the transmitters on Earth, the data exchange suffers from long signal delays. Nowadays, there is a trend to deploy satellites in Low Earth Orbit (LEO) with altitudes of less than 2000km [1], resulting in lower Round Trip Times (RTTs) compared to satellites in higher orbits [2]. This is in particular useful to meet the increasing demand for real-time connectivity, which is highly dependent on small RTTs. However, the deployment of satellites in LEO comes at a cost. Satellites orbiting at lower altitudes have smaller surface footprints, resulting in a smaller covered area, which leads to two problems. First, while satellites move along their trajectory, the area on Earth they cover is moving as well. Therefore, a significant number of satellites with tailored orbital parameters are required to ensure continuous coverage, so that if one satellite becomes invisible, the link can be handed over to another. Such a pre-planned deployment of satellites forms a constellation, but as the number of satellites grows, so do the launch and operation cost. Second, satellites in LEO are no longer suitable for operation as relays for mirroring data between a transmitter and a ground station. A relay requires that both the transmitter and receiving ground station are connected to the same satellite to enable communication between them. Thus, a constellation operator would have to operate a large number of ground stations in all parts of the world to cover all satellites simultaneously, which is not feasible.

The aforementioned problems can be mitigated by a number of technological advances that the space industry has made in recent years. Reusable rockets have dramatically reduced the average launch costs per satellite [3], allowing satellite constellation providers to build larger and denser satellite constellations than ever before without massively increasing the deployment costs. One of the best-known examples for these mega-constellations is Starlink, with over 3000 active deployed satellites. In addition, modern satellites are capable of exchanging data with each other via Inter-Satellite Links (ISLs), which allows data to be transported over several satellite hops until a satellite within range of a ground station is reached. Finding the optimal sequence of successive satellite hops is a non-negligible problem and is referred to as routing. Longer routes and unnecessary hops lead to higher RTTs, less throughput and eventually congestion in the constellation. Although routing is a well researched topic, the routing topology of a LEO satellite constellation is subject to frequent changes due to the orbital environment of the satellites. These changes cannot be handled by conventional routing algorithms that were not designed for such purposes. Therefore, algorithms like Distributed On-Demand Routing for Mega-Constellations (DISCO) [4] or Dynamic Detection Routing Algorithm (DDRA) [5] that can cope with these changes are necessary to find optimal routes even in evolving topologies.

1.1 Problem Formulation and Research Contribution

Numerous routing algorithms for LEO constellations have been proposed in various scientific papers in recent years. Some of the proposed algorithms are said to offer better performance than peer algorithms, but there is a lack of a general methodology to make the performance of Constellation Routing Algorithms (CRAs) comparable. Prior research, such as [6], have approached a comparison among a subset of LEO CRAs but their results did not yield the desired general methodology nor did they provide a comprehensive evaluation of the investigated algorithms. Moreover, the paper dates back to 2000 and satellite technology and routing requirements have evolved since then. This work attempts to provide such a general methodology by proposing a set of comprehensive benchmarks that allow the exploration of algorithm characteristics in presence of plannable and unplannable events, along with the creation of a platform, where arbitrary CRA for LEO can be added as modules and tested against the proposed benchmarks. For this purpose, a software able to simulate orbital mechanics of satellite constellations, their evolving topology, as well as modelling traffic patterns is realized. In addition, the software needs to be capable to support the creation and replay of several different scenarios (benchmarks), the collection of important metrics like E2E delay, throughput or hop count, but also to test the reaction of the system to unplanned events such as ISL failures or complete satellite outages.

An existing software that already implements some of the desired capabilities of the platform is the C++ framework FLoRaSat [7]. It provides an implementation of orbital mechanics and the ability to project satellite positions onto a user interface. Furthermore, it is based on Omnet++ [8], a discrete event simulator, which is well known for the study and simulation of networks and routing algorithms. Unfortunately, despite the implementation of orbital mechanics important required functionalities are missing. There is no module for dynamic creation of satellite constellations, no support for frequent updates of the constellation routing topology, no generator for traffic patterns and finally no scenario scripting capability that allows the creation of unplannable failures of links or entire satellites. After the addition of the described missing functionality during this work, the platform will then allow a detailed analysis of implemented LEO routing algorithms, with help of metrics collected during the benchmark runs. This helps to evaluate the tested algorithms in terms of performance, scalability, as well as network and economic efficiency and thereby contribute to the ongoing research in the field of LEO satellite constellations. Since the platform is open to extensions and provides a simple routing Application Programming Interface (API), related future work is enabled to add new algorithm modules to the platform and use its capabilities to compare and test their algorithms without the requirement to develop their own testing infrastructure. In addition, the general methodology will allow for a more comparable evaluation of new algorithm proposals when the authors consider the proposed comprehensive set of benchmarks and associated evaluation questions.

1.2 Research Questions

With help of the aforementioned problem formulation and research contribution, this works overall main research question can be formulated as the following:

RQ Is it possible to develop a general applicable methodology that allows the research of various LEO routing algorithms for single-shell Walker constellations by enabling a comprehensive and reproducible comparison of their performance?

For a more effective approach, this thesis main objective is divided into smaller units of work. As a result, three uterior tasks arise, each with a corresponding research question. These uterior tasks are:

RQ1 What are suitable general applicable benchmarks which allow a representative exploration of features and characteristics of LEO routing algorithms?

RQ2 How can a simulation tool be developed that allows the dynamic creation of Walker-Star and Walker-Delta satellite constellations, the addition of LEO routing algorithms, and the exploration and evaluation of their performance using simulation and visualization techniques?

RQ3 Can the developed platform be used to perform a comprehensive comparison of the performance of three representative LEO routing algorithms by leveraging the established benchmark scenarios for both Walker-Delta and Walker-Star constellations?

1.3 Outline

The following of this thesis is structured as follows:

- Sec. 2 (Background) provides the necessary theoretical background for understanding the fundamentals of satellite systems, constellations, and the process of routing in general by introducing the reader to substantial orbital mechanics, coordinate systems, communication devices, fundamental routing algorithms, and graph theory, as well as the concept of discrete event simulation.
- Sec. 3 (Related work) is split into three subsections, each associated with one of this work's minor research questions:
  - Subsection one reviews existing benchmark scenarios that have been proposed and utilized in related scientific work, including collected and compared metrics.
  - Subsection two provides an overview of available simulators for LEO routing algorithms, including their respective characteristics, advantages and drawbacks, together with a short overview of their architecture as far as it can benefit this work's methodology.
  - Subsection three presents two proposed routing algorithms, namely DDRA [5] and DISCO [4], which will be used later together with Directed Routing to validate the implemented capabilities of the simulation platform.
- Sec. 4 (Benchmarks) proposes a set of benchmark scenarios and metrics that combine identified scenarios from related work with with novel ideas, and discusses how they contribute to a comprehensive comparison of LEO routing algorithms.
- Sec. 5 (Simulation Platform) describes the performed software development to extend FLoRaSat with the required capabilities, including developed modules, changes to

the architecture and implemented methods to collect statistics. Furthermore, the creation of a library for result pre-processing, and the development of a Command-line Interface (CLI) for generating various visualizations of collected results is described.

- Sec. 6 (Algorithm Comparison) employs the proposed benchmark scenarios and associated evaluation questions along with the developed simulator to conduct a thorough analysis and comparison of DDRA and DISCO for Walker-Delta, and DDRA and Directed Routing for Walker-Star to validate the contribution of this work.
- Sec. 7 (Discussion) discusses whether this work has achieved its objectives and answers the posed research questions. In addition, key findings and contributions are listed, as well as a description of found limitations of this approach and possible future work.

2 Background

2.1 Satellite 101

In theory, all objects that revolve around another larger body can be referred to as satellites, e.g., the Earth is a satellite because it orbits the Sun. However, when people talk about satellites, they usually refer to artificial objects launched into space by mankind, rather than planets [10, 9]. The first satellite, Sputnik 1, was launched in 1957 and remained in space for only 96 days. Since then, satellite and rocket technology has evolved, enabling the launch of superior and larger quantities of satellites with similar deployment expenses.
Modern satellites are equipped with solar panels and batteries to provide continuous services and power hardware. This later becomes particularly useful for switching traffic and for running routing protocols that require the satellite to perform its own complex calculations.

2.1.1 Orbital Mechanics

To understand the design and deployment of constellations it is necessary to have a foundational understanding of orbital mechanics. First discovered by Copernicus in 1543 and later refined by Kepler's laws of planetary motion between 1609 and 1619, planets follow a trajectory around their primary body called an orbit, which is the result of physical forces. Satellites are similar to planets since they orbit their primary body (Earth in the context of this work) and are subject to the same physical laws. Kepler established six parameters that define an orbit, known as the six orbital elements: the semi-major axis a, eccentricity e, inclination i, Right Ascension of the Ascending Node (RAAN) Ω, argument of perigee ω, and true anomaly θ. In [11], the Federal Aviation Administration (FAA) categorizes these parameters into four categories of interest:

- Orbit's size (semi-major axis).
- Orbit's shape (eccentricity).
- Orbit's orientation (inclination, RAAN, argument of perigee).
- Spacecraft's position (true anomaly).

Further, the FAA gives the definition of some orbital elements by referring to an inertial coordinate system, known as geocentric equatorial coordinate system. Two of the three vectors defining the coordinate system lie in the equatorial plane, while the third vector is perpendicular to this plane. One of the vectors in the equatorial plane points towards the vernal equinox, and the other is determined using the right-hand rule. Finding the vernal equinox can be achieved by drawing a line through the Earth and Sun on the first day of spring. According to the FAA these categories can be categorized as follows:

Orbit's size and shape According to Kepler's Third Law, the time required to complete one orbit is known as the orbit period (T), which remains constant in absence of external forces. The shape of the orbit can be determined by the eccentricity (e), which describes the deviation from a circular shape, thus an orbit with $e = 0$ is circular. Two points of interest on orbits are the point closest to the central body, the perigee, and the point farthest away, the apogee. During an orbit, the distance between a satellite and its central body increases on the path from perigee to apogee and decreases from apogee to perigee.

These points exist only in elliptical orbits, because in circular orbits it is impossible to determine the nearest and the farthest point. As with any ellipse, there are designations for the longest and shortest axes of the ellipse, the major axis and the minor axis, with half of each of these axes being called the semi-major axis and the semi-minor axis, respectively. As illustrated in fig. 1, the axes in circular orbits are of equal length, and can be referred

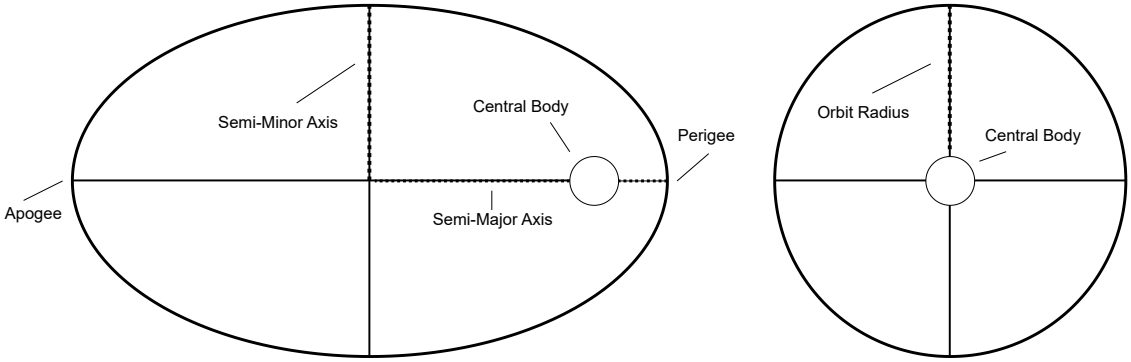


Figure 1: Elliptical orbit ($e > 0$) and circular orbit ($e = 0$) comparison.

to as the orbit radius. Due to equal length of the axes, the distance between the satellite and the center of the central body will remain constant and includes the radius of the central body (the radius of the Earth). However, more relevant is the distance between a satellite and a point on the Earth's surface, which is called altitude. For circular orbits, the altitude of a satellite Alt_{Sat} is defined as the difference between the radius of the orbit R_{Orb} and the radius of the central body R_{CB}: $Alt_{sat} = R_{Orb} - R_{CB}$. This is not fully accurate, because the Earth is not a perfect sphere, but rather an ellipsoid, meaning it has no unique radius. In [12], Earle examined the influence of employing a spherical Earth model in comparison to a spheroidal model defined by the WGS84[1] ellipsoid on navigation accuracy and concluded that the difference is small (about 0.5%). Since the deviations are small, it is possible to define an average Earth radius R_E equal to 6371.137 km, known as the volumetric mean radius[2]. Therefore, on Earth, R_{CB} can be approximately set to R_E.

Orbit's orientation Another important orbital parameter is the inclination, which measures the angle between the orbital and equatorial plane and allows to specify the orientation of the orbit in relation to the equatorial plane. Valid values range from 0° to 180°. If the inclination is equal to or greater than 0° but less than 90°, the satellite moves with Earth's rotation, while for inclinations greater than 90°, the satellite moves in the opposite direction to the rotation of Earth. An inclination of 90° results in a polar orbit since the satellite ascents toward the north pole and descents toward the south pole. This definition is not strictly bound to 90° because all kind of orbits with an inclination close to 90° are classified as polar orbits by other works [13, 14]. Two special cases are 0° and 180° where the orbital plane is identical to the equatorial plane.
To calculate the RAAN, the location of two specific nodes on the orbit is required: the ascending and the descending node. These points are located on the line given by the intersection of the orbital and equatorial plane. On non-inclined orbits ($i = 0°$ or $i = 180°$), these points do not exist due to the equality between the equatorial and orbital plane.

[1] `https://earth-info.nga.mil/index.php?dir=wgs84&action=wgs84`
[2] `https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html`

The ascending node is the point at which the orbit crosses the equatorial plane from the southern to the northern hemisphere, whereas the descending node is the point at which the orbit crosses the equatorial plane from the northern to the southern hemisphere. Measuring the angle (in the direction of Earth's rotation) between the vector of the coordinate system pointing to the vernal equinox and an imaginary vector from the center of the central body to the ascending node gives the RAAN (Ω) with $0° \leq \Omega < 360°$.
The last orbital element of interest is the argument of perigee ω, which gives the angle between the perigee and the ascending node, measured in the direction of orbital motion, with $0° \leq \omega < 360°$. For circular orbits the argument of the perigee is undefined, because the perigee is undefined, but also not needed for the determination of the circular orbit.

Spacecraft's position With the previously defined elements it is possible to precisely define an orbit, but what is missing is a way to define a satellite's position on the orbit. The position can be determined by the *true anomaly* θ, given by the angle between an imaginary vector from the center towards the perigee and an imaginary vector from the center towards the spacecraft with $0° \leq \theta < 360°$. Unlike other orbital elements, which are constant without external acting forces, this parameter evolves over time because a satellite is constantly moving. Another approach is to use the so-called *mean anomaly*, given as the angle between a vector from the center to the spacecraft and a vector from the center to the perigee on an imaginary circular orbit with the same location of perigee and orbital period as the actual elliptical orbit. On almost circular orbits the *mean anomaly* and *true anomaly* are approximately equal.
Again, neither the *mean anomaly* nor the *true anomaly* can be used with circular orbits because they are relative to the undefined perigee. To resolve this issue, an alternative orbital element to specify a spacecraft's position that is also defined on circular orbits is required. This element is the *argument of latitude* u and is given as the angle between the spacecraft vector and the ascending node vector. On elliptic orbits, u can also be calculated as the sum of θ and ω. An alternative way is to define the perigee in circular orbits to be equal to the ascending node, which allows an interchangeably usage of *true anomaly*, *mean anomaly* and *argument of latitude* since they specify the same position.

2.1.2 Orbit altitude types

Satellite orbits can be classified into several types based on their altitudes, including Low Earth Orbit (LEO), Medium Earth Orbit (MEO) and Geostationary Earth Orbit (GEO) [9, 15, 16]. The exact boundaries between these types are not well-defined, as their classifications do not match exactly, but the tendency is similar. Approximately, satellites in LEO have altitudes between 160km and 2000km with velocities around 7.8km/s resulting in an orbital period of about 90min, while orbits such as MEO (altitude from 2000km up to 35780km) or GEO (altitude of 35786km) are much higher with longer orbital periods. The differences shown in tab. 1 affect the connectivity and coverage characteristics provided by satellites on the particular altitudes. Satellites in LEO are closer to ground stations, resulting in faster signal reception and thereby decreasing the E2E latency through shorter propagation delays. It is possible to reduce the signal delay from 120ms in GEO down to 3.7ms using LEO (at an altitude of 550km) [18, 17]. On the other hand, because of the lower altitude, the visible coverage of a satellite in LEO is much smaller, so hundreds of deployed satellites are needed to provide global and continuous coverage, while in GEO three satellites are sufficient.

Property	Orbit types:	LEO	MEO	GEO
Altitude		Low	Medium	High
Coverage		Small	Large	Very Large
Deployment costs (per satellite)		Low	Medium	High
Required satellites		Hundreds	6	3
Latency		Very Low	Low	High
Satellite Velocity		Fast	Medium	Slower
Orbital Period		Short	Medium	Long

Table 1: Extended comparison of orbit types as illustrated by SES [19]

2.1.3 Earth-centered, Earth-fixed coordinate system

The Earth-centered, Earth-fixed coordinate system (ECEF) is a cartesian coordinate systems suitable for distance calculation and comparing satellite positions [4]. In WGS84 the origin of ECEF is the Earth's center of mass. The Z Axis goes through the south and north pole with the equator equal to 0° latitude. Perpendicular to the Z axis are the X and Y axes, which lie on the equatorial plane, with the X axis pointing to the prime meridian (IERS Reference median in WGS84) equal to 0° longitude. The Y axis can be found by the right-hand rule and is equal to 90° longitude. To obtain ECEF coordinates from Keplerian basic orbital parameters, this work employees the two-step approach described in [4]. First, the Keplerian orbital parameters are converted to latitude φ, longitude λ and altitude h, known as Geodetic coordinates or Latitude, Longitude and Altitude (LLA). Then, the geodetic coordinates are transformed to ECEF cartesian coordinates.

2.2 Constellations

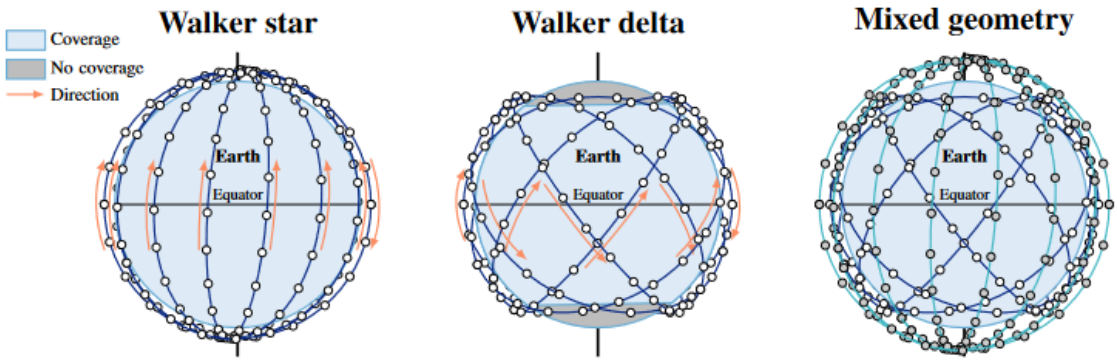


Figure 2: Walker-Star, Walker-Delta and a mixture, illustrated by [20]

With the acquired knowledge of orbital mechanics, it is now possible to understand the design of satellite constellations. Satellites on same orbits with a uniformly shifted *argument of latitude* form a plane, while multiple adjacent planes with uniformly shifted RAAN at equal altitude form a shell [20]. Since constant altitudes allow the reuse of the same transmit power for communications and, according to Palmerini and Graziani, circular orbits provide uniform coverage while elliptical orbits would destroy the symmetry of the constellation, satellites which are intended to be part of a constellation are usually deployed in circular orbits [21]. The eccentricity is considered a "parameter to be vanished". Due to the listed advantages and the utilization of circular orbits in deployed systems like Iridium [18], this work will only consider eccentricities equal or very close to 0.

2.2.1 Walker constellations

Two constellation types that provide continuous coverage have been proposed in the past by Walker, known as Walker-Star and Walker-Delta [22, 23]. Both types utilize circular orbits to provide continuous coverage in their designated areas and allow to maximize the provided connectivity [20]. While Walker-Star constellations enable global coverage, Walker-Delta constellations do not provide coverage in polar regions, but instead have higher satellite densities in more populated areas compared to Walker-Star constellation (see fig. 2). This property is accomplished by the usage of polar orbits in Walker-Star, as opposed to Walker-Delta, where satellites have inclined orbits with inclinations in the range of about 50° to 60°.
Another difference is the range of values used to distribute the RAAN of the orbits, known as the RAAN spread. In Walker-Star, orbits are distributed between 0° and 180°, hence the RAAN difference ($\Delta\Omega$) between neighboring planes is $\frac{180°}{P}$, where P is the number of planes of the constellation. Since Walker-Star does not use the entire possible 360°, two seams are created with counter-rotating satellites on adjacent planes [24, 25]. In contrast, Walker-Delta constellations utilize the full span, resulting in an overlapping mesh. Therefore, $\Delta\Omega$ can be calculated as $\frac{360°}{P}$ [20].
Each plane contains an identical number of satellites uniformly distributed around the entire 360° of the orbit. Hence, the *difference of the argument of latitude* ($\Delta\Theta$) between successive satellites on the same plane is $\frac{360°}{N}$, where N is equal to the number of satellites per plane.
Another property is the *inter-plane spacing*, which is the relative offset of *argument of latitude* between neighboring satellites in adjacent planes, also known as *phase difference* (Δf).

Walker notation One way to describe the structure of a Walker constellation is to use the so-called Walker notation. This notation can be used to precisely define the constellation shape and is given as $i : pq/p/f$. Here, i is the inclination, p the number of evenly spaced planes, q the number of satellites per plane and f the *phasing factor* with f in $[0, p-1]$ [4]. Sometimes the Walker notation is extended with the altitude alt, leading to: $i : pq/p/f/alt$. An overview of the formulas for calculating the orbital parameters for each satellite can be found in tab. 2. To perform all calculations it is sufficient to know the altitude of the shell, the corresponding Walker type, and the Walker notation. From now on, we will denote the i-th satellite in plane p by $sat[p][i]$. Note that satellites in Walker-Star constellations have no left neighbor in the first plane and no right neighbor in the last plane because of the counter-rotating satellites at the seams, which cannot establish ISLs [25, 26, 27].

2.2.2 Inter Satellite Links

In Walker constellations, satellites typically have four neighbors, except on Walker-Star where satellites have three neighbors on the seams (another restriction follows below). The previous and next satellite on the same plane are defined as intra-plane neighbors, while the left and right satellite are defined as inter-plane neighbors. A Satellite is able to communicate with its neighbors via intra-plane and inter-plane ISL. These modules can use either optical lasers or radio frequency beams to transmit data from a Transmitter (TX) to a Receiver (RX). Optical ISL have significant advantages, including their robustness,

Constellation Common Properties	Walker-Star	Walker-Delta
Altitude (alt)	(Provided)	(Provided)
Orbit radius (R_{Orb})	$alt + R_E$	$alt + R_E$
Inclination (i)	(Walker notation: i)	(Walker notation: i)
Number of planes (p)	(Walker notation: p)	(Walker notation: p)
Satellites per plane (q)	(Walker notation: q)	(Walker notation: q)
Phasing factor (f)	(Walker notation: f)	(Walker notation: f)
Number of satellites (n)	$p * q$	$p * q$
Mean anomaly diff. ($\Delta\Theta$)	$360°/q$	$360°/q$
RAAN diff. ($\Delta\Omega$)	$180°/p$	$360°/p$
Phasing offset (Δf)	$360° * f/n$	$360° * f/n$
Satellite $sat[o][i]$		
Altitude	alt	alt
Inclination	i	i
RAAN	$\Delta\Omega$ * o	$\Delta\Omega$ * o
Mean anomaly	$\Delta\Theta$ * i + Δf * o	$\Delta\Theta$ * i + Δf * o
Up neighbor	$sat[o][(i+1) \mod q]$	$sat[o][(i+1) \mod q]$
Down neighbor	$sat[o][(i-1) \mod q]$	$sat[o][(i-1) \mod q]$
Left neighbor ($o \neq 0$)	$sat[o-1][i]$	$sat[o-1][i]$
Left neighbor ($o = 0$)	None	$sat[p-1][(i-f) \mod q]$
Right neighbor ($o \neq (p-1)$)	$sat[o+1][i]$	$sat[o+1][i]$
Right neighbor ($o = (p-1)$)	None	$sat[0][(i+f) \mod q]$

Table 2: Walker constellation parameter calculation.

a much narrower beam which results in better resistance to interference. Another important advantage is that their frequency is not regulated by the ITU[3], which allows easier constellation planning and operation [28]. Furthermore, optical ISL enables data rates ranging from of 10 Gbps to 100 Gbps and low propagation delays due to the quick speed of light $c = 299.792.458$m/s.

Intra-plane ISL On both constellation types the intra-plane ISL are maintained continuously. The distance between one intra-plane hop L_p can be calculated as:

$$L_p = 2 * R_{Orb} * \sin\left(\frac{\frac{2\pi}{q}}{2}\right)$$

This distance is constant and depends only on the number of satellites per plane q and the radius of the orbit R_{Orb}. Thus, it is sufficient to calculate it once for the constellation,

Inter-plane ISL Inter-plane ISL connections are more complex because their connection state varies. Furthermore, their distance depends on the current latitudes of the communication pair, since satellites are closer together at the poles and farther apart at the equator. A suitable method to calculate their distance is to use the euclidean distance for 3-dimensional space. Referring back to ECEF, let satellite A be at (X_1, Y_1, Z_1) and the right neighbor B at (X_2, Y_2, Z_2), then the inter-plane distance is given by:

$$L_{ip} = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2}$$

[3]International Telecommunication Union

Walker-Delta constellations are able to maintain the ISL continuously, while Walker-Star constellations cannot. The reason is that in polar regions ($lat. > 70°$) the inter-plane ISLs are required to shut down due to limited steering capabilities of the antennas [13, 29, 26].

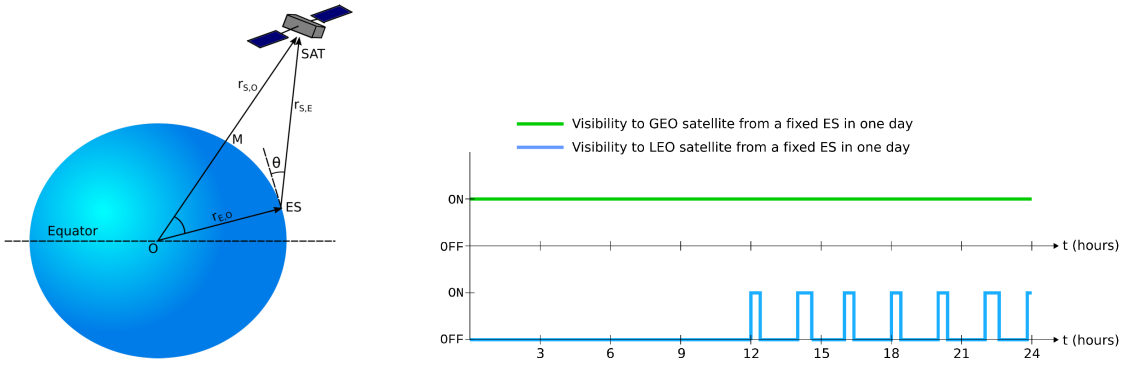
2.2.3 Ground-Satellite links

Another important satellite module is the communication module for exchanging data with a Ground Station (GS). This module has several names used among research works, such as up/downlink [30] or Ground-Satellite Link (GSL) [31]. In the context of this work, we will stick with GSL. GSL connections are only possible if a satellite is in sight of a GS. This is determined by the elevation angle, which must exceed a certain threshold, known as the minimum elevation e_{min} [32]. The elevation is relative to the position of the GS and the position of the satellite and therefore, the value must be recalculated whenever checked for a certain point of time and satellite-GS pair, since the elevation is constantly changing. Additionally, since the elevation depends on the position, although two constellations may have identical e_{min} requirements, a satellite in one constellation may become visible earlier and remains visible longer due to its higher orbit altitude than a satellite in the other constellation.

Elevation Calculation According to Gongora-Torres et al., the elevation can be defined as the angle between the local horizon of the GS and the satellite, and can be calculated as:

$$e = \arctan\left(\frac{\cos(\Delta)\cos(\Phi_{GS}) - (R_E/R_{Orb})}{\sqrt{1-\cos^2(\Delta)\cos^2(\Phi_{GS})}}\right)$$

where Φ_{GS} is the latitude of the GS, and Δ is the latitude difference between the GS and the subsatellite point M [33]. Fig. 3 (a) shows the relevant locations and lengths for elevation calculation, while (b) shows an exemplary visibility graph between a GS and two satellite, one in LEO and one in GEO. (b) shows that the higher altitude results in more visibility and single LEO satellites are not suitable to provide continuous coverage. It is important to note that this illustration refers to GSs as Earth Stations (ES).



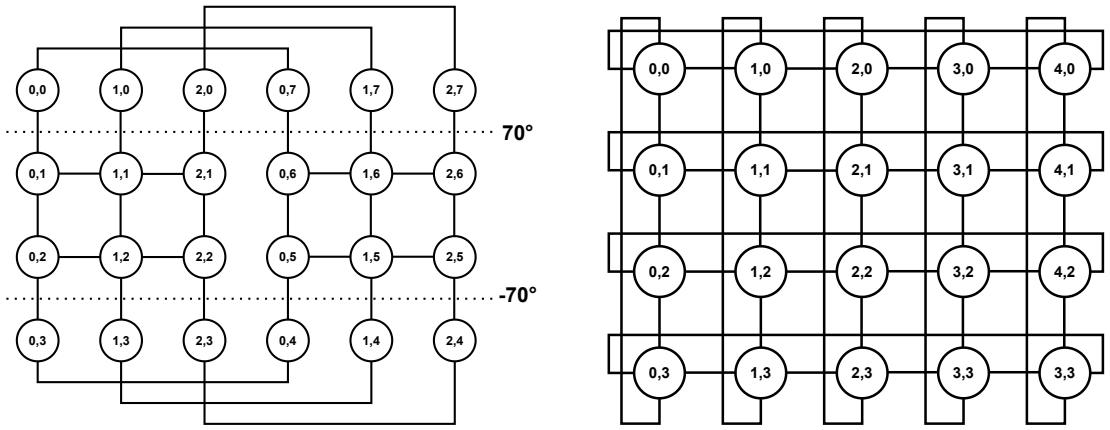
(**a**) Elevation angle, θ (**b**) Link availability as seen from a fixed ES

Figure 3: Elevation and subsatellite point illustrated by [33]

2.3 Graph theory

Interconnected satellites create a mesh network, also known as a topology [28]. Satellites can be represented as nodes (sometimes called vertex) and the ISL interconnections as edges between the nodes. Edges can be either unidirectional, where A can send to B but not the other way around, or bidirectional, where A can send to B and vice versa. ISLs are usually bidirectional, but due to problems with a transmitter or receiver, links may become temporarily or permanently unidirectional. Sending traffic from one node to another node is not free as the transmission is not instantaneous but comes with a certain delay, the propagation delay. In graphs, this can be represented by annotating edges with a certain cost, which could be equal to the distance between the nodes or the propagation delay of the transmission and is commonly called weight. Graphs with weighted edges are known as networks, and have been the subject of research for a long period of time, as evidenced by the Seven Bridges of Königsberg by Euler in 1736.

2.3.1 Manhattan Street Networks



(a) Walker-Star 24/3/0 *(b) Walker-Delta* 20/5/0

Figure 4: Sample Walker constellation's Manhattan Street Network, inspired by [14, 34].

One way to represent a constellation topology is the usage of a Manhattan Street Network (MSN). Fig. 4 shows two MSN representing a Walker-Star and a Walker-Delta constellation. Each satellite is represented by a node and is assigned a unique identifier within the network, consisting of the index of the plane and the number of the satellite in the plane. The Walker-Star constellation (a) is an irregular MSN, since the nodes are not connected at the seams, and due to the already discussed limited steering capabilities of the inter-plane ISLs in polar zones (see 2.2.2), the nodes above 70° and below -70° are only connected via intra-plane ISLs. The inclined Walker-Delta constellation (b), on the other hand, can be represented by a cyclic MSN [14], since due to the full use of the RAAN spread, there are no seams that would lead to missing inter-plane connections. In addition, the satellites' inclinations cause non-polar orbits, thus removing the need to disable ISLs in polar regions. If a Walker-Star constellation is used, which is capable of creating connections at the seams, it would be necessary to add cyclic horizontal edges to the network in the non-polar regions.

Manhattan Distance An important metric in MSNs is the Manhattan distance, which specifies the distance between two coordinates. In the context of routing topologies represented by a MSNs, the Manhattan distance is equal to the minimal number of hops between two satellites with specified coordinates. The distance between node $A = (A_x, A_y)$ and $B = (B_x, B_y)$ in a regular non-cyclic MSNs can be calculated as $|A_x - B_x| + |A_y - B_y|$. However, since Walker-Star produces a cyclic irregular network and Walker-Delta produces a cyclic but regular network, this formula does not yield the smallest possible distance. Therefore, it is necessary to distinguish between the two cases. Assuming a network with v planes and w satellites per plane, for Walker-Star the distance between A and B can be calculated as:

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

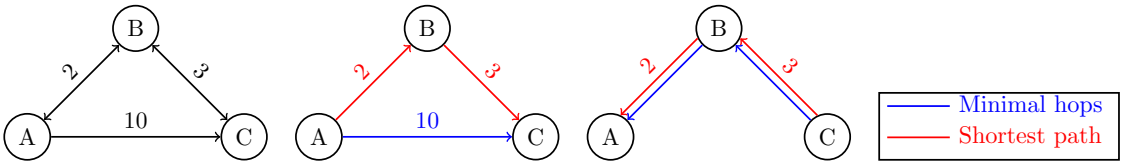
where d_1 is the direct distance between the coordinates, d_2 is the distance if the top edge is taken to the bottom of the cyclic network, d_3 is the distance if the bottom edge is taken to the top of the cyclic network. The actual distance d is then given as the minimum of $d1$, $d2$ and $d3$. A remaining problem is that the lack of inter-plane ISLs in polar regions is not taken into account, so the result is only an approximation or underestimate of the actual distance. For Walker-Delta, on the other hand, it is also required to take the cyclic edges of the nodes in the first and last plane into account, which are not present in Walker-Star because of the seams. The formula extends the formula for Walker-Star by adding the new possible paths which utilizes the horizontal cyclic edges from left to right and vice versa. This also creates the possibility of paths that contain a vertical and a horizontal cyclic hop, which must also be taken into account. With that the distance in Walker-Delta can be calculated as follows:

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

Here d_1 to d_3 are the same as in Walker-Star, while d_4 and d_5 consider the left and right horizontal cyclic edges, respectively, without including another vertical cyclic edge. In contrast, d_6 up to d_9 contain a horizontal and a vertical cyclic edge leading the path over the corners of the network, which are $(0, 0)$, $(0, (w-1))$, $((v-1), 0)$ and $((v-1), (w-1))$. Then, the Manhattan distance for regular non-cyclic networks from the corner to the destination is added to the distance needed to reach the corner.

2.3.2 Shortest path problem

A common problem in networking is finding a path from a source node to a destination node, known as path finding or routing. Paths are defined as a series of nodes that are interconnected and can be visited node by node from the source to the destination. Such paths can be arbitrary long and may include cycles and thus are not necessary optimal. More useful are paths without cycles but with a minimal number of hops (taking one edge) or minimal sum of edge weights. The path with least accumulated weight is the shortest path, but may not be equal to the path with fewest hops. In fig. 5, a network with three interconnected nodes is shown. The impact of unidirectional edges is shown by the difference of the path with minimal hops in (b) and (c) due to the unidirectional edge from A to C. Furthermore, it can be seen that the path with minimal hops costs twice as much as the shortest path in terms of weight and thus is not necessarily optimal.

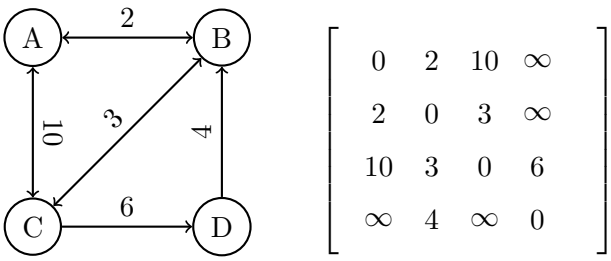


(a) Network *(b) Path from A to C* *(c) Path from C to A* *(d) Legend*

Figure 5: Shortest paths in a network with three interconnected nodes.

2.3.3 Connection Matrix

Network visualizations cannot be used as direct input to programs. Instead, they must be encoded somehow, and one way to encode them is to use of a so-called connection matrix. A network with n nodes can be encoded in a $n \times n$ matrix. Matrix row i corresponds to node i and the value of column x in row i is the weight of the edge from node i to node x. The entry for the edge from a node to itself has the value 0, since it is free to stay on the same node. If there is no edge between two nodes, so that a hop is not possible, then the distance is set to ∞.



$$\begin{bmatrix} 0 & 2 & 10 & \infty \\ 2 & 0 & 3 & \infty \\ 10 & 3 & 0 & 6 \\ \infty & 4 & \infty & 0 \end{bmatrix}$$

(a) Network *(b) Connection matrix*

Figure 6: Connection matrix for a simple topology.

This representation is suitable to be used as direct input to software, since a matrix can be represented by a two-dimensional array or similar structures in programming languages and scales well with large networks.

2.3.4 Dijkstra Shortest Path

Dijkstra Shortest Path Algorithm (DSPA) [35] is a classical algorithm for solving the shortest path problem in networks with weighted edges. The primary concept is to visit the node with lowest cost and update the predecessor and costs of unvisited neighbors if a cheaper path was found. Here, cost is defined as the sum to reach the visited node and the weight of the edge to the neighbor. At the start of the algorithm, the distance is set to infinity for all nodes except the source node, which has an initial distance of 0. The results of DSPA are optimal as long as the network has no negative weights, which is a result of the greediness of the algorithm. Greediness means that the distance to an already visited node is not updated if a cheaper path is found. Algo. 1 shows a sketch of DSPA.

Algorithm 1: Dijkstra shortest path algorithm

```
   Data: n : Const.Size, cm : Con.Matrix, src, dst (Optional)
 1 foreach i in [0..n] do                                   // Init node data
 2     distance[i] ← ∞
 3     ancestor[i] ← −1
 4     visited[i] ← false
 5 distance[src] ← 0
 6 foreach i in [0..n] do
 7     min ← ∞
 8     nearest ← −1
 9     foreach i in [0..n] do                    // Get unvisited nearest node
10         if !visited[j] && distance[j] < min then
11             min ← distance[j]
12             nearest ← j
13     Assert(nearest ≠ −1)
14     if nearest = dst then Break                          // Early abort
15     foreach j in [0..n] do
16         if !visited[j] && cm[nearest][j] ≠ ∞ then
17             alt = min + cm[nearest][j]
18             if distance[j] > alt then
19                 distance[j] ← ∞
20                 ancestor[j] ← −1
21 return distance[ ], ancestor[ ]
```

The algorithm returns two arrays, the *distance* array and the *ancestor* array. The value of $distance[i]$ is the distance from *src* to the node with index i, while the value of $ancestor[i]$ is the index of the direct predecessor node on the path from *src* to i. To obtain the entire path from a source node to a destination node, it is necessary to reconstruct the path using a different algorithm, called DSPA Path Reconstruction, shown in algo. 2. If DSPA is implemented without early abort, the returned arrays can be used to construct the shortest path to any node in the graph from the used source node. However, removing the early abort will increase the average runtime of the algorithm, since it will continue to visit unexplored nodes until all nodes have been visited, even though the shortest path

to the destination node has already been found.

Algorithm 2: DSPA Path Reconstruction

```
   Data: src, dst, ancestor[ ]
1  path ← [dst]
2  last ← dst
3  while last ≠ src do
4  |  last ← ancestor[last]
5  |  path ← last :: path
6  return path[ ]
```

2.3.5 Directed Routing

Another approach to path finding in graphs is to use a heuristic function to select the next node to visit from the open list, rather than the node with the lowest cost from the source. The idea is to discover nodes in the direction of the destination node to find the path faster by defining the heuristic function in such a way that the value gets smaller as the node gets closer to the destination. Directed Routing is a simplistic algorithm that uses the Manhattan distance to the destination as the heuristic in Walker-Star constellations using the equation given in eq. (1). If the algorithm is applied to Walker-Delta constellations (or Walker-Star with inter-seam ISLs), the Manhattan distance for regular cyclic networks must be used, shown in eq. (2). The computed Manhattan distance is only an approximation because the computation function could underestimate the actual distance if an included hop between planes could not be viable, given that inter-plane ISLs are disabled at latitudes over 70° and under -70° and satellites have no information about the location of other satellites.

2.4 The Concept of Routing

This work proposes a methodology to test arbitrary routing algorithms in LEO Walker constellations. Before the development of the methodology is possible, first it is necessary to understand the concept of routing in general. The functionalities of routing algorithms can be divided into the following three categories: *path finding*, *congestion control* and *fault recovery*. It is important to note that each algorithm must implement *path finding*, but not the other two capabilities, as they are not mandatory for providing basic routing functionality during regular operation.

Path finding The process of finding a suitable route from a source to a destination device across multiple intermediate interconnected devices is called *path finding*. It is possible to solve this problem by encoding all connected devices as a network and using aforementioned techniques to find a path from the source to the destination node. In most cases, the optimal route is also the shortest route, found for example with DSPA. However, choosing the shortest path does not always lead to optimal results because of rising congestion or failures along the path. Routing decisions can be made based on either pre-planned data only, pre-planned data combined with dynamic data or dynamic data solely. Dynamic data is data that is collected during operation of the system through mechanisms such as link discovery, *congestion control*, or *fault recovery*. Algorithms relying on pre-planned data are *proactive*, while those with influence from dynamic data can be classified as *reac-*

tive. Combinations of both types are referred to as *hybrid* and are proposed, for example, in [34], where Chan et al. propose a two-stage routing approach, where the first stage uses static interzone routing and the second stage implements adaptive intrazone routing.
Another aspect that can be distinguished between algorithms is the instance that is responsible for making routing decisions. *Centralized* algorithms use a central authority for routing decisions, such as controller satellites that are assigned for a particular zone. On the other hand, an algorithm where each satellite is responsible for making its own routing decisions independently of previous routing decisions is *decentralized*.

Congestion control The transmission and processing of packets (see 2.4.1) is not instantaneous but introduces certain delays (see 2.4.2). For example, routes over nodes that currently handle a lot of traffic (congested nodes) have a high probability of causing additional delays compared to the same route without intermediate congested nodes. As a result, the shortest route may cause more delay for traffic to reach the destination than a route with more but different intermediate hops. Splitting traffic among different routes is called *load balancing*, which can help to reduce congestion and therefore contribute to *congestion control*. In existing research, congestion is usually measured by the size of buffers or queues [36, 5]. Furthermore, queues have a maximum size, and all packets received during full queues are discarded. Insufficient *congestion control* can therefore lead to a high number of undesirable high queuing delays or packet loss.

Fault recovery Since ISLs and satellites are complex technical devices, they can be prone to failures, and due to their location in space, it is not necessarily possible to repair them. Thus, if a ISL link fails and the routing algorithm is unable to detect and handle such events appropriately, this will result in a high number of packet losses as the satellites continue to attempt to route traffic over broken links. This capability is called *fault recovery* and can be further distinguished between the ability to respond to link failures or respond to even complete node failures. It is important to distinguish between temporary and permanent failures. While an algorithm without this capability will recover after the temporary failure is resolved, a permanent failure will degrade the performance of the constellation until the satellite is replaced or if possible manually reconfigured.

2.4.1 Data transmission

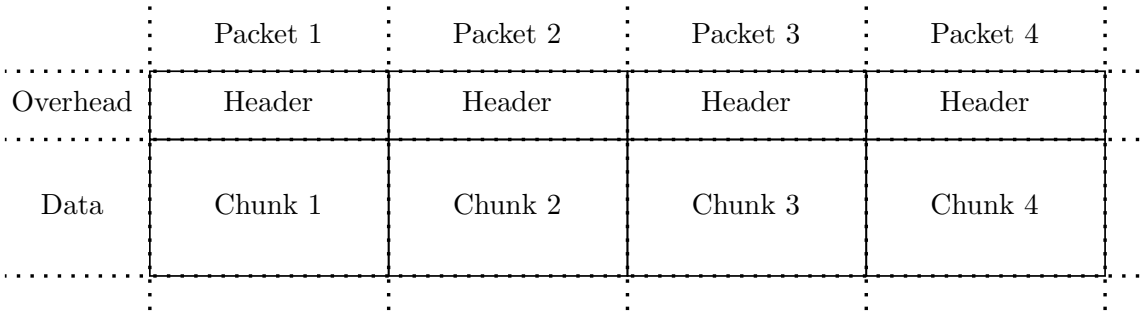


Figure 7: Division of data stream into smaller packets.

Data is usually transmitted as a series of packets. For that, the data is broken into smaller chunks, expanded with a header, and then transmitted, while at the destination

the received packets are sorted and the data is reassembled from the received chunks. Adding headers introduces overhead to each packet, but headers are necessary for the functionality of the protocol because they contain important metadata such as identifiers, the destination address or sequence numbers.

2.4.2 Delays

As previously mentioned, packets are not delivered immediately after initial transmission, due to the delays that occur along the way. There are four different types of network delay which are considered during this work. Namely, the *transmission delay* D_{Trans}, the *processing delay* D_{Proc}, the *queuing delay* D_{Queue} and the *propagation delay* D_{Prop}. These delays are not fixed and vary from transmission to transmission, depending on the number of hops, congestion, distance between satellites, communication module data rates, etc. The E2E delay D_{E2E} of a transmission is the sum of the above delays incurred during transmission from source to destination and is not the same as the RTT, which is the time between sending a packet to a destination and receiving a response back. Together with the utilized packet size the E2E delay is a fundamental metrics to measure the performance and throughput of the constellation. In the following the different delay categories are explained in more detail.

Processing delay Satellites must process packets upon arrival. This may entail verifying the checksum due to link errors, looking up forwarding tables, or even calculating the next hop from stored celestial and gathered network data. Processing delays occur at each node on a packet's path, and the greater the number of hops, the more overall processing delay will appear. Since determining the next subsequent hop can involve complex calculations, the time taken to process a packet header is not negligible and can contribute significantly to the overall delay.

Queuing delay If the rate of incoming packets is higher than the rate of outgoing packets, the affected satellite must delay the processing of the received packets by buffering them in queues, where they are then processed one at a time. The time from the arrival of a packet to the start of processing is called *queuing delay* and depends on the size of the queue and how fast the node can process the previously arrived packets. Unlike the other delays, the *queuing delay* is more variable because it depends on the existing queue and can be 0 if all intermediate satellites are idling upon packet arrival. Nevertheless, the expected *queuing delay* on a satellite can be predicted because the size of the queue is known to the satellite and the average processing time can also be determined. Such predictions are especially useful for *congestion control* mechanisms.

Transmission delay Once the next hop is determined during packet processing, the node must push the packet's bits onto the next transmission channel. If the channel is free, transmission begins immediately, otherwise the packet must be queued again until the channel is free, which causes additional *queuing delay*. The time from the beginning of the transmission until the last bit is pushed into the channel is called *transmission delay*, and can be calculated as packet size in bits divided by data rate of the next link. This work considers two types of channels: the GSL and the ISL with configurable constant data rate.

Propagation delay The time it takes for the bits on a transmission link to propagate from the sender to the receiver is the *propagation delay*, which is a function of distance and speed of the transmission channel, typically equal to the speed of light c. For ISLs, the propagation distance d is the distance between the communicating pair of satellites, while for GSL, d is the distance between the satellite and the GS. With that, the propagation delay can be calculated as d/c.

2.5 Discrete event simulation

Discrete Event Simulators (DESs) enable the simulation of systems in which one or more phenomena of interest change value or state at discrete points in time, rather than continuously [37]. Value and state changes are modeled as *events* associated with a specific execution time. This time must be in $[0, T]$, where T is the configured maximum simulation time, otherwise the event will not be executed. Events executed at time t_1 can create and schedule new future events for time t_2 with $t_2 \geq t_1$. Created events are inserted into a priority queue, with the lowest execution time having the highest priority. As depicted in fig. 8, at the beginning of the simulation run, the simulator loads and initializes the necessary state and schedules an initial event. Then, the simulator pulls and executes the next event in the priority queue. After the event has been executed, the simulation time t is set to the start of the next event in the priority queue, and the simulator collects desired statistics about the execution. If t exceeds T or the priority queue is empty, the stop condition is reached and the simulation ends with the return of collected statistics and results. Otherwise, the next event in the priority queue is pulled and executed.

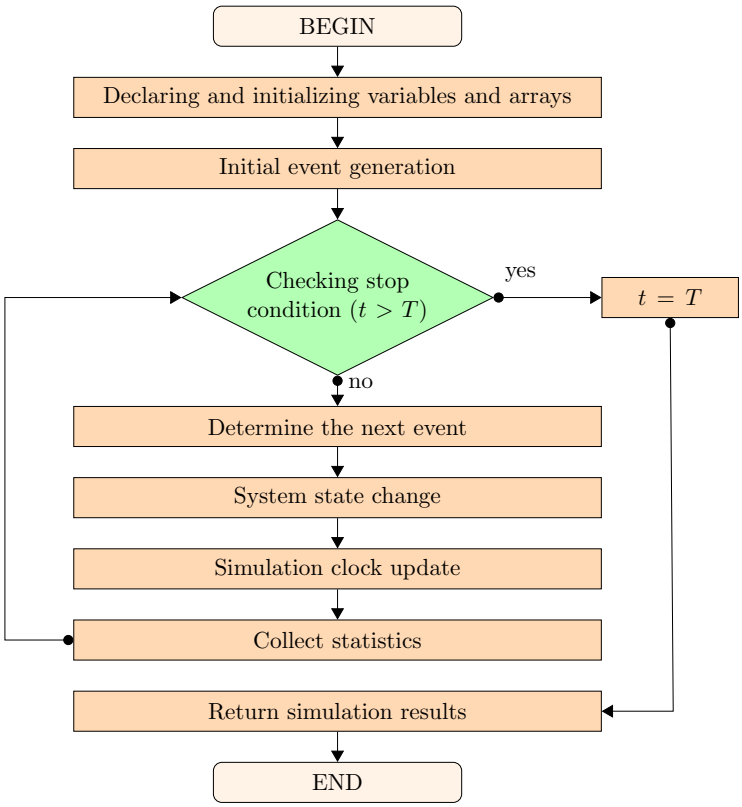


Figure 8: Basic architecture of discrete event simulators according to [38].

Components, such as network nodes, are modeled as simulation *modules* that store configuration and state. Each time an event is executed, it has a module as its target. Modules implement a callback with predefined arguments to which the event is passed as an argument. During the execution of the handler, the callback has access to the state of the modules and can create and schedule new events as described above. When a module creates a new event with itself as the target module, this is called a self-message.

2.5.1 Model network protocols

The state of networks typically changes at certain discrete points in time, such as the arrival of a transmitted packet, the expiration of a timer, or the end of a sleep. Thus, simple network protocols are well suited to be modeled as discrete systems. Network nodes are realized as modules, while packets, timer expirations, and sleeps are represented as events. Packet arrivals at nodes $n_1, ..., n_n$ occur at discrete times $t_1, ..., t_n$, so they can be modeled as events scheduled for simulation time $t_1, ..., t_n$ with target modules $n_1, ..., n_n$. Timers, sleeps and similar with duration d_{timer} that start during an event at time t can be modeled as new events executed at time $t + d_{time}$. This is essential for use cases such as periodic self-updates, waiting time or latency simulation.

2.5.2 Discretization of continuous events

Unfortunately, more complex networks are not only composed of discrete events, but may also contain continuous events such as the movement of a satellite on orbit. Continuous events cannot be modeled and scheduled inside a DES. First of all it is important to note that in most cases the function of time (e.g. the motion) itself is not of interest, but the state that can be derived from it. For example in the satellite case, one is interested in the satellites' current position, direction or orientation, but not necessary in the motion. There are two approaches on how to address this problem.
A simplistic approach is to compute the required state whenever it is needed. Although this is relatively simple to implement, it does not scale well. For instance, running DSPA on a constellation with thousands of satellites could become very slow due to the number of calculations required to determine satellite positions for calculating the distance.
The more complex to implement but better-scaling approach is to transform continuous events into a sequence of periodic discrete events using sampling. To do this, the continuous function of time is sampled at constant intervals (sampling rate). Whenever the continuous function is sampled, the affected module stores the result inside its state. However, this method is not error-free because the state between two consecutive samples is constant and no longer continuously changing. Since the duration between two successive samples depends on the sampling rate, the severity of the error also depends on the sampling rate. Thus, it is essential to select a sufficient small sampling rate to keep the error small or close to 0.

3 Related Work

Work related to this thesis falls into three categories associated with the works minor research questions. First, work related to benchmarking LEO routing algorithms including constellations, traffic patterns, leveraged metrics and posed evaluation questions, second, work dealing with the creation of satellite network simulators, and lastly, proposals for LEO routing algorithms.

3.1 Benchmarking of LEO routing algorithms

Previous overviews of routing techniques in LEO such as [40, 41, 39] are merely a listing and description of algorithms that existed up to the time of publication and do not include any comparison of their performance. In addition, due to the age of the surveys, no special attention was paid to upcoming mega-constellations such as SpaceX's Starlink[4] or Amazon's Project Kuiper[5], which place higher demands on routing capabilities due to the thousands of satellites operated in the constellation and require huge scalability of the applied algorithms.

By referring back to the three main task areas of routing in sec. 2.4, namely path finding, congestion control and fault recovery, it can be said that the scope of identified LEO routing algorithms differs heavily among proposals. Some algorithms like [4] propose new ways to handle the task of path finding for example by leveraging a minimum hop count instead of minimal distance, while they do not provide any methodology for handling congestion or link failures. Other works like [5] combine already existing pathfinding techniques such as DSPA with a system to handle congestion and link failures by flooding information to neighboring nodes.

3.1.1 LEO Routing Algorithm Classification

In order to create an overview or taxonomy of algorithms, it is necessary to find general concepts that allow the classification of routing algorithms. For this purpose, the following gives a listing and explanation of these concepts:

- **Type:** The algorithm type is used to distinguish the algorithm concept. *Proactive* algorithms calculate the required data in advance and cannot react to sudden changes unless there are pre-calculated alternative routes. This means that the next hop for each destination is already determined when a packet arrives, e.g., by pre-calculated routing tables. *Reactive* algorithms, on the other hand, are able to compute routes on demand, which often but not necessarily includes the ability to respond to sudden changes such as failures or congestion.
- **Network Model:** Most algorithms can be divided into space virtualization algorithms (*virtual node*) and time virtualization algorithms (*virtual grid*). In *virtual node* algorithms split the covered area into regions, in which fixed addresses and routing tables are assigned to the closest satellites, while *virtual grid* algorithms make use of the periodicity of satellites and divide the orbital period into discrete time slices or snapshots [5, 42]. Some algorithms like [26] use other techniques such

[4] https://www.starlink.com/technology
[5] https://www.aboutamazon.com/what-we-do/devices-services/project-kuiper

as software-defined networks, and therefore do not belong to these categories.

- **Optimization Metrics:** The optimization metric is the metric that the algorithm attempts to minimize or maximize. For minimization, this could be to achieve minimum hops or distance packets must travel, while for maximization it could be throughput, bandwidth, or fairness.

- **Optimization Method:** The method describes how the optimization is attempted. To achieve minimum distances, shortest path algorithms could be used, while to minimize congestion, the exchange of status messages between neighboring satellites could be used to share congestion status.

- **Path Computation Model:** For the path computation model it can be distinguished between *connection-oriented* and *connection-less* algorithms. An algorithm is *connection-oriented* if the path of the packet is calculated once and it will strictly follow this path until the destination is reached. *Connection-less* algorithms, on the other hand, forward a packet at each satellite regardless of previous routing decisions, which does not necessarily require recomputation in the presence of routing tables.

- **Decision Making:** For decision making, it can be distinguished between three categories. Decision making is *centralized* if there is a central authority, e.g., a controller satellite, which is consulted by other satellites for routing decisions, or if the routing is stateless and each satellite would make the same routing decision for a given source-destination pair. A system is *decentralized* if nodes make their own routing decisions, which could be based on gathered runtime data such as recognized failures or congested satellites. Futhermore, if a combination of *centralized* and *decentralized* routing is used, the decision making can be categorized as *hybrid*. For example [34] combines static inter-zone routing with controller based intra-zone routing.

3.1.2 Constellation Variants

During the literature review on proposals, surveys and reviews for LEO routing algorithms several constellation variants used for benchmarking were identified. We distinguish between findings for Walker-Star and Walker-Delta.

Walker-Star We found four different Walker-Star constellation groups, two of which occur several times in modified form, which are listed below:

- **Iridium**[6]: This is the most frequently used constellation for benchmark simulation in the reviewed papers, more precisely in [46, 47, 26, 34, 13, 43, 44, 5, 45, 29, 30]. Here, the Walker parameters for the constellation are slightly varying in terms of inclination (86° vs. 86.4°), altitude (780km vs. 1000km) and inter-plane spacing (0 vs. 2). A minimum elevation angle of 10° for GSLs is specified in [46], but in no other referenced paper. Using the found FCC filing, we were able to determine the following Walker parameters except for the inter-plane distance of 86°: 66/6/- at an altitude of 780km.

[6] https://fcc.report/IBFS/SAT-AMD-19920808-00021/1162673.pdf

- **OneWeb**[7,8]: We found two mentions of the OneWeb constellation in [46] and [48]. Their Walker parameters are different with 86.4°: 1000/25/10 at an altitude of 1000 in contrast to 87.9°: 648/18/- at an altitude of 1200. According to the authors of the first variant, their configuration is only roughly comparable to OneWeb, as it is supposed to be an up-scaled version of Iridium. The second variant, on the other hand, attempts to reproduce the original configuration. A value for the inter-plane spacing is only given in the first but not in the second variant. With respect to minimum elevation, the first variant provides a minimum elevation angle of 50, while the second does not. Both variants do not coincide with the FTC fillings we found for the first shell of OneWeb, which suggest 87.9°: 588/12/- in phase 1 and 87.9°: 1764/36/- in phase 2.
- **Star-128** The third constellation found in [36], increases the inclination to 89° with the Walker parameters of 89°: 128/8/3 at an altitude of $1050km$, which is comparable to Iridium but nearly doubles the satellites while only adding two more planes. Because of that, the number of satellites per plane increases from 11 to 16, which reduces the angle between satellites on the same plane from 32.7° to 22.5° and the angle between planes from 30° to 22.5°, which leads to a denser grid while maintaining similar characteristics to Iridium.
- **Star-288** For the last constellation found in [24] and [49], the authors did not specify an inclination, but since they do consider polar regions during their work, we classify the utilized constellation as Walker-Star. The number of satellites is more than doubled again in comparison to *Star-128* with four extra planes, while the altitudes varies between 760km (calculated from orbital period of 100 min) and 1375km. This is the highest altitude found for Walker-Star.

Walker-Delta By grouping the constellation findings for Walker-Delta by inclination, we end up with three different groups.

- **Odyssey** In [50] a small constellation was used, consisting of 12 satellites distributed on 3 planes. Although the authors refer to it as LEO, they give an orbital period of 6 hours, from which we have calculated an altitude of about 10390km. This clearly does not meet today's requirements for LEO, but for MEO. Therefore, we do not consider this finding.
- **Delta-48** The second constellation from [51] has the highest altitudes among our findings with an altitude of 1400km, which apparently serves to compensate for the small number of 48 satellites distributed over 8 planes with an minimum elevation of 10°.
- **Starlink**[9] Several Starlink-like constellations with an inclination of 53° at an altitude of $550km$, with varying number of satellites, plane count and inter-plane spacing were found in [4] and [52]. While the first work uses variants together with the original parameters 53°: 1584/72/39, the second one is limited to variations. In total, we found seven variants with equal inclination and altitude as Starlink, ranging from 288 satellites distributed over 24 planes to 32000 satellites distributed over 200 planes.

[7] https://fcc.report/IBFS/SAT-MPL-20200526-00062/2379565.pdf
[8] https://fcc.report/IBFS/SAT-MPL-20200526-00062/2379706.pdf
[9] https://fcc.report/IBFS/SAT-MOD-20181108-00083/1877671.pdf

Observations While studying LEO routing algorithms, it can be noticed that older papers often use an Iridium-like constellation, which was one of the first actually deployed LEO constellations. In addition, the used number of satellites in older papers is really small compared to constellations proposed in more actual papers, which use constellations with hundreds of satellites. This trend of increasing satellite numbers can be observed for Walker-Star and Walker-Delta.

3.1.3 Traffic Patterns

Regardless of the constellation type, we could identify four traffic pattern groups. The first traffic pattern is *burst+idle* used in [13, 43, 44, 45], which consists of two phases, the burst phase, in which many packets are sent in succession, and the idle phase, in which the traffic generator sleeps. In our findings, the burst duration is 200ms or 300ms long, while the sleep phase is either 300ms or 900ms long. The other three groups are more similar and model traffic according to distributions, namely the Poisson distribution [5, 52], a distribution according to world population (or rather their traffic requirements) [49, 29, 50, 51], or according to a Pareto distribution in combination with a *burst+idle* pattern of 200ms [45]. To select traffic sources and destinations, [46, 13, 4, 52] generate a random sample of source and destination pairs, although Roth, Brandt, and Bischl state that this is not realistic, since most traffic starts and ends in the same country. Other papers, such as [45, 49], do not specify how the communication pairs are chosen.
Another property which varies among findings is the utilized packet size, with 1000 bit being the most common [26, 13, 43, 45], but there are also outliers, e.g., 100 kbit in [46] or 8000 bit in [49].

3.1.4 Simulation Parameters

Among the reviewed papers, benchmark scenarios are run with different simulation parameters. The first category of parameters is the maximum simulation time, which varies widely among our findings, with 60s as the shortest simulation time in [26, 45], up to half a day in [46] and a full day in [49]. In the middle of that interval we found simulations with multiple hours as simulation time, such as 6050s in [44] and 7200s in [5]. A more special simulation time was given in [51], where the simulation ends after one orbit period of the satellites. Another property is the data rate of the ISLs with 5.12 Mbit as the smallest finding in [5], 10 Mbit in [44], 25 Mbit as the most common value found in [26, 34, 13, 43, 45], and the highest values 100 Mbit in [46] and 160 Mbit in [49]. Similarly, for the data rate of the GSLs, $4.8Kbit$ in [5], 15 Mbit in [44] and 25 Mbit in [45] were found.

3.1.5 Benchmarks and Evaluated Metrics

Most works attempt to benchmark the performance of their proposal by using one or more well-known algorithm as a baseline. For [4, 13, 24, 34, 5], this is either DSPA or the Bellman shortest path algorithm. This algorithms are roughly identical, while the Bellman algorithm is slower but also allows for edges with negative weights. In addition, the papers are comparing different routing topologies and baseline scenarios, which complicates the comparison of the proposed algorithms because some scenarios include dynamic events such as link failures while others not. It can be observed that most findings set their focus on outperforming predecessor algorithms, thus provide ways to increase throughput or reduce the E2E delay or RTT.

The metrics collected and used for comparison by the algorithm proposals include:

- **Throughput**: the number of delivered kilobytes per second, that can be calculated by the number of packets that are delivered by the constellation per second together with the packet size, found in [43, 13, 5, 45].
- **Runtime**: the time for computing a route between a random pair of satellites in constellations of different sizes, found in [4].
- **Deviation**: the deviation of the length of generated routes in comparison to the shortest possible route, found in [4, 24].
- **Packet loss**: the rate of dropped packets, which is the result of dividing the number of dropped packets by the sum of dropped and delivered packets, found in [47, 26, 43, 13, 5, 49, 45, 29, 52].
- **Queuing Delay**: the average time packets are enqueued on satellites, found in [26, 52].
- **Hop count**: the number of hops a packet has required to reach the destination, found in [5].
- **E2E Delay**: the sum of all delays for a packet to travel from source to destination, found in [46, 47, 26, 43, 13, 44, 5, 45, 29, 52].

Some papers also evaluated jitter [51] and call blocking [50], which is related to voice transmission.

Route variability Another metrics which is not directly connected to performance is discussed in [31], which is the route churn resulting from the frequency of rerouting. Bhosale et al. focus on the fact that most routes are used for less than half of their lifetime, with some routes being used for only a few seconds, and that rerouting results in little RTT reduction in half of the cases. This is due to the fact that one routing algorithm may choose the shortest path with a low remaining lifetime, while there is another route that is not necessarily much longer but remains valid for a longer time, which requires a trade-off between route churn and optimal latency. High churn, usually caused by shortest path algorithms, can come together with poor path utilization and poor performance of congestion control mechanisms.

3.2 Satellite Network Simulators

According to Breslau et al. network simulation is widely used to test protocols for requirements such as security, quality of service, and policy management. It can be differentiated between custom simulators tailored for one specific protocol and common simulators which are able to support multiple protocols. Custom simulators have certain drawbacks because they are expensive to build, inflexible, and unable to handle a variety of network phenomena such as interference. On the other hand, common simulators are designed to be flexible, support the addition of new protocols and allow for an easier comparison of results across research efforts [53].

NS3 [54] and *omnetpp* [8] are open-source simulators that belong to the group of common simulators. Both work on the principle of discrete event simulation and are widely used in the field of network simulation. While NS3 was developed with a focus on net-

works, *omnetpp* is designed for general simulation use cases. Nevertheless, *omnetpp* can be extended with the *inet framework*[10], which provides necessary functionality for network simulation, such as periodic updating modules, implementations of well-known protocols like UDP/TCP, and support for basic node mobility. In addition, unlike NS3, *omnetpp* comes with a rich graphical user interface that allows for visual validation of constellation topologies and traffic exchanges. There are a number of space-focused network simulators, such as the Satellite Network Simulator (SNS3) [55] and SCNE [56], which are based on NS3. On the other hand, there are *omnetpp*-based frameworks such as OS^3 [11] for basic celestial mechanics and leosatellites[12] that builds around OS^3 and provides mobility models suitable for use with the *inet framework*.

Commissioned by the European Space Agency (ESA), SNS3 models a satellite network focused on the physical layer, with a single GEO satellite covering Europe. Due to the consideration of a single satellite, important features for researching CRAs such as ISLs or the support for many moving satellites are not part of the simulator, making it unsuitable for the needs of this work.

SCNE, which is also developed by the ESA, has comparable research efforts to this work in that it aims to provide a methodology for studying and evaluating the performance of routing algorithms for LEO constellations. The ESA wants to provide a solution that is very flexible and allows the addition of different routing protocols. Unfortunately, the simulator has no support for the creation or propagation of satellite constellations but instead relies on the external commercial software STK[13], which is a proprietary software that requires a license, and thus complicates and limits the application possibilities.

In [57] Liu et al. describe the design and evaluation of LSNS, a large scale satellite network simulator based on *ONE simulator* [58]. The simulator comes with a module for orbital calculation, and an interface for visual inspection of real-time simulation results. In addition, the simulator has been designed in terms of performance to be suitable for simulating upcoming mega-constellations with thousands of simulated satellites. The architecture consists of four major building blocks, a module for event and traffic generation, a satellite motion module, the transmission module for link establishment and finally the routing module for switching packets. Although parts of the simulator are published on public sources, the development is sponsored and some parts are only available internally. Nevertheless, the architecture is promising and a good influencing factor for this work.

Other simulators such as [59, 58] either do not implement the necessary functionality for orbit determination or focus on delay-tolerant networks, which is outside the scope of this work.

FLoRaSat builds on top of OS^3, *leosatellites* and the *inet framework* and was originally developed for the study of Lora satellite networks. Although Lora satellite networks are not the subject of this work, the foundations provided by this framework are suitable for developing a methodology for simulating routing protocols in satellite constellations. The

[10] https://inet.omnetpp.org/
[11] https://github.com/inet-framework/os3
[12] https://github.com/avian688/leosatellites
[13] https://www.ansys.com/products/missions/ansys-stk

fundamentals include mobility models for ground stations and satellites with support for celestial mechanics like orbit determination, and a basic on-board packet processor structure that can be used as a starting point. In contrast, it provide no support for dynamic creation of satellite constellations, no handling of dynamic creation and tear down of ISLs according to given constraints, and no interface that can be used by routing modules to extend the platform. Since FLoRaSat currently has only a rudimentary "route top-right" approach and lacks a proper implementation of satellite routing protocols, extending the framework could also benefit the research efforts of the original FLoRaSat authors as a side effect of this thesis.

3.3 LEO Routing Algorithms

This section describes two proposed LEO routing algorithms that will be added to the simulator as routing modules and used for subsequent analysis along with the simplistic, already discussed, Directed Routing approach.

3.3.1 Dynamic Detection Routing Algorithm

DDRA [5] was first published in late 2014 by Tan and Zhu. The algorithm uses virtual topology snapshots, i.e., the satellite period is divided into time slices in which the constellation routing topology does not change. This means that the state of each ISL remains unchanged under normal conditions during a time slice. During the transition between two time slices, ISLs may change state, which means that a new topology snapshot must be loaded. These changes are classified as *predictable changes*, while changes originating from congestion or link failures are classified as *sudden changes*. The duration Δt of a time slice is a tradeoff, as smaller time slices lead to less variation in link costs, but also to higher memory requirements. DDRA defines the topology during the k'th time slice as $G(k)$ and the link cost at time t between node i and j as $c^k_{i,j}(t)$. The graph consists of edges, described as $E_{i,j}$ to represent an edge between the satellites i and j Furthermore, the algorithm imposes the following conditions on the length Δt for each time slice k:

- $G(k)$ reflects the actual topology of the constellation during the whole time slice,
- for any $t \in [k * \Delta T, (k+1) * \Delta T]$, it holds that $(c^k_{i,j}(t) - c^k_{i,j}(t))/c^k_{i,j}(t) << 1$,

which results in a valid representation of the constellation during that time slice, and that the link costs are either less than they were at the beginning of the time slice, or have increased, but are still less than twice as much.

Pathfinding DDRA does not propose a new method for calculating a route, but uses the well-known and researched DSPA, introduced in 2.3.4. DSPA is used in such a way that at the beginning of a new time slice k the current routing topology $G(k)$ is loaded and the shortest path to each satellite is computed and stored in a routing table (this could potentially be done before deployment). This does not guarantee that each packet will take the shortest path, but it can save a lot of processing power since DSPA does not have to be run per packet, only per time slice. The amount of deviation between the shortest path and the actual path taken can be controlled by the length of a time slice.

Congestion Control Tan and Zhu proposed a way to balance the congestion in the constellation by attempting to control the queue sizes of the satellites. To do this, each

satellite periodically monitors the number of packets in its packet subqueues. If this number exceeds a certain threshold $N1$, the satellite considers itself to be congested. In the paper, N_1 is calculated as follows:

$$N_1 \leq N_0 = \frac{T_{MAX}}{M} * \frac{V_{MAX}}{L_{MIN}}$$

where T_{MAX} is the maximal tolerable delay for one-way connections typical equal to $500ms$ which the authors obtained from ITU regulation. M is the average number of hops in the constellation, V_{MAX} is the maximum link transmission data rate and L_{MIN} is the minimum packet length. When a transmission subqueue of satellite i becomes congested, the satellite deletes the hop to satellite j represented by the edge $E_{i,j}$ from $G(k)$. The satellite remembers this state until the associated subqueue is empty. Then, the previously deleted edge $E_{i,j}$ is added back to $G(k)$. Furthermore, the satellite will inform its neighbors about congested edges and when they become empty again.

Fault Recovery To detect link failures, DDRA uses a periodic feedback mechanism to decide if a link is usable or has failed. For this, satellite i counts missing acknowledgements of sent packets to a destination satellite j. If the count for any destination reaches three, the link is considered to have failed and the edge $E_{i,j}$ is removed from $G(k)$. After that, the satellite will use DSPA to recompute the routing table with the adjusted connection matrix. The satellite remembers the failed edge until the end of the time slice and reapplies it as other errors are detected during the same time slice.

Algorithm Taxonomy DDRA is a *reactive* algorithm, as it combines static offline-generated routing topologies, valid in the corresponding time slice, with dynamic collected runtime data to react to both growing congestion and link failures. Since it relies on topology snapshots, the network model of this algorithm can be classified as *time virtual topology*. We would classify the optimization metrics of this algorithm to be *queueing delay* and *packet loss*. The path computation model is *connection less*, as packets are routed at each satellite hop regardless of routing decisions of previous hops; otherwise, the adaptive mechanisms would be inefficient because only neighboring satellites know about failures and congestion. Each satellite makes routing decisions *decentralized* without consulting other instances. Although the algorithm has been tested with the Iridium constellation (Walker-Star), there are no apparent restrictions preventing its possible use with Walker-Delta constellations due to the general applicability of DSPA.

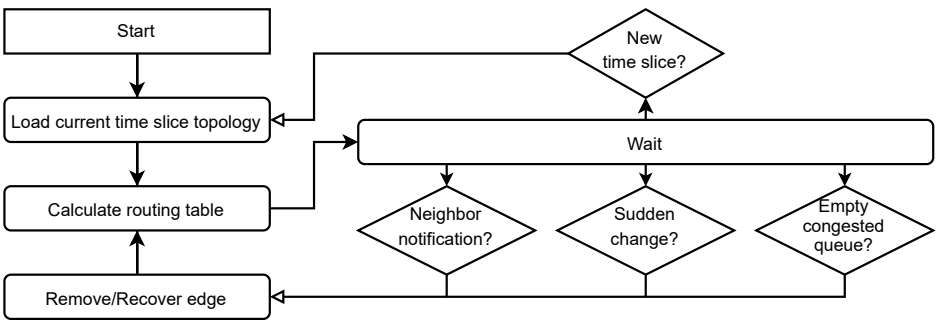


Figure 9: Flowchart of DDRA operation loop.

3.3.2 DISCO Route

DISCO [4] was first published in 2022 by Stock, Fraire, and Hermanns and sets its focus on Walker-Delta mega constellations, comprising hundreds and thousands of satellites. The algorithm takes advantage of the fact that orbital parameters of other satellites are deterministic or constant (in absence of acting forces) and therefore can be known to other satellites, either stored prior to deployment, updated via ground stations or computed. In addition, constellation properties such as *RAAN difference* $\Delta\Omega$, *phase difference* $\Delta\Phi$ and *phase offset* Δf are constant.

Pathfinding DISCO utilizes the orbital elements of satellites to calculate the minimum number of ISL hops between a satellite pair without providing a methodology for selecting adequate first and last satellites. Calculation of the minimum hop count is split into three parts, the minimum inter-plane hops, the minimum intra-plane hops and the minimum hops.

- **Min. inter-plane hops** H_h is given by the difference of the longitudes of the respective ascending nodes $\Delta L_0 = (L_{0,2} - L_{0,1}) \mod 2\pi$, which is the RAAN that must be covered in east direction. So the horizontal minimum hops for west and east direction can be calculated ($\lfloor x \rceil$ indicates commercial rounding) as follows:
  $H_h^{\leftarrow} = \left\lfloor \frac{2\pi - \Delta L_0}{\Delta\Omega} \right\rceil$ and $H_h^{\rightarrow} = \left\lfloor \frac{\Delta L_0}{\Delta\Omega} \right\rceil$

- **Min. intra-plane hops** H_v is given by the difference of the phasing angle of two satellites. Since the difference that must be covered by intra-plane hops is affected by H_h, and each inter-plane hop to the east increases and to the west decreases the phase angle of the current satellite by Δf, it is necessary to distinguish between two directions:
  $$\Delta\overrightarrow{u} = (u_2 - u_1 - H_h^{\rightarrow} * \Delta f) \mod 2\pi$$
  $$\Delta\overleftarrow{u} = (u_2 - u_1 + H_h^{\leftarrow} * \Delta f) \mod 2\pi$$
  where u_2 and u_1 are the argument of latitude of the satellites. With that it is possible to calculate H_v as:
  $$H_v^{\nwarrow} = \left\lfloor \frac{\Delta\overleftarrow{u}}{\Delta\Phi} \right\rceil \qquad H_v^{\nearrow} = \left\lfloor \frac{\Delta\overrightarrow{u}}{\Delta\Phi} \right\rceil$$
  $$H_v^{\swarrow} = \left\lfloor \frac{2\pi - \Delta\overleftarrow{u}}{\Delta\Phi} \right\rceil \qquad H_v^{\searrow} = \left\lfloor \frac{2\pi - \Delta\overrightarrow{u}}{\Delta\Phi} \right\rceil$$

- **Minimum hops** H_{min} can be calculated with the help of H_v and H_h as:
  $$H_{min} = min(H_h^{\leftarrow} + H_v^{\nwarrow}, H_h^{\leftarrow} + H_v^{\swarrow}, H_h^{\rightarrow} + H_v^{\nearrow}, H_h^{\rightarrow} + H_v^{\searrow})$$
  In addition, the two input variables whose sum results in the minimum hop count indicate the direction of travel.

DISCO utilizes the computed values of H_v and H_h with the minimal sum. The general idea of the algorithm is to perform inter-plane hops as close as possible to the poles, since the distance between pairs of satellites is smaller here than at the equator. The

algorithm distinguishes two cases: *A2A/D2D* where both satellites are ascending or both are descending, and *A2D/D2A* where one satellite is ascending and the other is descending or vice versa. A summary of both cases is given below, but more implementation details can be found in the authors' original paper.

- **A2A/D2D**: Intra-plane hops are distribute in the middle of the route and inter-plane hops at the beginning and end. To determine the satellite pairs for inter-plane hops, those with the highest absolute value of the latitude sum are selected.
- **A2D/D2A** Here, the route must pass through the polar region, because jumping from an ascending to a descending node (or vice versa) is only possible there. The hops are distributed in such a way that the intra-plan hops are performed at the beginning and end of the route, while inter-plane hops are in the middle of the route as close to the poles as possible. In contrast to **A2A/D2D**, those satellite pairs that have the minimum absolute value of the latitude sum are selected for the inter-plane hops.

Congestion Control/Fault recovery In contrast to DDRA, DISCO is limited to the proposal of a new pathfinding methodology. Nevertheless the algorithm has a much smaller processing time than DSPA based algorithms, which can, depending on how DSPA is implemented, contribute to less congestion due to faster clearing of the processing queue. In particular, the authors claim that DISCO has much better scalability than DSPA, since quadrupling the satellites increases the runtime by a factor of about 1.3x as opposed to about 4.3x for DSPA, while even having a shorter base level.

Algorithm Taxonomy DISCO is a *proactive* algorithm, since it uses known orbital parameters of the satellites and the constellation to compute routes. The network model of this algorithm is difficult to classify because there is no use of nodes temporary mapped to physical addresses or usage of time slices. Instead satellites are mapped by their orbital parameters, which is somehow similar to the concept of *virtual node*. The optimization metrics of this algorithm are *minimal hops*, *short distance* and *scalability*, which the authors try to achieve by utilizing an efficient calculation methodology based on minimal hops and selecting satellite pairs with minimal expected distance. The path computation model is *connection-oriented*, as the path will be decided once at the first satellite and not change afterwards. Routing decisions are *centralized*, because no matter which satellite would perform the routing, the route should always be the same for the given satellite pair as they are made based only on pre-planned properties known to each satellite. This algorithm only works in Walker-Delta constellations due to the fact that the algorithm does not consider the two seams and tries to optimize the route over an overlapping ascending and descending mesh, which does not exist in Walker-Star.

4 Benchmark scenarios

The next subsections propose multiple benchmark scenarios that allow the exploration of important features and characteristics of arbitrary routing algorithms. They consist of expressive scenario building blocks, including harvested constellation variants, traffic patterns and simulation parameters listed in sec. 3.1, and are supplemented by a broader set of building blocks added by this work. For this purpose, we have divided the benchmarking of routing algorithms into four categories that are linked with benchmark scenarios, utilized metrics, and evaluation questions to ensure a comprehensive study of algorithms. The forseen categories are:

1. **Parameter study:** Testing of the reaction and performance of an algorithm at different altitudes, inclinations, and data rates helps to understand the influence of constellation parameters and can be used for constellation optimization before deployment.
2. **Regular operation:** Using one small and one large constellation per Walker type, it is possible to investigate the day-to-day performance of a constellation that uses the tested algorithms. This enables a comparison of the system performance without failures or congestion.
3. **Failure reaction:** By utilizing the same base constellations as above it can be investigated, how the performance is influenced by:
   (a) an increasing ISL module failure probability,
   (b) an increasing node failure probability,
   (c) recovering ISL modules and nodes.
4. **Congestion reaction:** By utilizing traffic burst patterns on the identical base constellations as before, this benchmark explores the congestion handling capabilities of the algorithms for various levels of burst severity.
5. **Algorithm Efforts:** Using collected data and one-to-one comparisons, how do the algorithms differ in terms of:
   (a) required computational effort for different fleet sizes,
   (b) memory utilization, i.e., what needs to be stored on the satellites,
   (c) computational overhead generated by protocol negotiation for path finding, congestion control and fault recovery.

The following subsections discuss, one category at a time, the necessity for and contribution to the characteristic exploration and introduce the envisioned associated benchmark scenarios.

4.1 Parameter Study

A parameter study is the subsequent execution of simulations with a single varying parameter to observe how extracted results change under different circumstances. For the research of LEO routing algorithms, this could involve changes to the constellations Walker parameters or ISLs and GSLs link data rates. The purpose of this evaluation question is to identify if there are situations where it is inappropriate to use certain algorithms. For

example, if algorithm A requires high data rates for good performance, it is not advisable to apply it to a constellation that cannot support those rates, if there is another algorithm B that provides better performance at low data rates. Another use case is when an algorithm has already been selected to determine optimal Walker parameters, such as an ideal inclination angle, to improve the overall performance. Since these values can only be changed with difficulty or not at all afterwards and then cause high costs, they must be defined before operation. Furthermore, this method can identify the optimal relationship between constellation size and throughput to keep deployment costs low without compromising system performance.

Evaluation Questions The aforementioned use cases help to answer evaluation questions about the general performance of an algorithm in a variable constellations. It also allows conclusions to be drawn about the economic viability of the system as a whole. Based on the aforementioned points, this work proposes a parameter study as a good initial evaluation approach for LEO CRAs, which helps to answer the following evaluation questions.

1. How does the altitude affect system performance in terms of measured delays and route length?
2. How does the inclination affect system performance?
3. Does higher ISLs and GSLs data rates contribute to less delay and congestion?

Scenario For this purpose, two parameter studies will be prepared, one for Walker-Delta and one for Walker-Star. The two studies will use a small constellation variant taken from the results of the related work study as a base and evolve from there in terms of Walker parameters and system parameters. Walker-Star will use the Iridium constellation as a base and gradually evolve to the parameters of the OneWeb constellation in terms of number of satellites, planes and inter-plane spacing. For Walker-Delta, this work will stick with to Starlink-Like constellation findings, starting with 288 satellites and growing to up to 1584 satellites, which is equal to the first shell of the Starlink constellation.

Constellation Params:	Walker-Star	Both	Walker-Delta
Satellite count	66, 500, 1000		288, 600, 1584
Plane count	6, 10, 20		24, 20, 72
Inter-Plane Spacing	2, 5, 10		1, 10, 39
Groundstation count	10 (Incl. Polar)		10
Variable params:			
Altitude[km]	780, 1000, 1200		550, 700, 1000
Inclination	81°, 86.4°, 89°		53°, 60°, 66°
ISL + GSL data rate[Mbps]		25, 50, 100	

Table 3: Variable parameters for use during parameter study.

Tab. 3 shows the setup of the parameter study and the varying values that will be consider. The header of the table shows three base constellations and the number of ground stations used, followed by the values tested in combination with each of these bases. Since each combination must be studied to provide a comprehensive comparison of a parameter's impact on system performance, a large number of scenarios is created. The number of combinations can be calculated by multiplying the number of base constellation com-

binations (3) by the number of entries in each row designated for this type, yielding 81 combinations per Walker type. By using two different algorithms per type and two random seeds to minimize errors, this benchmark will end up with 324 simulations per type. In theory, additional parameters could be investigated, including packet size or maximum queue size. However, adding another parameter would at least double the simulation count and exceed the time frame of this work.

Traffic For large traffic and constellations, the simulation speed can drop below one simulated second per second. Therefore, this work proposes to employ a *burst+idle* pattern for traffic, as shown in fig. 10, which allows the exploration of different satellite topologies, since idle times can be simulated quickly, and at the beginning of the next burst, the topology has evolved due to satellite motion and earth rotation. In addition, this allows for longer simulation times.

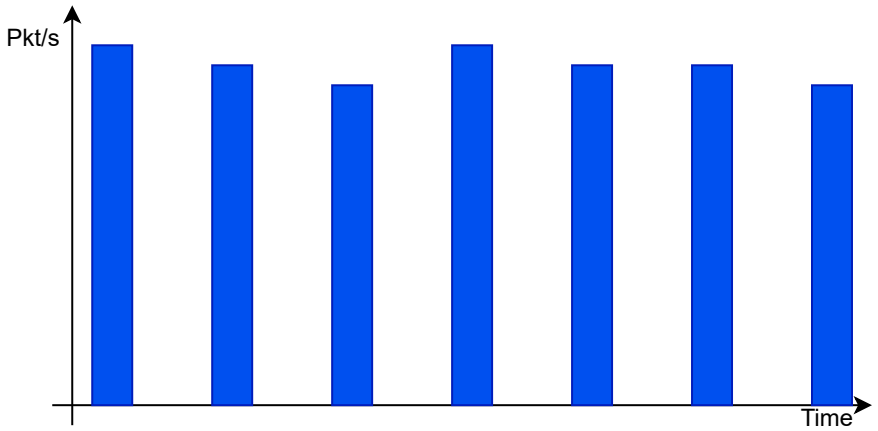


Figure 10: Burt+Idle pattern for topology exploration.

4.2 Regular Operation

This benchmark aspires an evaluation of the daily performance of the algorithm. To this end, several ground stations are distributed around the globe according to an almost realistic distribution with a slight surplus of ground stations in Europe and North America. In the following, the day-to-day performance without failures and no attempt of the benchmark to grow congestion is referred to as the the regular operation. In addition, this scenario attempts to test the system's ability to handle slowly increasing user traffic. Another important performance characteristic is also whether the constellation can still meet the maximum permissible delay regulations of 500ms. This benchmark can be used to answer the following evaluation questions:

1. How do the generated routes differ?
By comparing the variations in terms of distance traveled and number of hops from source to destination between the routes, it is possible to sketch a first comparison of the path finding performance and to see the differences in the computed routes in both small and large constellations. Using a Cumulative Distribution Function (CDF), it is also possible to determine the number packets that could be delivered within certain distances and

number of hops. Moreover, a CDF that requires a high number of jumps or a large distance to attain 1.0 indicates the presence of outliers.

2. Are there noticeable problems with path finding?
The packet loss and throughput statistics, along with the hop counts and distances previously studied, can be used to investigate whether the algorithm is failing to deliver packets. During normal operations, the percentage of undelivered packets should be zero, as there are no failures and the traffic generated by the traffic pattern should be not high enough to overflow the first satellites. In addition, problems that are related to the used constellation can be seen, e.g. if an algorithm struggles with small or high number of satellites.

3. How do the enabled packet delays differ?
If the packet loss rate is low or even zero, the next critical metric for good system performance and user experience is the E2E delay, which is the time it takes for packets to travel from source to destination. High average E2E delays lead to high average RTT since $\overline{\text{RTT}} \approx 2 * \overline{\text{E2E}}$. This can be explored by using a CDF of E2E delays to examine the delay distribution and how many packets can be delivered with which delay. In addition, box plots for the delay subtypes (see 2.4.2) can show how much they contribute to the total delay. Usage of percentiles to get the limit and average time to deliver 99% and 99.9% packets is a good metric to see quality differences between the constellation sizes and algorithms.

Scenario To test the regular operation of the algorithms, we propose the use of a large and a small constellation for each Walker type. This makes it possible to differentiate performance in constellations of different size. For Walker-Star the first smaller choice is obviously the Iridium constellation with 86.4°: 66/6/2/780, which has been used most often in past research, and the second larger one is an OneWeb-like constellation (equipped with ISLs) with 89°: 1000/20/10/1000. Further, a small variant with the same inclination and altitude as Starlink is chosen for Walker Delta, i.e. 53°: 288/24/1/550, along the original first shell of Starlink with 53°: 1584/72/39/550 as the larger variant.

Traffic To simulate the desired small growing traffic, the pattern shown in fig. 11 is implemented. Here, the traffic pattern is divided into repeating blocks that are separated by small sleep periods. This small sleep time is used to allow the constellation topology to evolve and the satellites to clear queues and complete the delivery of any remaining packets. During each repetitive block, traffic increases slowly over time.

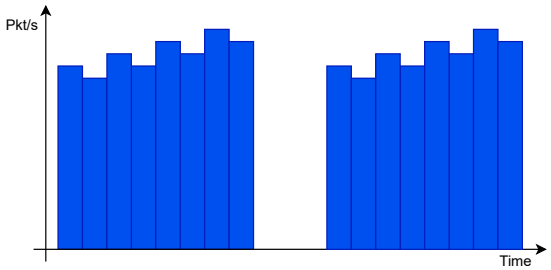


Figure 11: Repeating slow growing traffic pattern.

Figure 12: Utilized worldwide ground station positions.

4.3 Failure Response

Another important event to consider for benchmarks are the link and node failures introduced in sec. 2.4, from which satellite constellations are not spared. Because of this, the response of the constellation to such failures is an interesting property for comparing LEO routing algorithms. This criterion also helps to evaluate the cost of operation, which is an important factor in the economics of a constellation, because if the system is unable to respond to such failures, performance degrades and can only be restored by satellite replacement or manual intervention by overwriting the satellite software, both of which can be expensive. By examining the characteristics of fault recovery, it is possible to answer the following evaluation questions:

1. How capable are the algorithms in handling failures?
Algorithms with and without different failure handling mechanisms are affected differently by link and node failures. By comparing the packet loss rate for each constellation, algorithm and failure severity, it is possible to see the ability to still deliver packets and how much the implemented mechanisms can help, and if there are differences between the constellation sizes.

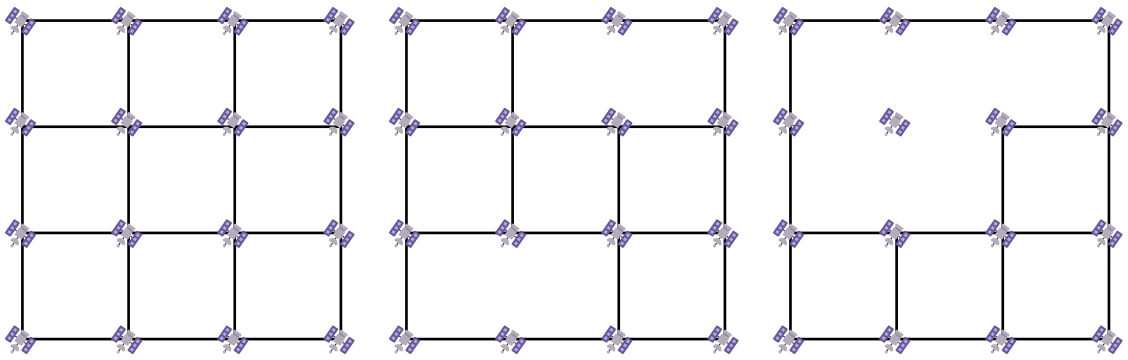
2. Are the algorithms able to recognize recoveries?
Failures can recover, and if an algorithm is unable to detect recoveries, the system performance will be permanently degraded by the ongoing failure recovery mechanism. Either by excluding certain links from the routing topology, resulting in unnecessarily long routes, or by causing a node to believe that no route exists, resulting in a subsequent packet drop. This evaluation question can be addressed by examining the change in packet loss after all failures have been resolved from a scenario in which failures were previously introduced.

3. Does the failure handling enable a still acceptable system performance?
This last evaluation question attempts to answer whether the capabilities are sufficient to

mitigate or reduce packet loss in the presence of failures to a level that is still acceptable and would allow an almost regular operation of the system. Any reduction in packet loss and associated increase in system performance is desirable, but a reduction from, say, 20% to 15% remains on an unacceptable level.

Scenario This scenario can be built using the same constellations configurations as described in sec. 4.2. The scenario is divided into several phases of equal length, with each phase representing the failure states of the constellation, i.e. which link and node failures are currently present in the period from the start of the current phase to the start of the next phase. During the transition from one phase to the next, each functioning link and node has a probability p_{LF} and p_{NF}, respectively, of switching to the failed state. Similarly, each previously failed link and node has a probability p_{LR} resp. p_{NR}, to be recovered at the beginning of the next phase. Before the first phase starts, there is a warm-up phase that allows dynamic algorithms to set up without the presence of failures.
To test different degrees of failure severity, three scenarios with 30 phases and a warm-up time of 100s are created for each of two seeds per failure probability level. This results in two low failure probability scenarios (0.25%), two medium failure probability scenarios (0.5%), and two high failure probability scenarios (1%) for links and nodes per algorithm and constellation. In addition, some failures should be repaired from time to time to test whether the algorithm is able to detect recovered failed links and nodes, so the recovery probability is set to 60% in each scenario. The last phase should be completed before the end of the simulation to test if the algorithm is able to return to regular operation after all failures have been resolved.

The visualization of an example sequence of evolving phases is shown in fig. 13, where the subfigure shows (a) phase 0, the original network, (b) phase 1, the network with two link failures added, and finally (c) phase 2, the network with one recovered link failure and one newly added node failure.



(a) Initial Network *(b) Applied Link failures* *(c) Node failure and recovered Link failure*

Figure 13: Sample visualization of failure response scenario.

Traffic Since the purpose of this scenario is to investigate the response to failures, continuous and regular non-bursty traffic that does not cause congestion is adequate. For this many globally distributed ground stations send little data traffic to random destination ground stations, so the traffic flows over many first and last satellite pairs. The reason for

using low traffic is that this benchmark attempts to achieve an isolated examination of the *fault recovery* capabilities, so no congestion control functions of the algorithm should be triggered that could interfere with the methods for handling link and node failures.

4.4 Congestion Response

The previous section sec. 4.3 proposed a benchmark for testing the *fault recovery* capabilities of routing algorithms. Continuing, this section focuses on the yet not investigated capability of routing algorithms, the *congestion control*. To this end, this work proposes a benchmark that leads to growing queues and requires the algorithm to negotiate for alternative paths to keep the queueing delay low and potentially prevent the occurrence of packet drops due to overflowing queues. Since the E2E delay consists of the queuing delay in addition to other delays, a low queuing delay contributes to a good user experience and system performance. To accomplish this, the benchmark answers the following evaluation questions based on the results of running the scenarios.

1. How capable are the algorithms in handling congestion?
Congestion handling is important to ensure future scalability of the user base and traffic. Heavy congestion results in growing queues and thus higher queuing delay, which leads to higher E2E delays. There are numerous methods that an algorithm can implement to handle congestion. By comparing the queuing delays of packets, the congestion handling capabilities can be visualized and the performance of different algorithms can be compared. In addition, it is possible to compare the varying impact of burst traffic on the algorithms in different sized constellations and whether an algorithm can benefit from a certain constellation size.

2. Which drawbacks are created by the mechanisms?
If an algorithm responds to congestion by, for example, rerouting traffic or excluding nodes from the topology, this can result in increasing delays or unroutable packets. The purpose of this evaluation question is to investigate whether the response will cause problems and degrade system performance without reducing congestion. For this, interesting metrics to investigate the side effects could be the observed hop counts, distances, and the packet loss, i.e., if smaller queuing delays lead to a massive increase in hops and distance, the overhead of the algorithm may be greater than the return. On the other hand, if the packet loss grows to unacceptable high rates, an investigation of routes generated by the algorithm is not necessary since the methodology can already be classified as being unsuitable.

Scenario Equivalent to the *fault recovery* benchmark multiple globally distributed ground stations are used to uniformly overload the constellation. In several zones, multiple ground stations are placed close to each other to increase the chance that a first satellite is the computed optimal first satellite for multiple ground stations, resulting in high incoming traffic via GSL on the satellite. Just as sec. 4.3 attempts to achieve an isolated benchmarking of the *fault recovery* capabilities, this benchmark strives for an isolated investigation of the *congestion control* methodology of the algorithm, so no link or node failures should be added to the scenario that could lead to active *fault recovery* mechanisms that have an impact on the *congestion control*.

Traffic An ideal traffic pattern for this benchmark is the *burst+idle* pattern, which can come in in various compositions such as 1s burst and 1s idle. The only condition is that the burst duration is long enough to trigger the congestion control behavior of the algorithm under test and that the effects are recordable for at least some time. In addition to the length of the burst duration, the packet sending frequency is also important, as it should be at approximately equal or greater than the sum of the transmission and processing delay, so that enqueuing of packets becomes necessary for a satellite. On the other hand, congestion can also occur without such a high sending frequency, but this requires multiple satellites to route traffic over the same satellite, so that traffic from multiple ISLs arrives on that satellite at the same time. This is an unreliable source of congestion, but the effect still amplifies the existing congestion.

4.5 Algorithm Efforts

The algorithm efforts evaluation focuses on evaluating more technical implementation details of algorithms, such as the runtime to find a path from a source to a target satellite. This evaluation differs from the previous benchmarks because the realization for this type of evaluation is more artificial and takes place in the "laboratory" rather than in a real scenario that must be simulated. The evaluation can be divided into three categories: computational effort, required memory capacity for storing algorithm data, and computational overhead generated by algorithm negotiation. The following subsections suggest methods for testing and evaluating the effort subcategories.

4.5.1 Computational Efforts

The computational effort is the runtime required to make routing decisions, which can either the entire path for *connection-oriented* algorithms or a single next hop for *connection-less* algorithms. For the computational effort, this work proposes the use of the methodology used in [4], where a constellation is chosen and the execution time is measured for a number of possible satellite pairs. This can be repeated for different sized constellations.

Evaluation Method First, five different constellation sizes are selected, and then the algorithm computes the route for each satellite pair while measuring the execution time. For each pair of satellites, the route is calculated 10 times, and then the average value is taken as the result of the execution time to eliminate measurement errors as much as possible. The routing implementations should be optimized as much as possible to provide a fair comparison. Also, the algorithms should be run in an isolated environment, not during an actual simulator run, to avoid noise from other running computations. For this purpose, the algorithm implementation added to the simulator can be reused or the code can be implemented using any fast system programming language.

Evaluation Questions Using the extracted results for the average path-finding runtime, it is then possible to classify the scalability and cost of path-finding for different algorithms and answer the following evaluation questions:

1. How expensive is the algorithm in terms of required computational power? Path lookup time is critical for algorithms that require either on-demand calculation of routes for packets, or algorithms that require recomputation of routing tables during

operation. More expensive algorithms can lead to increasing *processing delays* and longer *queuing delays*, as all arriving packets are queued during processing of another packet.

2. Does the algorithm scale well with an increasing number of satellites?
As the cost of launching satellites decreases, the number of satellites in proposed constellations continues to grow. It has even become feasible to expand constellations that are already in operation. The system responsible for routing must therefore be scalable and perform well even as the constellation grows.

3. Is the algorithm suitable for usage with LEO mega constellations?
Proposed future mega constellations consist of more than 10000 satellites. It needs to be tested whether an algorithm is scalable enough to perform route computations for this large number of nodes and beyond, in order to be ready for possible future implementations.

4.5.2 Memory Utilization

The memory usage category includes all data that must be stored on a satellite prior to operation, required for routing decisions, and all data that is generated during operation, required for dynamic *pathfinding*, *congestion control* or *fault recovery*. Unlike other evaluations, this evaluation requires a more manual review of the algorithm by checking implementation details and the algorithm's proposal for details about stored and loaded data.

Evaluation Method First, all locations in the implementation are identified where offline generated data, such as topologies associated with time slices, are loaded or stored. Then, these locations are checked and only unique locations are kept so that two different loads of the same data are not counted twice. Then an appropriate minimal representation is created that would allow the individual data to be stored, and the total size is calculated. This behavior is repeated for data that the algorithm generates and stores on its own, such as the identifiers of congested satellites or failed links and nodes.

Evaluation Questions After the calculation of the required memory requirements, it is possible to answer the following evaluation questions:

1. How expensive is the algorithm in terms of required storage space?
Memory is typically cheap for computers on Earth, but high memory requirements on satellites add new points of failure and can complicate the design. Therefore, memory requirements should be kept as low as possible. Different algorithms have different requirements, which are examined and compared here.

2. Is the required storage space influenced by parameters of the algorithm?
Some algorithms can be configured differently, e.g., the duration of the utilized time slices. The purpose of this evaluation question is to examine how the requirements change when the parameters are changed, and whether there are ways to optimize the requirements.

4.5.3 Algorithm Negotiation Computational Overhead

The last category deals with computations that are required for *congestion control* or *fault recovery* and for other events such as the start of a new time slice. Again, this evaluation

requires a more manual review of the algorithm implementations and proposals, but it is also possible to measure and time the computations associated with reactive routing.

Evaluation Method The first step is to identify all the places where a recalculation or modification of the routing tables is done, triggered by *congestion control* or *fault recovery*. Then, it is compared whether the recalculation algorithm is similar to a regular recalculation, e.g. at time slice change or at the start of a constellation operation, and it is measured what the difference in runtime is for finding the generated overhead.

Evaluation Questions By examining the locations that trigger recomputation and identifying the severity of the recomputation required, which can be individual routes, the entire table, or an eager recomputation after the arrival of a packet, it is then possible to answer these research questions:

1. How often is the algorithm required to recalculate routing tables?
If the algorithm is slow and requires regular recalculations due to failures or congestion, it can degrade the performance. An alternative route will only be available once the recalculation process has been completed, so if this process takes one second, the previous path will still be utilized for that duration and a potentially broken link is still used. Here, we want to evaluate how often recalculations of the routing tables are required.

2. Is a full recalculation required?
The time it takes to clear a routing table and recompute the next hop for each possible destination satellite is vastly different from the time it takes to calculate a single route. Again, a full recalculation will delay the application of alternative routes due to the longer calculation time. Therefore, the purpose of this evaluation question is to examine the severity of the required recalculation in terms of overhead.

3. How much calculational overhead is generated?
This evaluation question aims to evaluate the actual runtimes generated by the reactive methods of the algorithm for failure and congestion control when a recalculation as found by evaluation question ***1.*** occurs.

5 Simulation Platform

Before creating routing modules and testing algorithms against benchmark scenarios, a suitable simulation platform is essential. This platform must provide the following capabilities:

- Simulate and visualize data networks.
- Create satellite constellations dynamically.
- Update topology by dynamically establish and tear down ISLs and GSLs.
- Provide a clean interface for routing module extensions.
- Create traffic from arbitrary patterns.
- Create and configure different benchmark scenarios with support for:
  - Define the constellation using the Walker notation.
  - Decide and swap the utilized routing algorithm.
  - Apply temporary and permanent non-functionalities of ISLs.
  - Specify the used traffic pattern.
- Collect and store statistics during simulation.

5.1 Omnetpp and FLoRaSat

In sec. 3.2 multiple network simulators with their benefits and limitations were presented. To achieve this work's goal, the choice fell on the *omnetpp* based FLoRaSat because of its superior visualization capabilities and the already existing satellite mobility fundamentals.

Omnetpp is an object-oriented simulation tool with an architecture, where each simulation consists of networks, modules, and channels. Multiple modules that are interconnected by channels are assembled into networks, utilizing a domain-specific language known as NED. The entire simulation can be configured in INI files in which parameters can be set or overwritten.

Modules There are two types of modules, *simple modules* and *compound modules*. Simple modules have no submodules, while compound modules can have multiple submodules. Both types can hold gates that are either input, output or inout gates hat serve as docking points for module connections. A non-abstract module is composed of a NED file that describes the structure of the module and the lower level C++ class it represents and eventually a C++ header and a source file with the actual implementation of the represented class.

The structure of the module defined in the NED file consists of a list of parameters and gates with type and name and, in the case of composite modules, an additional list of submodules and optional connections between submodules and/or the composite module itself. Parameters are variables or values defined either by default values, in the .ini configuration file, or by the parent module or network that contains the module containing the parameter.

Depending on the module type, the represented class inherits either from the *omnetpp* class *cSimpleModule* or *cModule*. Inside the header and source file it is possible to override methods of the *omnetpp* superclass that are called by the simulation kernel during simulation. Important methods that are regularly overridden are lifecycle callbacks such as *initialize*, which is called at the start of a simulation, *finish*, which is called right before the simulation ends, or callbacks called due to events such as *handleMessage*, which is called when a self-message is received or a message arrives at any input/inout gate of the module.

Channels They represent connections between modules and can be used to model data links or internal connections. For example, an internal connection within a satellite between a module that processes packets and a queue can be modeled with a connection. Another use case is an ISL between two satellites. Channels are a special kind of module in the sense that they are also C++ classes that can be subclassed to add new functionality. Creating new channel types is not always necessary, as *omnetpp* provides two generally applicable channel implementations. One is the *DelayChannel*, which can be used to specify the delay a message takes to propagate through the channel, and the other is the *DatarateChannel*, which can be used to specify the data rate of the channel, i.e. how long it takes to push the entire packet into the channel. It is also possible to disable a channel so that it drops all packets it receives, or to set a bit error rate and packet error rate with help of parameters.

Networks A simulation follows a top-down approach, with the network as the top component, also known as the system module. For each simulation it is necessary to configure which network to use, usually in the ".ini" configuration file. Several general parameters for simulations can be set, such as the maximum simulation time and multiple seeds for random generators, which increases the reproducibility of simulation runs.

5.1.1 FLoRaSat software architecture

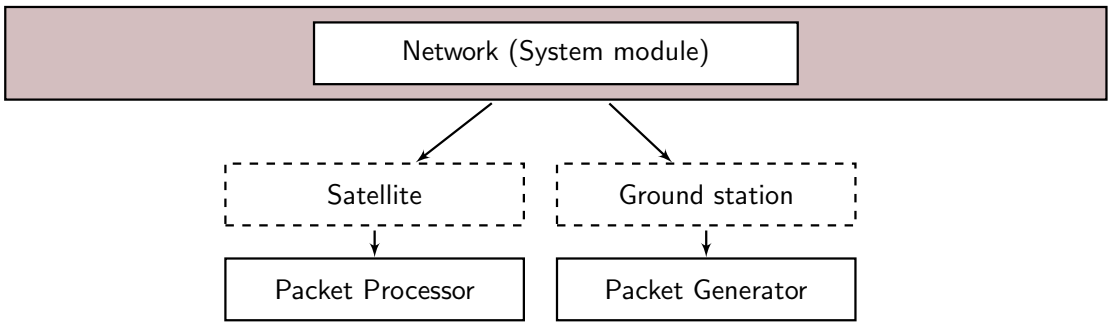


Figure 14: Initial FLoRaSat architecture.

In fig. 14 the initial high-level structure of FLoRaSat is shown. For simplicity, the parts of the framework that relate to the research of Lora satellite networks that are not relevant to this work are omitted and not considered further here. The two nodes in the bottom layer ("Satellite" and "Ground station") are dashed because they are module vectors, i.e. they can exist multiple times. The following paragraphs describe the missing functionality and the proposed modules to be added to the architecture to solve mentioned problems.

Problem 1 Currently, it is already possible to have multiple satellites, which is the first step in creating a constellation, but it is necessary to manually configure the orbital parameters for each satellite in the configuration file. This is feasible with a small number of satellites, but not with the size of modern mega-constellations with hundreds of satellites. To solve this problem, a "constellation creator" module is needed that is able to add satellites to the satellite module vector and configure them with orbital parameters to form a Walker constellation, given only the Walker type, Walker notation and altitude.

Problem 2 Links between satellites and between ground stations and satellites are currently not dynamic. They are set statically in the network file and do not take into account the physical constraints that would apply in a real constellation, such as tear downs in polar regions or the minimum elevation required between a satellite and ground station to allow communication. In addition, the delay of channels in *omnetpp* is not automatically updated, so the delay remains the same even if the distance between two nodes changes, which can lead to incorrect propagation delays for ISLs between satellites in different planes and GSL. A "topology control" module is required that creates and tears down ISLs and GSL as needed, and periodically updates the existing inter-plane ISLs channel and GSL channel delays.

Problem 3 To test the response of routing algorithms to unplannable and unforseeable events such as link failures, a module is required that applies these events to the simulation environment. An "failure event manager" module is required, which takes a configuration file with an arbitrary number of events as input and schedules them as part of the DES. Events are composed of a simulation time at which the event should occur, the target satellite, and optionally the target ISL direction. If no target direction is specified, a complete node failure is assumed. To model temporary failures, it must also be possible to create repair events that reactivate the failed links or nodes.

Considering the proposed modules, the evolved architecture of FLoRaSat is shown in fig. 15.

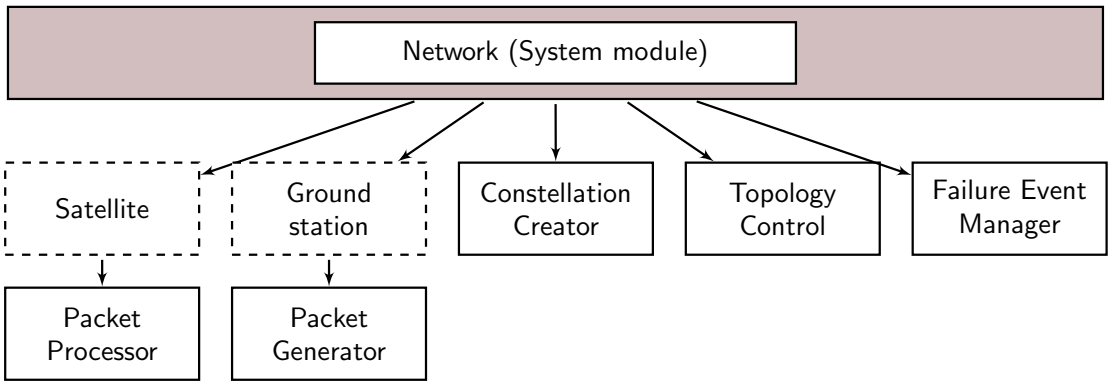


Figure 15: Extended FLoRaSat architecture.

5.2 Constellation Creator

Algorithm 3: Calculate Constellation Parameters

Data: $type$: `Walker type`, $i : pq/p/f$: `Walker Notation`

$raanSpread \leftarrow$ **if** $type = Star$ **then** $180°$ **else** $360°$
$\Delta\Omega \leftarrow raanSpread/p$
$\Delta\Theta \leftarrow 360.0/q$
$\Delta f \leftarrow (360.0 * f)/(pq)$
foreach $plane$ ***in*** $[0..p]$ **do** `// Iterate over planes`
  $raan \leftarrow \Delta\Omega * plane$
  foreach $planeSat$ ***in*** $[0..q]$ **do** `// Iterate over sats per plane`
    $index \leftarrow planeSat + plane * q$
    $meanAnomaly \leftarrow (plane * \Delta f + planeSat * \Delta\Theta) \mod 360.0$
    yield $index,\ raan,\ meanAnomaly$

To configure each satellite with the necessary orbital parameters they must be determined by the "constellation creator" module. Each satellite requires the following parameters to be placed correctly: index, initial mean anomaly and RAAN of the satellite, as well as the altitude, eccentricity (which is equal to 0 for typical Walker constellations) and inclination of the orbit. Given a Walker notation and additionally the altitude and Walker type, it is possible to apply the formulas in tab. 2. The algorithm given in algo. 3 uses the above mentioned formulas and provides the necessary dynamic parameters for each satellite, namely index, RAAN and mean anomaly. All other parameters are static for the constellation and can be assigned to each satellite without computation.

To add these capabilities to FLoRaSat, it is necessary to hook into the lifecycle callbacks of *omnetpp*. The constellation creation module must override the "initialize" callback that is called by the simulation kernel when the simulation is initialized. After loading the Walker notation, altitude, and Walker type from the configuration file, the module uses the described algorithm to create a new satellite module for each set of values, initializes it, and adds it to the network's satellite module vector. The integration and workflow is sketched in fig. 16.

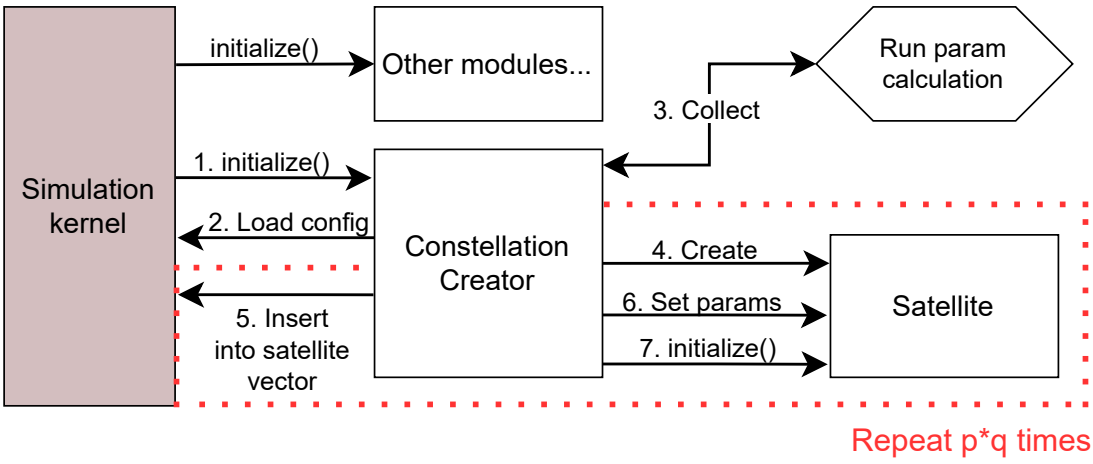


Figure 16: Constellation creator integration into FLoRaSat.

5.2.1 Validation of satellite positions

To show that the created constellations match the given Walker notation, the satellite positions in FLoRaSat must be sufficiently close to their expected positions. For this purpose, STK is used to create a representation of the constellations. Since STK is widely used in research and industry, this work assumes that the calculations of STK are valid and thus can be used to validate the satellite positions of FLoRaSat. By extracting the LLA from STK and FLoRaSat, it is possible to compare the longitudes and latitudes of the satellite pairs.

The latitude and longitude of each satellite pair are converted to positive numbers (adding of +90 or +180) and then the deviation in percent is calculated as follows:

$$Dev_{lat} = \left| \frac{Lat_{STK} - Lat}{Lat_{STK}} * 100 \right|, \quad Dev_{lon} = \left| \frac{Lon_{STK} - Lon}{Lon_{STK}} * 100 \right|$$

where $Lat_{STK}(Lon_{STK})$ is the latitude(longitude) extracted from STK, while $Lat(Lon)$ comes from FLoRaSat.

Tab. 4 shows a position comparison for the Iridium constellation, consisting of the LLA-satellite positions extracted from STK and FLoRaSat, combined with the calculated latitude and longitude deviation per satellite pair, as well as the standard and mean deviation.

Satellite	STK-Lat	STK-Lon	Lat	Lon	$Dev_{Lat}[\%]$	$Dev_{Lon}[\%]$
0	90.11	79.64	89.88	79.39	0.26	0.32
1	122.92	81.96	122.72	81.70	0.17	0.31
2	155.45	87.50	155.29	87.21	0.10	0.33
3	171.02	56.32	171.10	55.83	0.05	0.87
..	..	..	..	..	..	..
62	95.38	49.99	95.36	49.73	0.02	0.52
63	128.16	52.51	128.17	52.22	0.01	0.56
64	160.59	60.03	160.66	59.67	0.04	0.60
65	166.06	34.83	165.96	34.90	0.06	0.22
Std-Dev.:					0.34	1.56
Mean-Dev.:					0.22	0.71

Table 4: Position comparison between FLoRaSat and STK.

The standard deviation and mean deviation is less than half percent for the latitude while it is about 1.5 and 0.71 percent for the longitude. Since the results are very close to STK, and due to the fast propagation delay of ISLs, we consider them as acceptable.

5.3 Topology Control

This module is responsible for creating and deleting the communication channels between satellite modules and between satellite and ground station modules. Furthermore, a periodic update of the inter-plane ISLs and GSLs is required account for the changing distances due to movements. The establishment of intra-plane ISL, inter-plane ISL and GSL is separated from each other.

5.3.1 ISL channel creation

In order to model a realistic routing topology, it is necessary to consider the prerequisites for establishing a link between satellites, which were discussed in 2.2.2. Algo. 4 shows the creation of satellite links via intra-plane links, while inter-plane links are shown subsequently in algo. 5. It is sufficient to create the intra-plane ISLs once and not update them, since the distance and thus the delay will not change after that. On the other hand, inter-plane ISLs require a periodic check of the connection preconditions in Walker-Star and, for both types, an update of the current channel delays. To calculate the aforementioned channel delays, the topology control needs to know the propagation delay of the simulated physical ISL devices in $\frac{ms}{km}$.

Algorithm 4: Create Intra-plane ISL

```
Data: i : pq/p/f : Walker Notation, isl_delay : ms/km
foreach plane in [0..p] do                                  // Iterate over planes
    foreach planeSat in [0..q] do                           // Iterate over sats per plane
        firstSat ← sat[plane][planeSat]
        secondSat ← sat[plane][(planeSat + 1) mod q]
        delay ← isl_delay * Distance(firstSat, secondSat)
        Connect(firstSat, secondSat, delay)
```

Algorithm 5: Create/Update Inter-plane ISL

```
Data: type : Walker type, i : pq/p/f : Walker Notation, isl_delay : ms/km
foreach plane in [0..p] do                                  // Iterate over planes
    foreach planeSat in [0..q] do                           // Iterate over sats per plane
        firstSat ← sat[plane][planeSat]
        lastPlane ← if plane = p − 1 then true else false
        if type = Delta then
            if lastPlane then
                secondSat ← sat[0][(planeSat + f) mod q]
            else
                secondSat ← sat[plane + 1][planeSat]
            delay ← isl_delay * Distance(firstSat, secondSat)
            ConnectOrUpdate(firstSat, secondSat, delay)
        else
            if lastPlane then Break all                     // Last plane has a seam
            secondSat ← sat[plane + 1][planeSat]
            if |Lat(firstSat)| > 70° or |Lat(secondSat)| > 70° then
                // If one satellite is above 70° or below -70° latitude
                Disconnect(firstSat, secondSat)
            else if Ascending(firstSat) ≠ Ascending(secondSat) then
                // If satellite pair has different direction
                Disconnect(firstSat, secondSat)
            else
                // They are allowed to connect
                delay ← isl_delay * Distance(firstSat, secondSat)
                ConnectOrUpdate(firstSat, secondSat, delay)
```

5.3.2 GSL channel creation

The positions of ground stations and satellites change with each update, so a periodic update of channel delays is required to stay close to reality. In addition, the prerequisite for communication is the elevation (see 2.2.3), which must be checked to be higher than a given *min_elevation* each time the link is updated. Otherwise, the channel between the GS and the satellite must be deleted. Algo. 6 shows a sketch of the implemented algorithm.

Algorithm 6: Create/Update Ground-Satellite Link connections

```
   Data: groundstations, satellites, gl_delay in ms/km, min_elevation
1  foreach sat in satellites do
2      foreach gst in groundstations do
3          elevation ← Elevation(sat, gst)
4          if elevation > min_elevation then
5              delay ← gs_delay * Distance(sat, gst)
6              ConnectOrUpdate(sat, gst, delay)
7          else
8              Disconnect(sat, gst)
```

5.3.3 Integration into FLoRaSat

Implementing the topology control module requires a mechanism for regular updates, and it is important to consider which parts need to be created once and which need to be updated frequently. To solve the problem of recurring periodic updates, this work will leverage the self-message capabilities of *omnetpp*. Modules can schedule self-messages using the simulation kernel, which are sent back to the module at a desired simulation time. By using the "initialize" lifecycle hook within the topology control module implementation it is possible to create the ISLs and GSLs and schedule an initial self-message. If the method that handles the returned self-message schedules a future self-message itself, a periodic loop is created. At the same time, the method can update the channels that need frequent updating, i.e. GSLs and inter-plane ISL. The interval of the periodic update loop can be configured for each simulation and is set to 1s by default. Shorter intervals result in greater computational overhead, while longer intervals can speed up the simulation but reduce the accuracy of the simulation. The overall flow is shown in fig. 17.

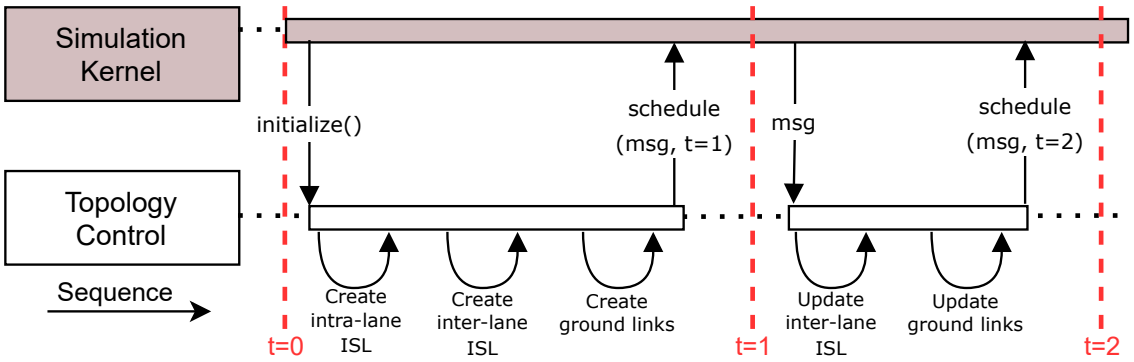


Figure 17: Topology Control integration into FLoRaSat.

5.4 Failure Event Manager

The "failure event manager" module is responsible for simulating link and node failures. For this, it is necessary to establish an event representation, an input file format, and to specify the integration of these events into FLoRaSat. In lst. 1, the proposed representation for events used to apply states to ISLs is shown. Mandatory attributes are the time at which the event should occur (t), the satellite index to which the new state should be applied ($satId$), and the value of the new state ($value$). The direction attribute (dir) is optional and if not specified, the state will be applied to all directions, which is equivalent to a node failure.

```
type "Set-Isl-State Event" = {
    t: SimulationTime,                              // event time
    satid: number                                   // target sat. index
    dir: Optional<"LEFT"|"UP"|"RIGHT"|"DOWN">       // target ISL direction
    value: "WORKING"|"DISABLED"                     // applied state
}
```

Listing 1: Proposed Event representation.

5.4.1 Event file format

In order to allow platform users to specify arbitrary events and combinations, there must be a file format which can be loaded by FLoRaSat. To achieve this goal, this work expands upon the scenario scripting methodology offered by the *inet framework* by integrating the previously suggested event type. With the inclusion of the event type, FLoRaSat can parse and process multiple events given in the above proposed format. A sample file demonstrating the format is provided in lst. 2. Here, directly after start of the simulation, the up ISL module (sender and receiver) of satellite 21 is disabled (link failure), while satellite 25 experiences a node failure with no working ISLs. These two events are grouped inside an $<at>$ tag, which can be used to define t for all at once. At $t = 50s$ into the simulation, the up ISL module of satellite 21 recovers, while satellite 25 does not recover and remains a failed node for the remainder of the simulation.

```
<scenario>
    <at t="0s">
        <set-isl-state satid="21" dir="UP" value="DISABLED" />
        <set-isl-state satid="25" value="DISABLED" />
    </at>
    <set-isl-state t="50s" satid="21" dir="UP" value="WORKING" />
</scenario>
```

Listing 2: Sample event configuration file.

5.4.2 Integration into FLoRaSat

The integration is achieved through the creation of an *FailureEventManager* class that extends the *ScenarioManager* class of the *inet framework*. This class is able to load and parse *xml* files whose paths are configured in the simulation configuration file. By overriding the *initialize* lifecycle hook, it is possible to load a *xml* file from the file system after the simulation start. Parsing of the content yields 0 to n events, which are then grouped by event arrival time and a self-message is scheduled for each group. After reception by the *FailureEventManager*, the module applies the desired state changes of the group's events with assistance of the *TopologyControl*.

By knowing the changed states, the *TopologyControl* can then update the channels and enable or disable them according to the desired state. The flow of the procedure is visualized in fig. 18.

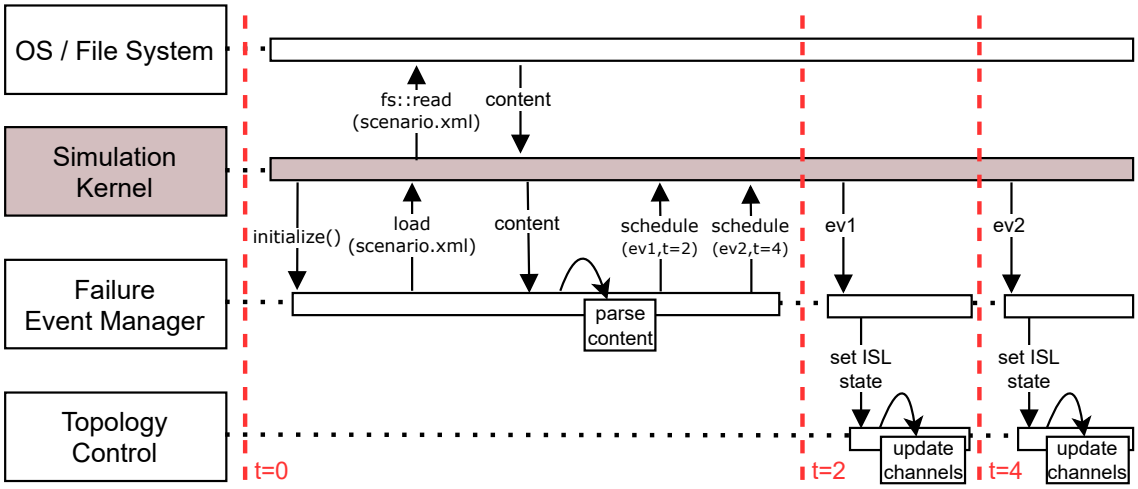


Figure 18: Failure Event Manager integration into FLoRaSat.

5.5 Routing API

To allow the flexible addition of new routing algorithms to FLoRaSat without touching modules not directly related to the routing process, e.g. the packet processor module or queues, this work implements a scheme based on the idea of software defined networks. For this purpose, each satellite contains an abstract placeholder submodule, the routing module. The routing module is the part that varies depending on the algorithm, while the packet processor is static and does not require code changes when implementing a new algorithm. Omnetpp allows placeholder modules to be swapped by specifying the type name within the simulation configuration file. This can be used to configure the algorithm that will be used to replace the routing module. Consequently, it is crucial for these classes to possess the identical interface or base class as the abstract placeholder class. For this purpose, an abstract class called "RoutingBase" was created, which has to be extended by other algorithms to enable the usage as the routing submodule on a satellite. Aside of allowing a dynamic module replacement, the abstract class also allows us to specify method signatures which are inherited by other routing modules and can then be called externally,

e.g. by the packet processor. Unfortunately, creating a routing module for satellites is not enough. For certain algorithms, a satellite may send a do-not-send packet message to its communication partners, e.g. for congestion control, and still receive messages because the platform currently lacks support to add handlers to ground stations. To make this possible, another dynamic routing module for ground stations is required, which inherits from an abstract "RoutingBaseGs" class. With the addition of this two new modules, the further evolved architecture of FLoRaSat is shown below in fig. 19.

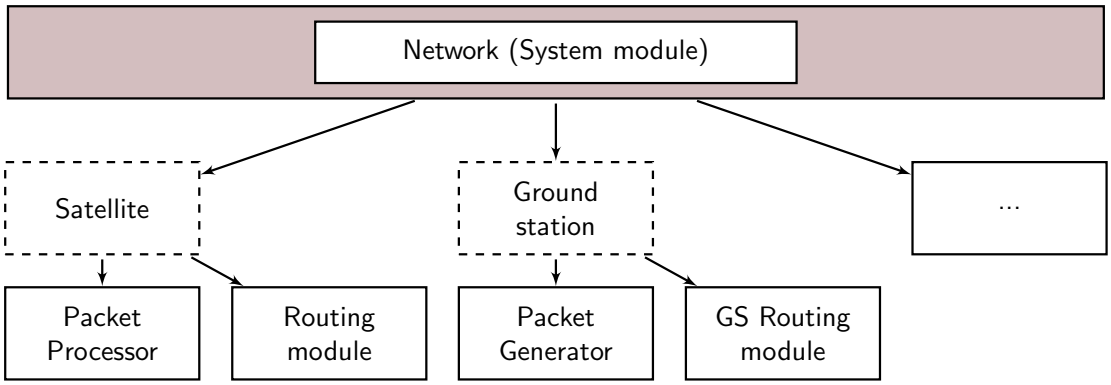


Figure 19: FLoRaSat architecture with integration of routing.

5.5.1 PacketProcessor

The packet processor is responsible for receiving packets from ISLs and GSLs, processing packets, queuing packets if there is still a packet in process, and processing the queue one by one until it is empty. To enable the collection of metrics, the packet processor is also responsible for collecting and updating the processing delay, queuing delay, propagation delay, and transmission delay associated with the currently processed packet. To process a packet, the processing packets holds a reference to the routing module of the same satellite and uses this reference to call methods of the public API of the routing module.
Another task involves updating the hop count of the processed packet and checking it against the configured maximum hops in order to handle the Time to Live (TTL). The packet processor is employed for this to avoid duplicating the same logic and code in each routing algorithm implementation.

5.5.2 Satellite routing module

The routing module can not only handle routing requests through the *handlePacket* callback but also other events (recorded by the packet handler) through method calls. These other events could be packet drops, e.g., due to failed links, the current *queue size* after arrival of a packet, or packets which have reached the maximum number of hops. Moreover, the module can process packets that are marked as satellite-to-satellite messages rather than ground-to-ground messages, which may be utilized for protocol management and state exchange. Each routing module has a reference to the packet processor (called sender) which can be used to instruct the packet processor to send the packets via ISL or GSL, or to drop it.

In lst. 3, mandatory and optional handlers of the RoutingBase class are shown with

```
// Abstract base class RoutingBase
class RoutingBase : public cSimpleModule {
    // must be overwritten by all algorithms
    void handlePacket(Packet *pkt) = 0;
    // Optional
    void handleTopologyChange() { /* empty */ };
    void preparePacket(Packet *pkt) { /* empty */ };
    void wrapUpPacket(Packet *pkt) { /* empty */ };
    void handleMaxHopsReached(Packet *pkt) {
        sender->dropPacket(pkt);
    };
    void handlePacketDrop(Packet *pkt, Reason reason) { /* empty */ };
    void handleQueueSize(int size, int maxSize) { /* empty */ };
}
```

Listing 3: Simplified definition of the routing base class.

their (simplified) signature. Hereafter follows a description of the handlers with potential use-cases and possible special aspects that may need to be taken into account when overriding them.

Handler: handlePacket Whenever a packet is processed, the packet processor calls this handler with a pointer to the packet as an argument. After the call, *handlePacket* is responsible for the further processing of the packet, as the packet processor is lazy and needs instructions from the routing module. Constellations in FLoRaSat support two types of packets, normal (ground station to ground station) and control (satellite to satellite or satellite to ground station) packets. If the packet is of type control and the processing satellite is the destination, this handler must implement the logic that should be executed after the arrival of a control packet, which depends on the implemented routing algorithm. Otherwise, if the packet is not destined for this satellite or is of normal type, the routing module must perform the following tasks: First, decide on the next hop, which could be a gate to a ISLs or GSLs, or to drop the packet if it is not routable. Second, communicate the next instruction to the packet processor by calling the *sendMessage* or *dropPacket* method on the sender reference variable. This is the only handler whose implementation is mandatory for all extending routing modules.

Handler: handleTopologyChange To simulate precomputed time slices with a constant routing topology, i.e., no GSLs or ISLs connects or disconnects occurred during that duration, the topology control calls the *handleTopologyChange* handler of each satellite routing module after performing a pre-plannable topology change. Thus, because they need to be pre-plannable, topology changes instructed by the "failure event manager" do not trigger a handler call and cannot be identified in this way. Furthermore, this handler can be used to compute new routes and store the next hop inside a forwarding table, or to reset state.

Since topology changes are deterministic and can therefore be calculated and stored prior to operation, it is permissible to inform all satellites at all locations around the world of topology changes, not just the affected satellites. If such a restriction is required, the unaffected satellites can simply return within the callback. This method eliminates the

need to compute and store pre-planning data, but still allows simulation and testing of algorithms that follow this approach during routing.

Handler: preparePacket After a packet processor receives a packet from a GSL, this handler is called with a pointer to the packet. The base implementation is empty because not all algorithms require a response to this event. A use case might be to add a routing protocol specific header to the packet or to compute the full path to the destination satellite and store it in the packet header.

Handler: wrapUpPacket This handler is called before the packet processor sends a packet to a GSL. The base implementation is empty, as with *preparePacket*, as it is not always needed. One possible use is to remove routing protocol specific headers before the packet is reinserted into the regular network.

Handler: handleMaxHopsReached Since the packet processor keeps track of the lifetime of packets, it calls this handler with a pointer to the packet when the TTL drops to 0. Without overriding the base logic, the handler will simply tell the sender to drop the packet and do nothing else.

It is important to note that satellite-to-satellite packets may end up in this handler if they reach their destination at the same time that the TTL reaches 0.

Handler: handlePacketDrop Whenever the packet processor drops a packet, whether because an interface is down or the queue is overflowing, it uses this handler to report it to the routing module. Unlike the other handlers, this handler takes a pointer to the packet and the reason it was dropped as an argument. The base implementation is empty, but can be overridden for use cases such as congestion control or fault recovery. When the routing module uses the sender to send a packet to a next hop, the routing module can specify the transmission to be silent so that the packet processor does not report the drop of the packet to that handler, which is useful, for example, to implement a broadcast and some links may not be connected, but we do not care if the message is not acknowledged.

Handler: handleQueueSize When a packet is added to the satellite's queue because another packet is being processed, the packet processor uses this handler to tell the routing module the actual queue size and the maximum queue size. Here, the base implementation is again empty, because it is not required for proactive routing algorithms, but an override could be used for example to react to certain thresholds the queue exceeds, which requires a reaction of the system, like broadcasting a request to reduce the number of packets that are sent to this satellite.

5.5.3 Groundstation routing module

The ground station routing module has two functions: It provides a handler for control packets received from satellites and it provides a method for selecting the first and last satellite for a destination ground station. Control packets are intended to influence the way the first and last satellites are calculated. For example, if the ground station knows that a first satellite is overloaded, it might make sense to use another first satellite if available. As shown in lst. 4, the default implementation of *handlePacket* is empty, because it is not necessarily needed, while *calculateFirstAndLastSatellite* uses DSPA to calculate

the first and the last satellite with the shortest distance between them as the default implementation.

```
1  // Abstract base class groundstation routing
2  class RoutingBaseGs : public cSimpleModule {
3      void handlePacket(Packet *pkt) { /* empty */ };
4      std::pair<int,int> calculateFirstAndLastSatellite(int dstGsId) {
5          return DSPA(this.gsId, dstGsId);
6      };
7  }
```

Listing 4: Simplified definition of the groundstation routing base class.

5.5.4 Addition of new routing algorithms

The work described so far allows an easy and flexible addition of new routing modules to FLoRaSat. To add a simple routing algorithm, it is sufficient to extend the routing base for the satellites listed in lst. 3. After that, the following line shows how to assign a specific routing type in the configuration file, here DSPA routing, for all satellites:

*.loRaGW[*].routing.typename = “DSPRRouting”

where loRaGW is the name of the satellite module vector and * is a wildcard to match all satellite ids. Theoretically, it would also be possible to create and specify different routing module types for specific satellite ids, for example, to simulate a network of satellites, some of which are used as controllers for a number of non-controller satellites.

If the added algorithm also makes use of control packets that may be sent to ground stations, or if it is desired to change the way the first and last satellites are calculated (e.g. to use hops instead of distance), this can be accomplished by extending the routing base class for ground stations shown in lst. 4 and overriding the desired handlers. To use a custom ground station routing module, the following line must be added to the configuration:

*.groundStation[*].routing.typename = “DDRARoutingGs”

5.6 Statistics and Results Collection

The code for FLoRaSat is written in C++, which is not suitable for an easy statistical analysis of results. At the time of writing, data science is typically done in Python or R. To be able to use these programming languages, FLoRaSat needs to serialize the collected results into a well-defined data format and write them to the file system, which can then be read by other software. Storage efficiency is important because simulations can have packet counts in the millions. A suitable serialization format is .csv (Comma-separated values), which is widely known and does not require any additional dependencies.

To evaluate the performance of the algorithms, we are interested in the following statistics. First, for each packet, any delays it has experienced, the time it was created and recorded, the type of packet (data, control), and finally, if a packet was dropped and not delivered,

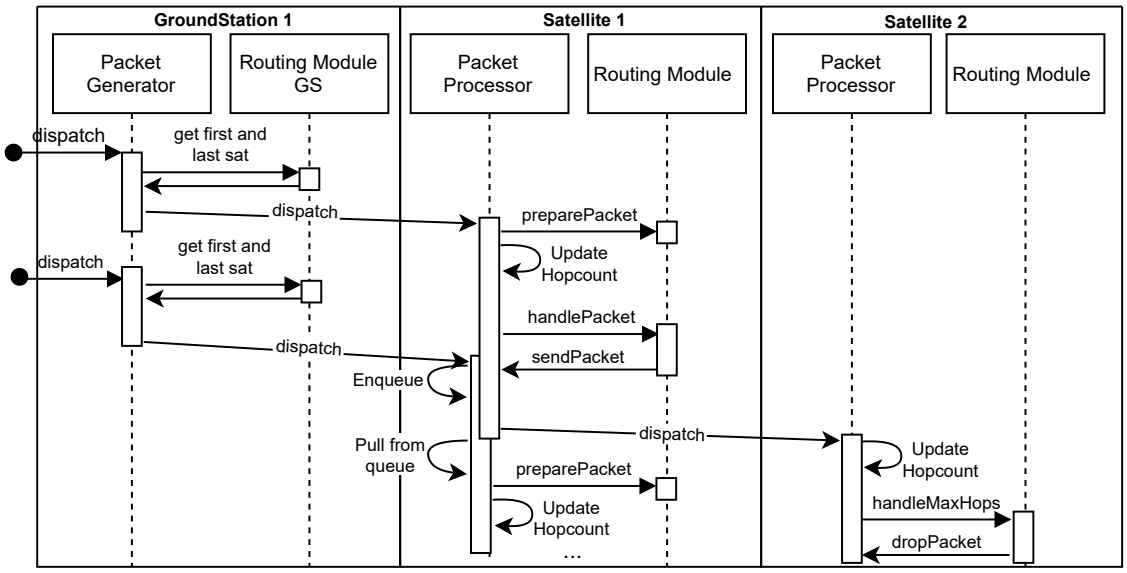


Figure 20: Routing sequence example.

the reason it was dropped. Second, in order to analyze the routes for each packet, e.g. to calculate the distance traveled, the hops taken by the packet with the corresponding locations must be recorded. Finally, for the constellation, satellite states such as the size of the processing queue can be tracked.

5.6.1 Recording of packet statistics

To represent the aforementioned packet statistics it is possible to represented one packet by one entry. The serialized packet statistics are written to a .csv file, and when interpreted as a table (simplified for ease of viewing), the data looks like tab. 5. Each time a packet is delivered or dropped, an entry is added to the table with the associated collected statistics. In addition to metrics that can be read directly from the table, such as hops or source and destination, the table allows the calculation of the following metrics:

- *Packet loss*, which is the number of dropped packets divided by the number of packets (overall and over time).

- *E2E delay*, which is the sum of all delays.

ID	Type	Drop	Sats	GSs	Hops	Creat.	Rec.	q./proc./trans./prop. delay
1	N	-	-/-	0/4	7	1.0	1.014	0.42/3.8/0.87/8.75
2	N	-	-/-	4/0	6	1.0	1.016	0.52/3.7/0.97/8.85
3	N	6	-/-	4/0	8	1.014	1.0195	0/2.5/0.4/3
4	C	-	33/66	-/-	1	1.024	1.028	0/2.5/0.5/1
					...			

Table 5: Sample packet statistic file.

5.6.2 Recording of routes

Because routes are arbitrarily long, it is difficult to represent them as a single entry in a table without deviating from the CSV format. To solve this problem, multiple entries are added with the ID of the packet they belong to. So whenever a packet is recorded into tab. 5, FLoRaSat will also record the route by adding one entry per hop to a table as shown in tab. 6. By recording the type and LLA of each hop, it is possible to calculate the total distance traveled and to draw the route on a map. Furthermore, by selecting a dropped packet from tab. 5, it is possible to find the last satellite or ground station and its location where the packet was dropped by filtering for the last entry of the packet id.

pID	Type	ID	Latitude	Longitude	Altitude
1	G	0	54.2	56.4	0
1	S	6	57.5	57.1	650
1	S	7	62.3	60.5	652
1	G	4	65.2	60.7	0
2	G	4	65.2	60.7	0
		...			

Table 6: Sample route recordings file.

Route pre-processing Due to the large number of packets and the fact that some packets may have many hops, the size of this table/file can become very large. Also, the format of routes split into multiple entries makes them difficult to use in data analysis because they must first be reassembled. The goal is to pre-process the routes and create an object for each route with the packet id, hops, and calculated total euclidean distance as fields. Our first attempt was to implement a solution in pure Python as a prior step to the data analysis. Unfortunately, with increasing length and more traffic in the simulation, this implementation was no longer capable of processing this large amount of hops, which is why it was replaced by a Rust implementation. Although the code is very similar, this implementation is able to reduce the runtime by a factor of 50 and no longer runs into memory limits. The Rust implementation is built as a cdylib library so that its bindings can be imported and called directly from Python code, allowing it to be used as a preliminary step to data analysis as it was previously. It works as follows: First, the library loads the raw route file from the FLoRaSat results folder, second, it assembles the routes, third, it calculates the Euclidean distance of the route by converting LLA to ECEF coordinates, and finally, it stores the generated route objects on the file system so that they can be loaded when needed. In addition, the library provides a function to reload the pre-processed routes from the file system.

The reason why this implementation is not directly added to FLoRaSat is that it would slow down the simulation significantly due to the expensive calculations. Moreover, one may not be interested in the routes at all, so this approach saves computing power. The same argument holds for 5.6.3.

5.6.3 Recording of satellite state

Satellite state, like routes, cannot be encoded by a single entry, but requires multiple entries to represent a history over time. To minimize required storage space, an entry

only holds the data of the state changes and will not copy the unchanged state into the new entry. Therefore, the unchanged state is represented by an empty column as shown in tab. 7.

Satellite ID	Timestamp	Queue size	Metric 2	Metric 3
1	1.005243	1	-	-
1	1.005548	0	-	-
2	1.005552	-	1	-
1	1.007233	-	-	1
2	1.008234	1	-	-
		...		

Table 7: Sample satellite state recording. Empty columns are represented by a dash.

Currently, FLoRaSat only collects data on queue sizes, but the addition of new metrics is already considered in the functionality of the library, facilitating future extensions. Conceivable measurements could be the battery charge of a satellite or the load on the battery.

Pre-processing This representation again requires pre-processing, which has been added as functionality to the Rust library introduced in 5.6.2. The pre-processing is done in multiple steps:

First, the library iterates over all entries of the same satellite, sorted by timestamp in ascending order, and fills all empty columns with 0 until an initial value is found. After an initial value is found, this value is added to all subsequent lines of the satellite until another value is found. This is repeated for all found values and then for all satellites until all lines are complete. If necessary, it is possible to merge multiple lines with similar timestamps to reduce space requirements.
Second, a state representation object is created for all rows. This state object is filled with the values of the corresponding row and is valid from the timestamp of this row until the timestamp of the next row of the same satellite. To represent the state for the entire simulation time, each satellite is given an initial state object with all values set to 0. This object is valid until the timestamp of the first line (or until the end if no line exists for that satellite). The last state object for each satellite is valid until the simulation end timestamp.
Third, the library stores the pre-processed satellite state in a space-saving format on the file system for later use.

5.6.4 Storage locations

After simulating a scenario with a specific seed (referred to as a run), FLoRaSat stores the results in a well-defined storage path that is build from the constellation name, the name of the benchmark scenario, the routing algorithm name, and the run identifier (0, for the first run with seed A, 1 for the second run with seed B, ...). For example, the result files of the first run of a configuration named “BurstTraffic” using the Iridium constellation and DDRA as the routing algorithm, are stored at the following location: “[...]/Iridium/BurstTraffic/DDRA/”. These files are “0.stats.csv”, “0.routes.csv” and “0.queues.csv”. The results of another configuration with a different name that uses the Iridium constellation

would also be stored in the Iridium folder along with the BurstTraffic folder, but in its own folder.

This structure allows multiple configuration runs to be stored without affecting old result files, and it allows structured access by subsequent pre-processing methods such as Python scripts, which will be of interest in the following subsection. Also, by giving the files a unique name per seed, they are accessible in the same folder and the preprocessing steps can combine and average the results.

5.7 FLoRaSat CLI

This subsection introduces the functionality of a CLI, which has been developed to allow the creation of failure scenario files that can be loaded by the "failure event manager" (see sec. 5.4), and to automate the generation of visualizations for extracted statistics and states. It can be configured to save generated scenarios and graphs to a desired path and to load extracted statistics directly from the FLoRaSat results folder.

5.7.1 Scenario Generation

Manually creating scenario files for the "failure event manager" would be tedious and unlikely to represent randomly distributed link and node failures and random subsequent recoveries. For this reason, a subcommand has been added to CLI that takes as input the probabilities of link and node failures, the probabilities of link and node recoveries, a number of phases into which the scenario should be divided, the length of the simulation, a warm-up period, and multiple seeds. Then, the following is done for each seed. After using the seed to initialize a pseudorandom number generator, the specified warm-up time T_w is subtracted from the simulation time T_s and the difference is then divided by the number of phases N_p, which gives the length of each phase. With that, the start of phase i can be calculated as: $T_w + ((T_s - T_w)/N_p) * i$. Now for each phase, each ISL and node has the given failure probability of changing to the failed state and any ISL or node currently in the failed state has the given recovery probability of changing to the working state. Then, the state changes are written to the scenario file associated with the start of the phase as timestamp.

An example of calling this subcommand is shown below. Here, two failure scenario files are created for the given seeds 24543 and 74245, considering 1000 satellites, a simulation length of 6000s, five phases, and an warm-up time of 50s. Each ISL and satellite has a failure probability of 10% and a probability for recovery of 30%. The files will then saved in the default CLI results folder with the name "OneWeb0.xml" and "OneWeb1.xml".

```
#!/bin/bash
florasat scenario create failures 1000 6000 5 0.10 0.10 0.3 0.3 \
--warmup 50 --name OneWeb --seed 24543 74245
```

5.7.2 Visualization Generation

This subcommand of the CLI uses state-of-the-art data science tools such as Pandas for data processing and Plotly for graph generation, with the goal of allowing future research to focus on improving routing algorithms rather than graph generation. In addition, the CLI can execute methods of the FLoRaSat Rust library to load and pre-process routes and satellite states so that they can be included in subsequent data analysis.

After loading the stored result from the location defined in 5.6.4, the routes can be pre-processed using the following command:

```
#!/bin/bash
florasat statistics --cstl Iridium --name Normal \
    --alg DDRA Directed --run 1 --preprocess-routes
```

The argument "--run 1" indicates that combining multiple runs with different seeds is not desired, but only the evaluation of the first run is pre-processed. Afterwards, with the following command, it is possible to create the graphs for hops (--hops) and for the traveled distances (--distances):

```
#!/bin/bash
florasat statistics --cstl Iridium --name Normal \
    --alg DDRA Directed --run 1 --hops --distances
```

The resulting graphs are shown in fig. 21, namely a cumulative distribution function of the hop count on the left, and another CDF for the distance packets traveled on the right.

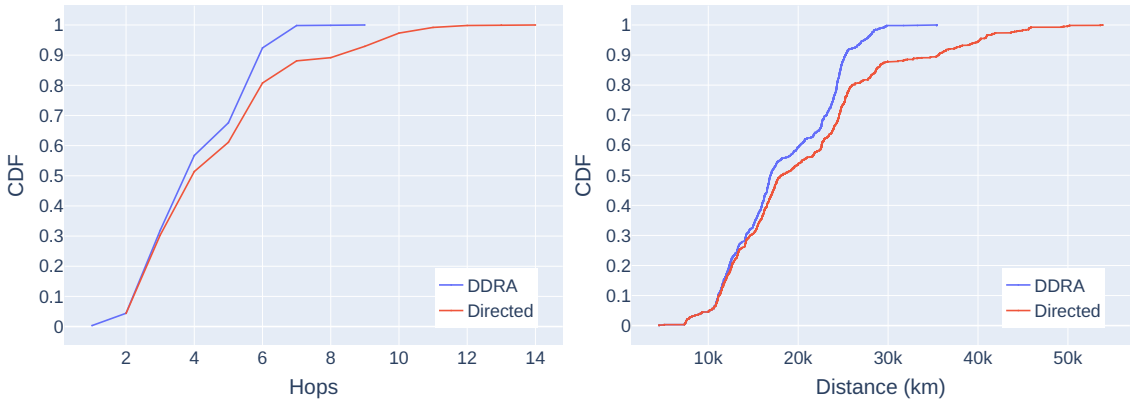


Figure 21: Automatic generated hopcount and distance CDF for DDRA and Directed Routing.

The CLI supports the creation of charts and maps for most of the automatable metrics found in related work (see 3.1.5), which are shown in tab. 8. For many evaluation cases, this should be sufficient, so that little time needs to be spent on data processing and scripting in future research. And in cases where this is not sufficient, other supported

metrics and output formats can be included in the CLI due to the extensive statistics collection built into FLoRaSat.

Metric	Output	Preproc. Routes	Preproc. Sat.-States
Hopcount	Multiline CDF	✗	✗
Distance	Multiline CDF	✓	✗
Delivery ratio	Scattergraph	✗	✗
Packet loss	Scattergraph	✗	✗
Throughput	Scattergraph	✗	✗
Drop heatmap	World heatmap	✓	✗
E2E Delay	Multiline CDF	✗	✗
Sub-Delays	box plot per delay type	✗	✗
Avg. queue size	Barchart	✗	✓
Protocol-Overhead	Scattergraph	✗	✗
Failure comparison	Multiple Scattergraph	✗	✗
Queuing delay comparison	Multiple Box plots	✗	✓

Table 8: FLoRaSat CLI: Supported metrics and graphs.

5.7.3 Metric guideline

This subsection tries to suggest a prioritization and guideline for the usage and expressiveness of the metrics shown in tab. 8.

- **Hopcount and Distance** allow for a first classification of the delivered performance of algorithms. In the same constellations, hop count and distance are correlated to each other, since each hop taken leads to an increase in physical distance. This can be used to distinguish the abilities of algorithms to find the shortest path as two paths with the same number of hops can have different lengths.
- **Delivery ratio, Packet loss and Throughput** help to classify the quality of service, since a high packet loss or a low delivery ratio requires frequent retransmissions or lead to entire information being lost and consequently reduce the throughput of the constellation. They are related because the sum of packet loss and delivery ratio is 100% and throughput is the number of delivered packets times the packet size.
- **Drop heatmap** can be used to visually inspect the locations of packet losses on the globe, which helps to identify potential points of failure of the routing algorithms.
- **E2E Delay and Sub-Delays** visualizes the time from a sender to the recipient and the contributions of the sub-delays. This metric is one of the most important to classify the quality of service but should be lower prioritized than delivery ratio because it only tracks the delays for delivered packets that were delivered in order. Thus, this metric could indicate good performance although only 1 out of 100 packets was delivered.
- **Avg. Queue size.** displays the avg. satellite queue sizes over time. To interpret this metric, it is important to consider the strong influence by the number of satellites and used traffic pattern. With that an congestion estimation is possible, but the queuing delay seems to be more informative and should be prioritized.
- **Protocol-Overhead** helps to differentiate the overhead created by algorithm negotiation by plotting the relation of data to negotiation packets.
- **Failure/Queuing delay comparison.** visualizes the packet loss and queuing delay of several different scenarios executed in the same constellation, with the possibility

to display several algorithms. This is particularly useful for comparing performance of algorithms at different failure and traffic severity levels.

5.8 Experiment Pipeline

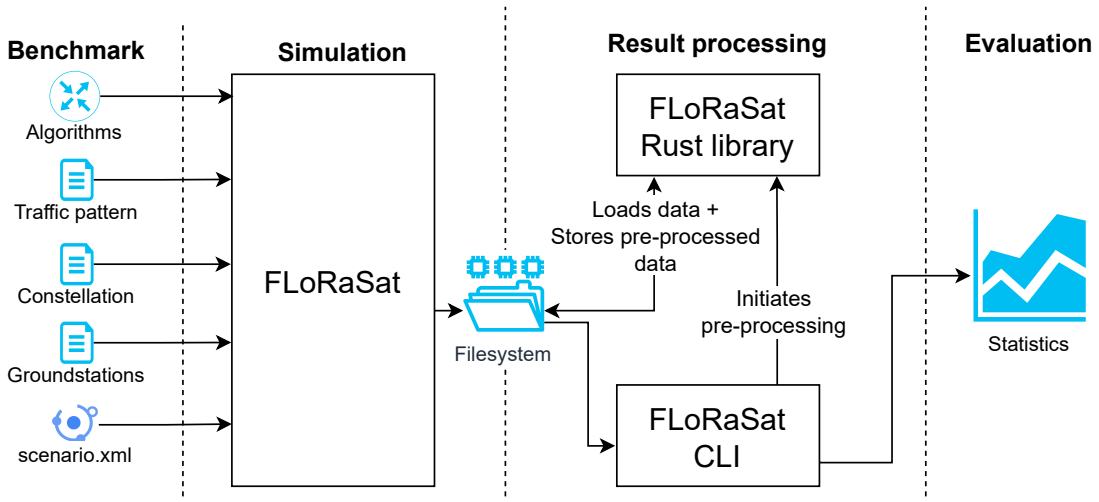


Figure 22: LEO algorithm experiment pipeline.

The preceding work enables us to create a four-step experiment pipeline, illustrated in fig. 22. It allows us to go through the process of configuring benchmarks, running them, processing the results, and ultimately creating visualizations of the extracted data. This is accomplished by combining the created simulation platform with the CLI enabled by the Rust library. These steps can be described as follows:

1. Benchmark In this step, the benchmark scenario for the desired experiment is created by combining several configuration files for the number and location of ground stations, the traffic pattern, and the parameters of the constellation. It also specifies the routing algorithms to be used, and adds a scenario file describing the frequency and timings for the occurrence of failures and recoveries for links and nodes.

2. Simulation By using the above configuration and scenario files as input to FLoRaSat, it is possible to run the same benchmark against multiple specified algorithms. The simulator user does not need to make any changes to the simulator platform code; instead, a configuration change is all that is required to replace the routing algorithm, for example. FLoRaSat will write the collected results to the file system which will be used in the next step. Due to the dependence on random generators in some parts of the simulator, the simulation can be run with different seeds, which is called a run.

3. Result processing Here the FLoRaSat output, which requires pre-processing, is loaded, pre-processed and stored on the file system in the space-saving msgpack format using the Rust library. This step is triggered by the FLoRaSat CLI for both recorded routes and recorded satellite states.

4. Evaluation During analysis, the FLoRaSat CLI loads the pre-processed data together with the remaining unprocessed output and uses data science and statistical methods to create the desired graphs and charts for the metrics which were configured by the user.

6 Algorithm Comparison

This chapter combines the developed simulation platform with the proposed benchmark scenarios to demonstrate and validate the developed capabilities. To this end, the created experiment pipeline will be used. Using the metric collection capabilities of the simulator and the visualization and scenario generation functionalities of the CLI, a comparison between the different algorithms will be made based on the established evaluation questions.

To demonstrate that the platform is capable of simulating arbitrary Walker-Star and Walker-Delta constellations, the algorithm comparison is divided into two parts. First, Walker-Star algorithms and associated benchmark scenarios, and second, Walker-Delta algorithms and benchmarks. Both parts start with a parameter study of different constellation and system parameters for this type of constellation, together with an evaluation of the performance changes due to the parameter variations. Then, the other benchmarks are implemented and the associated evaluation questions are answered, i.e., regular operation, response to failures, response to congestion, and finally algorithm efforts. At the end of each part, a summary of the algorithm evaluations is provided to draw a final conclusion in terms of performance, scalability, and network and economic efficiency.

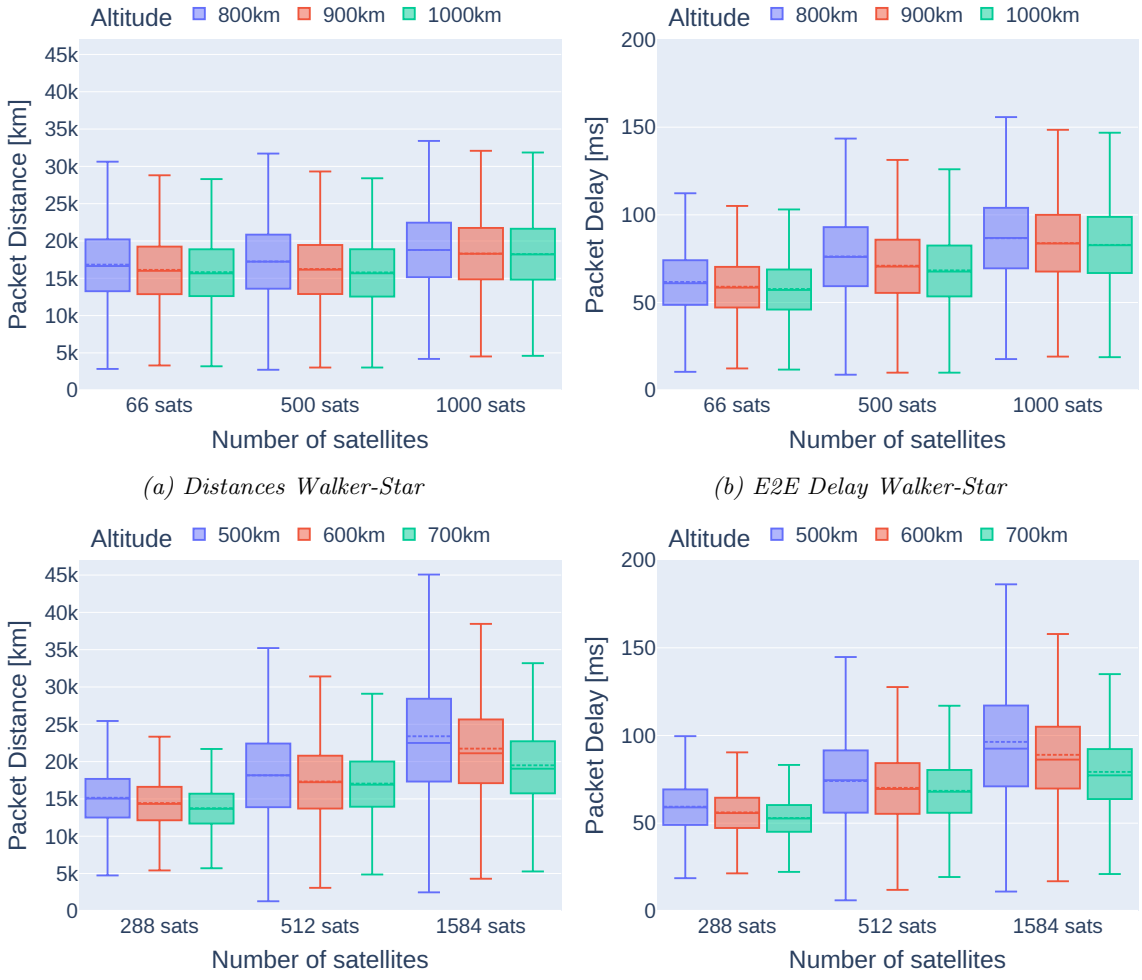
Reproducibility All simulations were run on an Ubuntu Linux system powered by the Windows Subsystem for Linux with an Intel Core^{TM} i5-10600KF@4.10GHz CPU and 32 GB of RAM. To reduce the impact of outliers, all parameter studies were run three times, and the other benchmarks were run twice with different seeds for omnetpp's random number generator. Additionally, the constellation base day at the simulation start is changed between the different seeded runs so that the constellation does not provide the same initial routing topology.

6.1 Parameter Study

This subsection presents the results of the parameter study discussed in sec. 4.1. Using three base constellations, the influence of altitude, inclination, and data rate on system performance was studied. For the parameter study, both algorithms used for the Walker comparison were used, i.e. DDRA and Directed Routing for Walker-Star and DDRA and DISCO for Walker-Delta. Since for each algorithm the simulation was run with three different random seeds, the individual results for each run were combined and averaged. Subsequently, the averaged results are also combined and averaged for both algorithms. The reason for this is that by averaging the two algorithms, the influence of the respective algorithm on the results is reduced and the influence of the respective varied parameter is highlighted.

6.1.1 Altitude influence on system performance

Walker-Star As shown in subplot (a) of fig. 23, the altitude of the constellation has an influence on the distance of the packets. Recall, packet distance is defined as the physical distances from the source GS through all intermediate satellites to the destination GS. The explanation is that with otherwise the same system properties, i.e the same minimum elevation, a higher altitude increases the number of possible first and last satellites because a higher altitude favors elevation. If there are more possible first and last satellite pairs, the route may be shorter than at a lower altitude. For the large constellation due to the already



(a) Distances Walker-Star

(b) E2E Delay Walker-Star

(c) Distances Walker-Delta

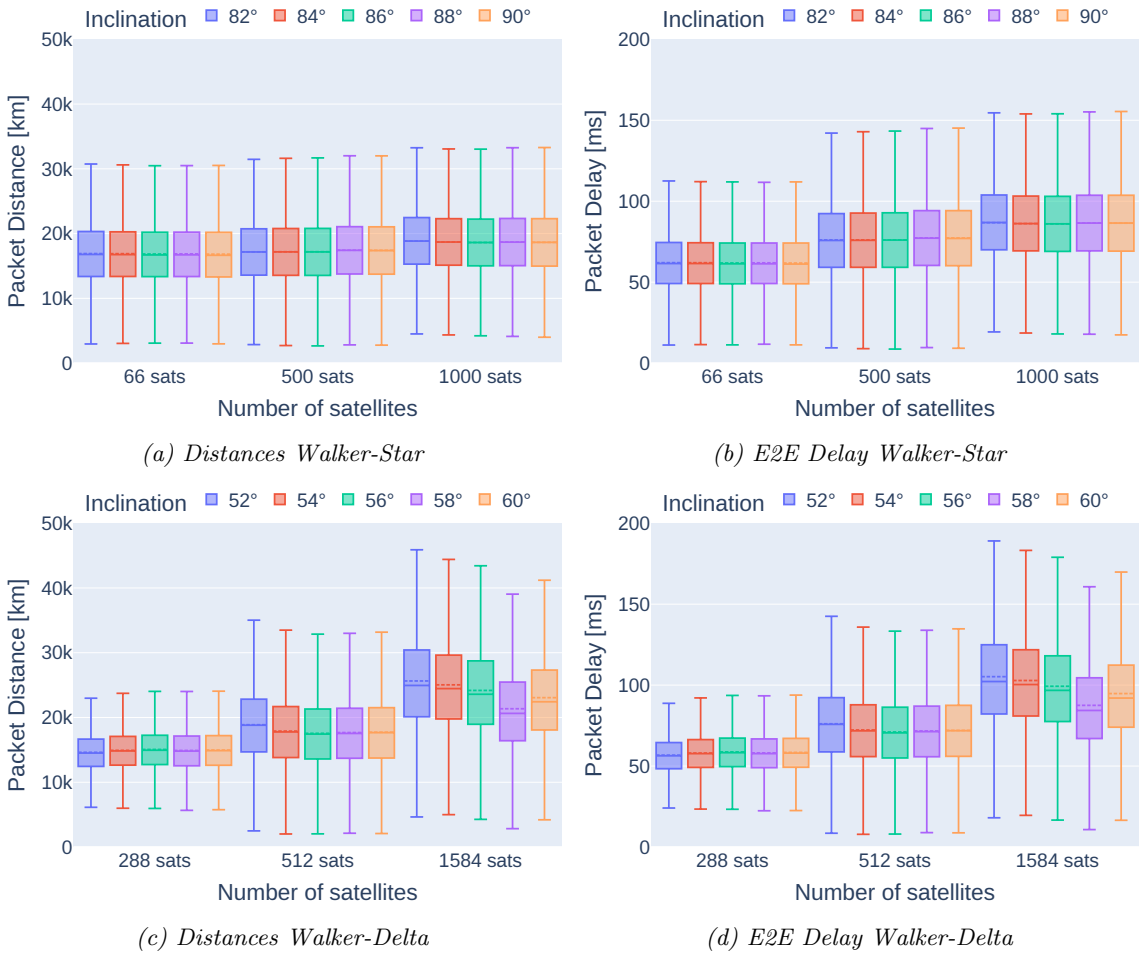
(d) E2E Delay Walker-Delta

Figure 23: Parameterstudy: Influence of altitude on Walker constellations.

larger set of potential first and last satellites, this effect loses impact and only results in small decrease of the mean and median of the packet distance. For all constellations, a downward shift of the upper quartile ($Q3$) and the lower quartile ($Q1$) is observed, as well as a convergence to the mean of the upper and lower whiskers (indicating outliers). While a downward shift of the quartiles and the convergence of the upper whisker are positive for the performance, the upward shift of the lower whisker is negative, indicating that the distance for the previous packets with short distances increases, which is logical considering that the altitude was increased, and if the optimal first or last satellite was already reachable with lower elevation, the now higher altitude increases the length of the route. Due to the large influence of the propagation delay caused by the path length on the E2E delay, the same effect can be observed here, visualized in subplot (b). To classify the results, it is necessary to consider the traffic distribution used, which consists of random global source-destination pairs. The increasing lower whiskers show that performance would not benefit in the same way as for average global traffic if most of the traffic was between pairs with small physical distances.

Walker-Delta For Walker-Delta, the same effects as for Walker-Star are observed with larger impact, which is shown in fig. 23. Again, we attribute this to the extended set of first and last satellites. This observed effect suggests that a reduction in the minimum elevation, if possible, would have resulted in a similar route shortening effect, and the use of communication modules that allow this would be beneficial. However, the impact on the height of the lower whiskers is also stronger than for Walker-Star, which results in an even higher degrade of the performance for communication pairs with small physical distances. For even higher altitudes, however, we expect that the average packet distances will increase again, since the set of valid first and last satellites will stop growing.

6.1.2 Inclination influence on system performance



(a) Distances Walker-Star

(b) E2E Delay Walker-Star

(c) Distances Walker-Delta

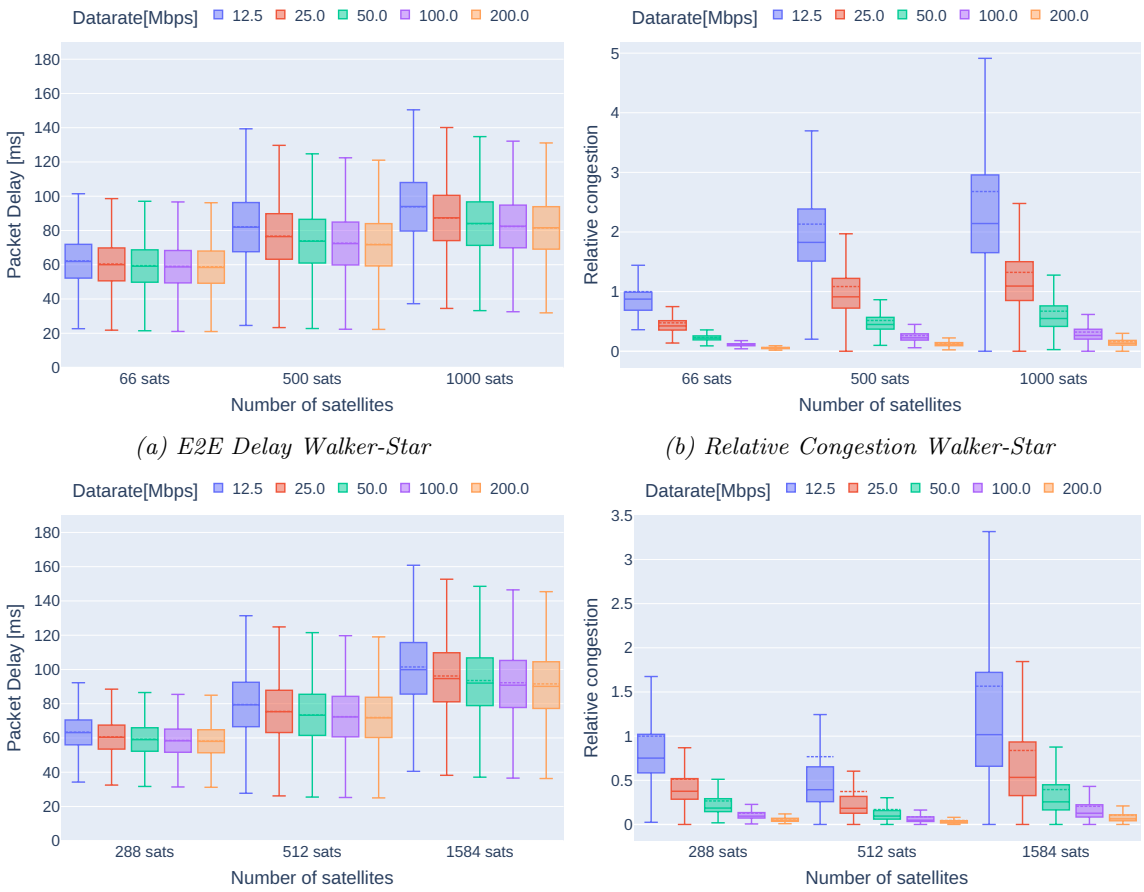
(d) E2E Delay Walker-Delta

Figure 24: Parameterstudy: Influence of inclination on Walker constellations.

Walker-Star Looking at (a) of fig. 24, it can be seen that for the smallest constellations, there is hardly any change between the different inclinations, while for the constellation with 500 satellites, there is a tiny increase in quartiles, mean, and median with an increasing inclination, although the difference is very small. For the largest constellation small but hardly noticeable differences can be seen. The observed trends are also reflected in the packet delays shown in (b) due to the large impact of the propagation delay. These data suggest that at Walker-Star the inclination of the polar orbits has almost no influence.

Walker-Delta The results for Walker-Delta differ significantly from those for Walker-Star. For the inclinations in the smallest constellation, the results are again very similar, with 52° having the smallest upper whiskers, smaller quartiles, median, and mean, but also a slightly higher lower whisker for both distance and delay. However, in the medium constellation 52° shows the worst performance in both distance and delay, while the other inclinations are more similar with a better performance. For the largest constellation, on the other hand, the results show large differences between the inclinations. Here 58° shows by far the best performance, with smaller whiskers, quartiles, and mean and median values. The reason for that is that the planes, due to their inclination, are potentially arranged in such a way that they allow shorter distances between traffic source-destination. In terms of system performance, it can be concluded that no general trend for the influence of the inclination at Walker-Delta can be derived; instead, an optimal inclination must be determined by parameter study for a given constellation and expected or modeled traffic source-destination distribution, which can in contrast to Walker-Star significantly influence the experienced distances and delays.

6.1.3 Data rate influence on delay and congestion



(a) E2E Delay Walker-Star

(b) Relative Congestion Walker-Star

(c) E2E Delay Walker-Delta

(d) Relative Congestion Walker-Delta

Figure 25: Parameterstudy: Influence of installed data rate on delay and congestion.

Walker-Star By looking at fig. 25 (a), it becomes evident that an increase in data rate results in smaller packet delays. For all constellation, this effect can be observable, albeit with less significant impact on smaller constellation. This is not surprising since the data rate is correlated with the transmission delay that occurs at each hop, and constellations with smaller number of satellites allow for smaller average hop counts. Therefore, faster data rates should be installed, especially in large constellations, to compensate for the disadvantages of the higher average hop counts.
Furthermore, (b) of fig. 25 shows the relative congestion in the constellation. Here, the boxplots were created using the average congestion of each satellite in the constellation calculated from the number of packets in the queue over the entire duration of the simulation with a sampling time of 100 microseconds. All calculated average congestion values are normalized for the data rate increase factor, meaning that a 2-fold increase in data rate, would expect a 2-fold increase in congestion to be relatively equal. In addition, the values were increased so that the mean value of 12.5Mbps in the smallest constellation is equal to 1.
The graph shows that the relative congestion decreases by a factor of about 2 for each increase in the data rate in all constellations. This indicates that the total non-relative congestion remains almost at the same level for each data rate. But, even though the trend in congestion is the same for increasing data rates in each constellation, the congestion itself is higher in larger constellations. The medium constellation has an average relative congestion that is more than twice that of the small constellation, while the large constellation experiences an even higher congestion, but one that increases at a lower rate than the small to medium constellation. With the highest data rate, the congestion drops to a very similar level in all constellations. This again shows that large constellations benefit the most from high data rates in Walker-Star.

Walker-Delta For Walker-Delta, exactly the same behavior of decreasing average packet delays with increasing data rate can be observed for each constellation, as shown in fig. 25 (c). Again, the relative decrease in E2E delay between each step is greater for larger constellations because of the contribution of the transmission delay, which is correlated to the average required hop count, and higher data rates decrease the transmission delay per hop.
In terms of congestion, the same trend is observed between data rate levels as in Walker-Star, while between constellation sizes, the medium constellation has lower congestion than the small one, which is contrary to Walker-Star. The reason for that could be, that the constellation is able to better distribute the packets among idling satellites, which leads to less queuing. For the 512-sized constellation is noticeable that the mean value is higher than $Q3$ and thus outside the quartiles. This indicates the presence of large outliers, i.e., some satellites are experiencing much larger queues than other satellites. The large constellation again exhibits a higher average congestion.
Considering the delay gains and the almost equal congestion of the results for the data rates 50 Mbps, 100 Mbps and 200 Mbps, the question arises of whether a data rate of 50 Mbps would not already be sufficient to achieve optimum system performance in terms of cost-effectiveness, since modules with higher data rates are likely to increase deployment costs without providing a much better performance according to this parameter study.

6.2 Walker-Star: DDRA and Directed

This subsection is dedicated to a comparison of the performance of the algorithms DDRA and Directed Routing on Walker-Star. For this purpose, the set of benchmarks with the corresponding evaluation questions from sec. 4 is used and evaluated using the two Walker-Star constellations, i.e. the Iridium and an OneWeb-like constellation. The two algorithms differ in type and optimization metric, with DDRA being a reactive algorithm that optimizes to achieve shortest paths, and Directed Routing being a proactive algorithm that optimizes for minimal manhattan distance at each hop. Finally, a summary concludes the performance comparison of DDRA and Directed Routing during regular operation, as well as in presence of failures and congestion, for both small and large constellations.

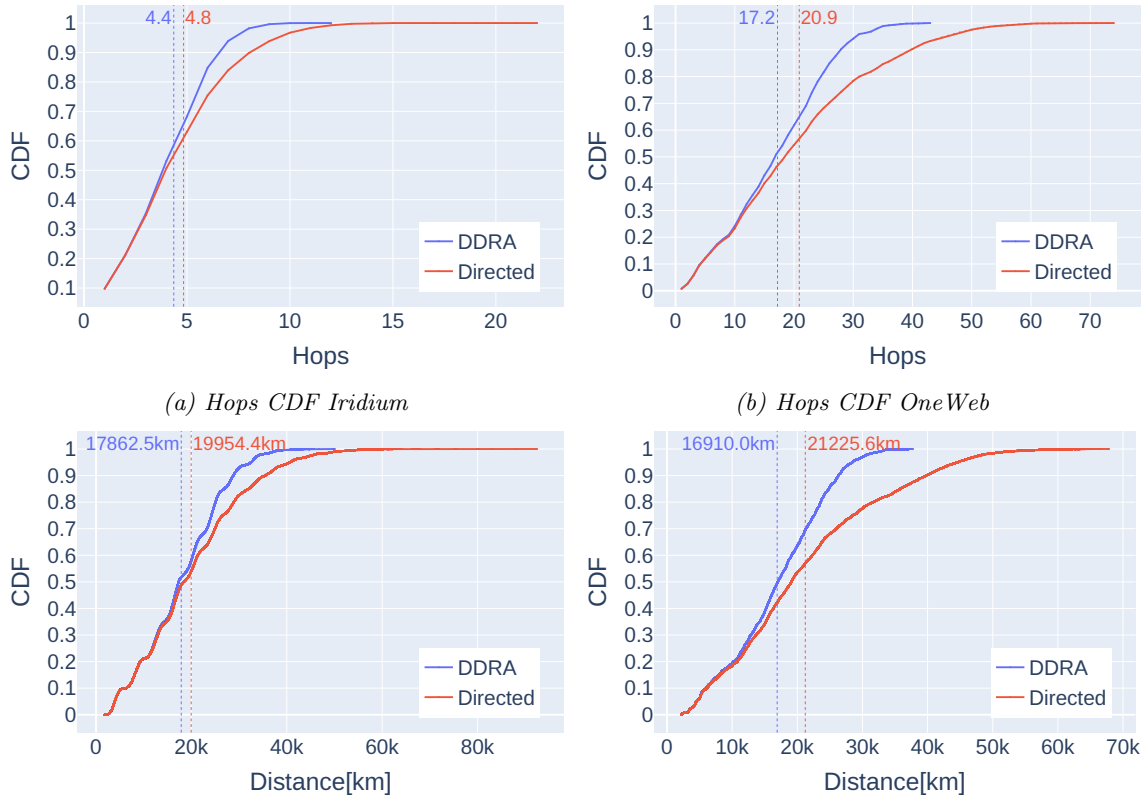
6.2.1 Regular Operation

This benchmark aspires an evaluation of the regular operation without failures and congestion. The traffic pattern and ground stations from sec. 4.2, together with simulation parameters shown in tab. 9 are used to configure the simulations for FLoRaSat. Two different random seeds provide better comparability by excluding that an algorithm is favored by a particular matching seed. Furthermore, two different base days lead to two starting points of the simulation and reduce the impact of optimal initial topologies on the algorithm performance.

Parameter	Common	Iridium	OneWeb
Simulation time	6000 s		
Random seeds	23543, 84242		
Data rate	50 Mbps		
Orbit base day	1.5, 2		
Maximum hops		25	75
Processing overhead	1 ms		
Queue capacity	40 packets		
Maximum GSL per Sat./Gs.	40		
Minimum elevation		5°	45°

Table 9: Walker-Star: Regular Operation Simulation Parameters.

A. How do generated routes differ? In fig. 26 the cumulative distribution functions of hops taken and distance traveled by packets in Iridium and OneWeb are shown. Packets in OneWeb have to take more hops to reach their destination than packets in Iridium, which is not surprising given the large difference in the number of satellites, resulting in a denser satellite network. Since DDRA relies on DSPA to find the optimal shortest path in the absence of failures or congestion, it can be seen that a higher network density allows for finding of shorter routes. This becomes evident when comparing the average route length of DDRA with 17862.5km in Iridium and 16910km in OneWeb, a decrease of 5.33%. Directed routing, however, is unable to benefit from this denser network, as the average route length experiences a 6.37% increase from 19954.4km to 21225.6km. For both constellations, DDRA performs better than Directed Routing in terms of average hops and distances. Also, DDRA's CDFs grow faster, indicating its ability to deliver a higher proportion of packets with fewer hops or distances and because the CDFs reaches 1.0 faster, the maximum number of hops and the maximum distance are smaller, resulting in fewer outliers than with Directed Routing.



(a) Hops CDF Iridium

(b) Hops CDF OneWeb

(c) Distances CDF Iridium

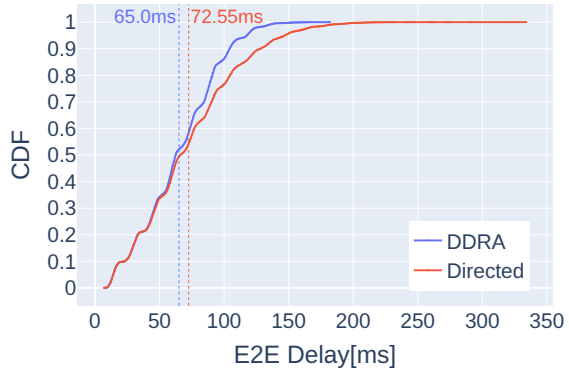
(d) Distances CDF OneWeb

Figure 26: Walker-Star: Hops and Distance CDF for DDRA and Directed Routing.

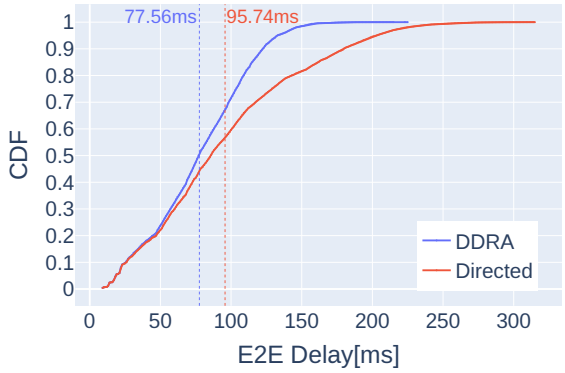
B. Are there noticeable problems with pathfinding? The hop CDF of Directed Routing for Iridium and OneWeb reaches almost 25 and 75 hops, respectively. Since this is one hop below the TTL for the constellation, and only statistics for delivered packets are reported here, this indicates a potential for packet loss due to expiration coming from excessively long routes. In contrast, DDRA can deliver packets without coming close to expiration. Looking at the packet loss shown in tab. 10, it is clear that the above considerations are correct, but the proportion of packets lost is minimal, resulting in an average packet loss rate of only 0.001% with Directed Routing in OneWeb. More interestingly, DDRA has a higher average packet loss than Directed Routing, with 0.003% on Iridium and 0.019% on OneWeb. After analyzing the causes of packet drops, we see that neither DDRA nor Directed Routing experienced dropped packets due to queue overflows. Instead, DDRA drops packets because it is unable to find a route to the destination, although each ground station is continuously connected. This indicates interference caused by a reactive mechanism, which should be the congestion control, since there are no failures present in this simulation. Recalling the congestion control mechanism of DDRA, where congested satellites are removed from the routing topologies of neighboring satellites before the queue is full, one may suspect that the first or last satellite on a packet path is no longer reachable, resulting in a subsequent packet drop since no route to the destination exists. This will come up again later in 6.2.3. Furthermore, the queuing delay shown in fig. 27 does not exceed acceptable thresholds, indicating that the congestion control is overreacting, resulting in packet loss.

	Iridium		OneWeb	
	DDRA	Directed	DDRA	Directed
avg. packet loss	0.003%	0.000%	0.019%	0.001%
Packet drops	99	1	566	17
→ If. down	2	1	26	5
→ Unroutable	97	0	540	0
→ Expired	0	0	0	12
avg. E2E delay	65ms	72.55ms	77.56ms	95.74ms
Percentile				
→ 1% limit	135ms	182ms	154ms	239ms
→ 1% avg.	143.8ms	197.77ms	163.96ms	257.65ms
→ 0.1% limit	153ms	212ms	176ms	276ms
→ 0.1% avg.	158.28ms	221.21ms	184.56ms	287.2ms

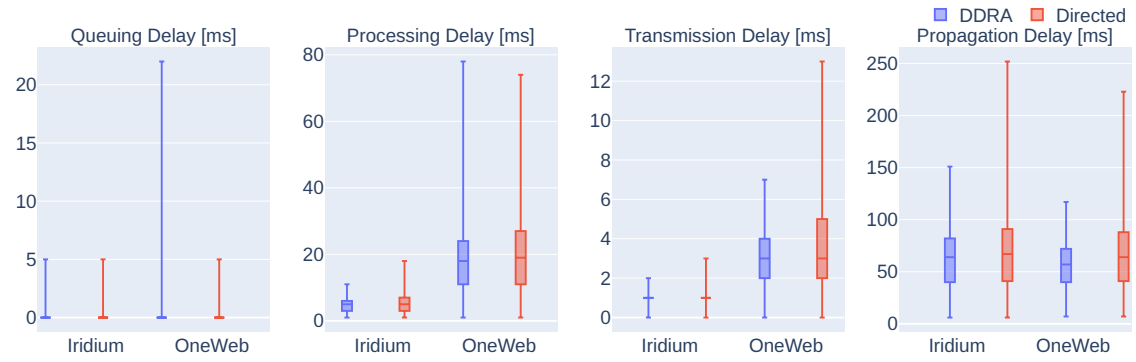
Table 10: Walker-Star: Regular operation numerical results.



(a) E2E Delays Iridium



(b) E2E Delays OneWeb



(c) Delay category distribution

Figure 27: Walker-Star: E2E delay and delay category contribution.

C. How do the enabled packet delays differ? Fig. 27 (a) and (b) show the E2E delay CDFs for delivered packets, while (c) shows the subcategory delays as box plots. It can be seen that DDRA again performs better than Directed Routing, with 10.4% less average E2E delay in Iridium and 19% less in OneWeb. This is further supported by the faster growing CDF and the ability to reach 1.0 with less delay. In addition, an analysis of the average delay of the 1% and 0.1% percentile packets with highest delay, shown

in tab. 10, indicates 27.3% resp. 30.8% less delay in Iridium and 36.4% resp. 35.7% in OneWeb achieved by utilizing DDRA.
On average, the largest impact on the E2E delay comes from the propagation delay with a median of more than 50ms, followed by the processing delay, then the transmission delay, and finally the queuing delay. The queuing delay whiskers for DDRA in OneWeb are more than four times higher than the whiskers of Directed Routing. This indicates larger outliers that may be the result of congestion control mechanisms (failure handling can be ruled out since there are no failures in the scenario) and will be investigated in 6.2.3. In **A**, this work found that a denser grid allows finding shorter paths, which is again confirmed by the lower upper quartile of propagation delay for OneWeb compared to Iridium for both algorithms and the correlation between propagation delay and distance. The near-zero median queuing delay in all scenarios confirms that the simulation parameters and traffic pattern selection were appropriate, as large growing queues were not desired in this benchmark.
Transmission and processing delay correlate with the number of hops and are therefore much lower on Iridium, with small advantages for DDRA with lower quartiles and fewer outliers.

6.2.2 Failure Response

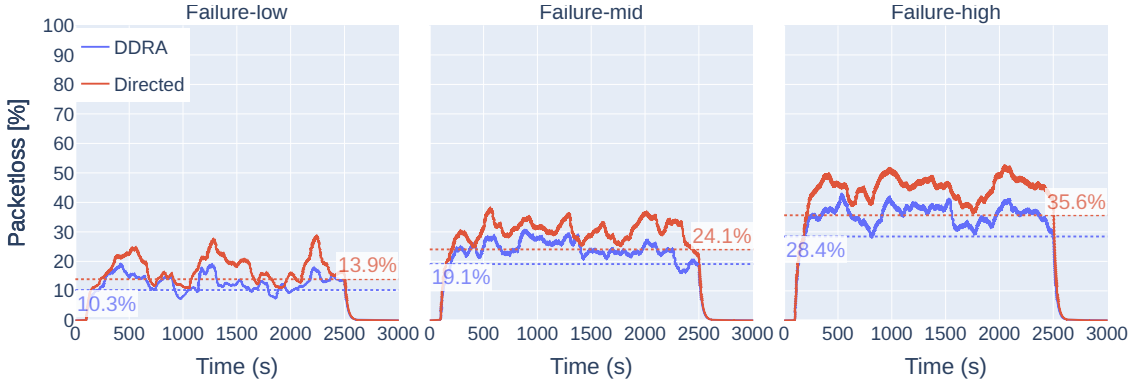
This section realizes the benchmark and answers the evaluation questions from sec. 4.3. For the evaluation, the simulation is configured with the parameters shown in tab. 11. To test different levels of failure severity, three failure scenarios with 30 phases and a warm-up time of 100 s are created for each of two scenario seeds per failure probability level. This results in two low failure probability (0.25%), two medium failure probability (0.5%), and two high failure probability (1%) scenarios for links and nodes. In each scenario, the probability of recovery is set to 60%. Although the simulation time is 3000 s long, the failure scenarios end after 2500 s, to create a 500 s long time span without failures, which allows to test if the algorithms are able to return to regular operation. Unlike DDRA, Directed Routing has no built-in failure handling mechanisms. Therefore, it is possible to estimate the benefits of implementing a failure handling mechanism compared to not having one. However, these gains are only approximations, since DDRA and Directed Routing compute different routes, resulting in different probabilities that a packet will encounter a failed link or node. Thus, the need for error handling is different.
To reduce the influence of beneficial initial topologies and and routing decisions that randomly route packets past failures, the six created failure scenarios are run for two random seeds and base days, and the results are averaged.

A. How capable are the algorithms in handling failures? Fig. 28 shows the packet loss for the aforementioned low, medium and high failure probability scenarios averaged for different base days, random and scenario seeds. By comparing the two constellations it can be seen that the same failure probabilities for nodes and links have an bigger impact on the performance in bigger constellations since the average packet loss rate is always at least 3 times higher in OneWeb than in Iridium. When considering the algorithms it becomes clear that DDRA is able to reduce the packet loss in comparison to Directed Routing in both constellations for all scenarios. The reduction is 24.06% in average for Iridium and 22.29% for OneWeb.

Parameter	Common	Iridium	OneWeb
Simulation time	3000s		
Random seeds	23543, 84242		
Data rate	50 Mbps		
Orbit base day	1.5, 2		
Maximum hops		25	75
Processing overhead	1ms		
Queue capacity	40 packets		
Maximum GSL per Sat./Gs.	40		
Minimum elevation		5°	45°
Failure seeds	43236, 62276		
Prob. link failure	0.01, 0.005, 0.0025		
Prob. link recovery	0.6		
Prob. node failure	0.01, 0.005, 0.0025		
Prob. node recovery	0.6		

Table 11: Walker-Star: Failure Response Simulation Parameters.

(a) Packetloss Iridium



(b) Packetloss OneWeb

Figure 28: Walker-Star: Packet loss comparison for failure scenarios.

B. Are the algorithms able to recognize recoveries? It can be seen that DDRA is able to return to regular operation following the recovery of all failures in the scenario. The packet loss decreases to the same near-zero level as seen during regular operation. This can be seen by looking at the packet loss for both constellations after 2500s when all link and node failures have been removed from the scenario. Directed Routing also returns

to near-zero packet loss, but there is no mechanism that could have prevented this.

C. Does the failure handling enable a still acceptable system performance? If we consider 5% as the threshold for acceptable system performance, there is one case, the medium scenario for Iridium, where DDRA is able to raise packet loss to an acceptable rate. Rather, the observation is that DDRA mechanisms only allow acceptable system performance when packetloss would be close to acceptable even without a mechanism. Any reduction in packetloss is desirable, but considering, for example, the high-failure scenario in OneWeb, a packetloss of 28.4% is better than 35.6% but still remains unacceptable.

6.2.3 Congestion Response

After examining regular operation and failure response, this section investigates the final unstudied routing algorithm capability, congestion control. Simulations will be configures as shown in tab. 11. Here, the maximum queue size is reduced to increase the need for congestion handling to avoid packet drops due to full queues. Three scenarios are created for each sending interval using a 0.03s+1s *burst+idle* pattern.

Parameter	Common	Iridium	OneWeb
Simulation time	1500s		
Random seeds	23543, 84242		
Data rate	50 Mbps		
Orbit base day	1.5, 2		
Maximum hops		25	75
Processing overhead	1ms		
Queue capacity	25 packets		
Maximum GSL per Sat./Gs.	40		
Minimum elevation		5°	45°
Burst+Idle Duration	0.03s + 1s		
Sending Interval	1ms, 0.5ms, 0.25ms		

Table 12: Walker-Star: Congestion Response Simulation Parameters.

A. How capable are the algorithms in handling congestion? Directed Routing does not provide congestion control mechanisms and therefore requires sufficiently large queues and fast packet processing. DDRA employs the concept of notifying neighbors when a certain queue threshold is exceeded. Upon receiving such a notification, a neighbor will remove the congested satellite from its currently loaded connection matrix and clear the routing table. Fig. 29 shows box plots of queuing delays for delivered packets forwarded by the algorithm in the given constellation. The first observation is that DDRA has more outliers than Directed Routing in all scenarios and constellations. While in Iridium the median queue delay and the quartiles are almost the same for both algorithms, in OneWeb the median queue delay, the whiskers and the upper quartiles are much lower for Directed Routing. This shows a better overall performance of Directed Routing, despite the lack of congestion control mechanisms in this algorithm.

B. Which drawbacks are created by the mechanisms? The inferior results of DDRA are made even worse by a closer look at the limited system performance. Fig. 30 shows the packet loss rate in each scenario for both algorithms and constellations. It can be seen that DDRA's congestion handling mechanisms lead to a massive increase in packet loss

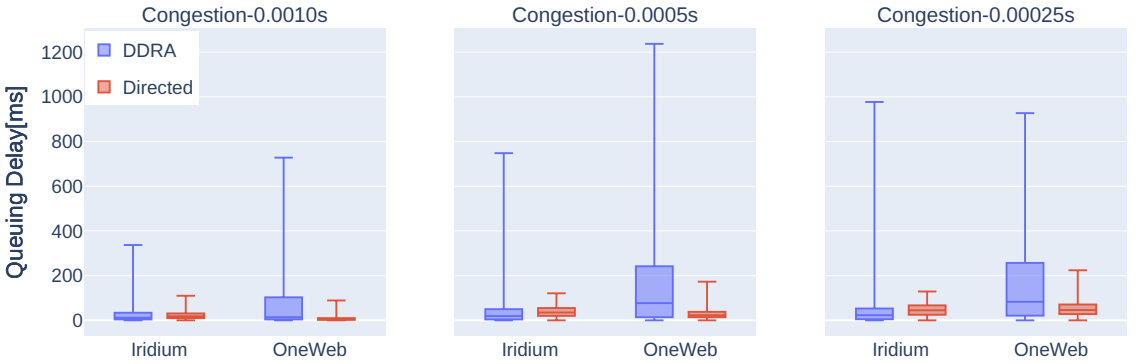
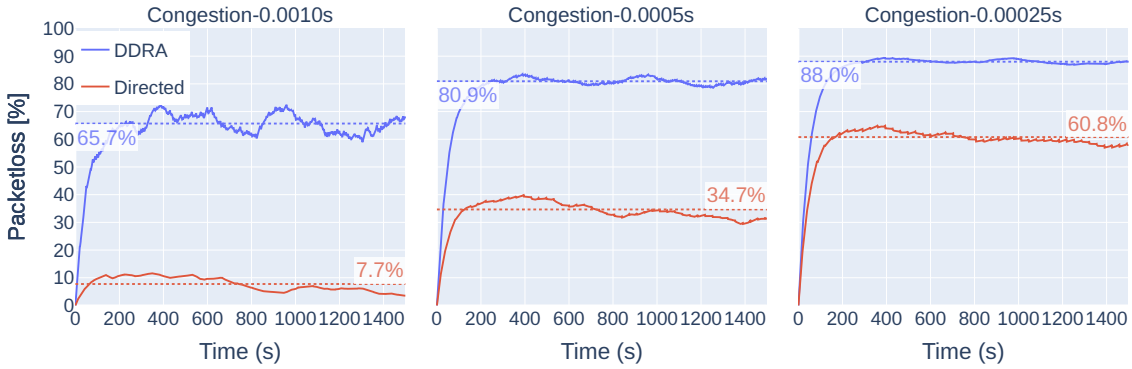
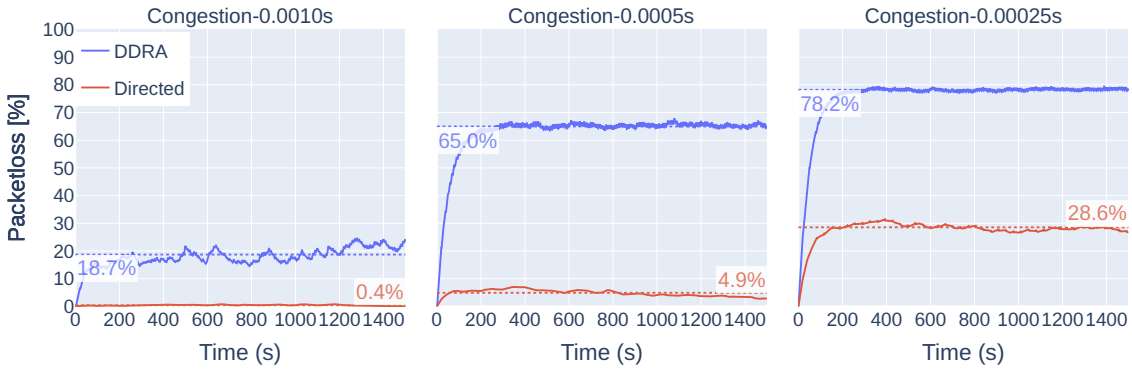


Figure 29: Walker-Star: Queuing delay boxplots for Iridium and OneWeb.



(a) Packetloss Iridium



(b) Packetloss OneWeb

Figure 30: Walker-Star: Packet loss comparison for congestion scenarios.

rates. Upon closer analysis, it becomes clear that the reason for the high packet loss is that when satellites start dropping neighbors from their connection matrix, holes are created in the satellite network. This causes packets to get stuck in loops between pairs of satellites because they both think the next hop of the shortest route is the other satellite, causing the packets to TTL. The reason for this is that if the overloaded satellite only notifies its direct neighbors, the neighbor of the neighbor will think that node is still operational and include it in routing decisions.

In addition to the phenomenon of packets getting stuck in a loop, another problem can

occur. If a ground station sends enough traffic to congest all connected first satellites, other ground stations are no longer able to reach this ground station, since all neighbors of the first satellites will stop to route traffic to these satellites. This could lead either to further loops and thus to packet losses or, in the case of a single overloaded first satellite, to packet losses because the neighbouring satellites cannot find an alternative path.

6.2.4 Algorithm Efforts

A. Computational Efforts

DDRA and Directed Routing have different approaches to path finding. While DDRA uses DSPA to find a suitable next hop, Directed Routing relies on a heuristic, the Manhattan distance, to determine the next node to discover until the destination is reached. This is similar to the A* algorithm, but Directed Routing may not find the optimal shortest path because it does not consider the path to the node.

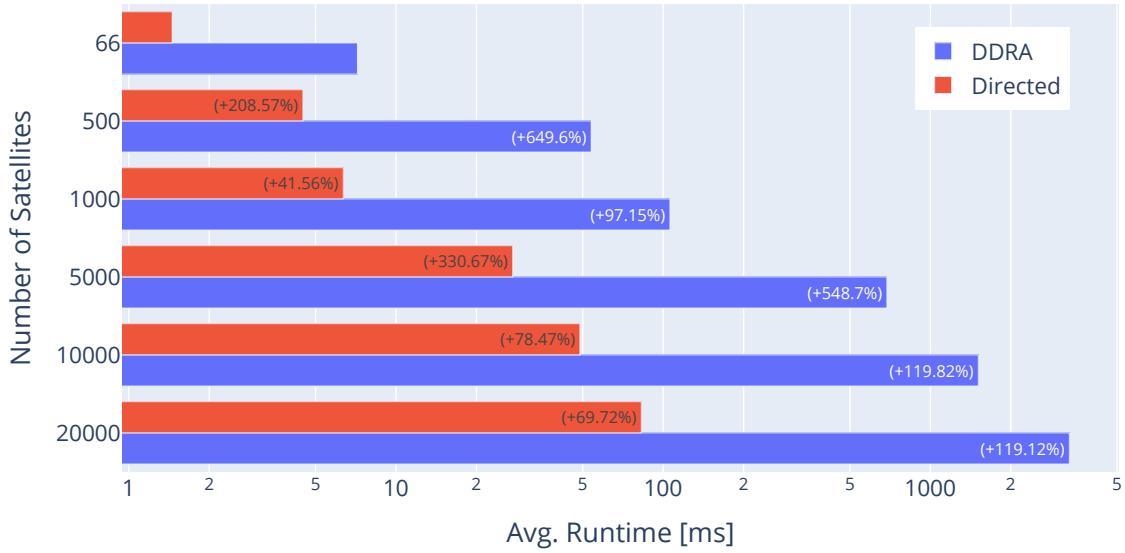


Figure 31: Average runtime for pathfinding between random pairs in different sized graphs.

DSPA is known to give optimal results when all edges have a positive weight, but on the other hand the expected runtime is really high for graphs with many nodes. Directed Routing tries to only discover nodes in direction of the destination which might not yield the optimal path due to flaws of the heuristic, such as not paying attention to the actual euclidean distance of the discovered nodes. Nevertheless, the number of explored nodes is smaller, thus the runtime is expected to be smaller. In fig. 31, the runtime for finding a path between a random pair of nodes within a graph of given size is given. It is important to note, that the Y-axis is scaled logarithmically. With the help of this graph we want to answer the associated evaluation questions:

How expensive are the algorithms? It can be seen, that Directed Routing outperforms DDRA by far and is much less expensive to calculate. Even in the smallest constellation DDRA is 5 times slower than its counterpart. The difference is most obvious in the largest constellation of 20000 satellites, where Directed Routing finishes in about 90 ms,

while DDRA takes over 3 seconds, about 33 times longer.

How scalable are the algorithms? Directed Routing scales better with an increasing number of satellites. With more than 5000 satellites, doubling the number of satellites using DDRA results in a relative increase in runtime of about 120 percent, while using Directed Routing the increase is only about 70 to 80 percent. Also, the baseline for Directed Routing is lower, which classifies the relative increases better and makes DDRA's numbers look even worse because the absolute increase is much higher.

Are the algorithms suitable for LEO mega constellations? This work assumes that the minimum number of satellites for LEO constellations to be classified as mega constellations must be greater than or equal to 10000 satellites. In a constellation of 10000 satellites, the average runtime for Directed Routing to compute a route for a random pair is about 50 ms, while DDRA completes in about 1500 ms. Therefore, DDRA is clearly not suitable for on-demand routing during operation, since the runtime is much higher than the maximum allowed packet latency of 500 ms. Only off-line calculations and storage of routes in forwarding tables allow the utilization of DDRA in such big constellations. Directed Routing, on the other hand, would be able to fulfill the latency requirements and would allow on-demand routing. Nevertheless, the runtime of 50 ms would contribute to the E2E latency and thus limit the realtime requirements and competitiveness with other network or constellation providers.

B. Memory Utilization

Directed Routing does not need to store topology slices, instead it is sufficient to store the number of planes, the number of satellites per plane and the id of the satellite. This results in 16bit + 16bit + 32bit = 64bit, as the minimum required memory capacity for each satellite, assuming that on-demand routing is desired and that a satellite can deduce which satellites it is connected to during operation.
DDRA has higher memory requirements because it is based on time slices. Each time slice represents the routing topology as a connection matrix valid for the entire duration of the slice, thus the maximum length is limited by the frequency of topology changes. On the other hand, it is possible to shorten the time slices to make the distances in the connection matrix more accurate and to reset failure states more often (see [5]). Hence, given an orbital period of T and a time slice length of S, the satellite must store N_{TS} time slices, calculated as $N_{TS} = T/S$ where T must be divisible by S. If there are N_S satellites in the constellation, the size of a time slice can be represented and stored in two ways. First, for each satellite, each satellite stores the identifiers of its four neighbors and their distance, assuming that all other distances are infinite (unreachable), resulting in a size of $N_S * (4 * 16\text{bit} + 4 * 32\text{bit}) = N_S * 192\text{bit}$. Or second, a 2-dimensional array where the value of row X and column Y represents the distance of satellite X to Y, which has a size of $N_S * N_S * 32\text{bit}$. It is not sufficient to store the forwarding table for each time slice, because DDRA needs to know the connection matrix to recompute routes in case of link/node failures and congestion. In addition, the satellite must be able to store flags for failed links and the congestion state of its four neighbors and itself, adding $5 * 4bit + 5\text{bit} = 25\text{bit}$.

How storage expensive are the algorithms? Assuming an orbital period of $T = 90\text{min}$, a time slice length of $S = 5\text{s}$, and a constellation of $N_S = 500$ satellites, for Directed Routing each satellite is required to store 64bit, while for DDRA each satellite is required to store (assuming the more space efficient representation) $25\text{bit} + 1080 * 500 * 192\text{bit} = 103680\text{kbit}$, with 1080 being the number of time slices required to represent the orbital period of 90min, each of 5s length. This representation will require DDRA to generate the connection matrix at the start of each time slice, which leads to an increase in runtime, but otherwise storing the whole matrix representation would take up $25\text{bit} + 1080 * 500 * 500 * 32\text{bit} = 8640\text{Mbit}$. This example shows that DDRA has higher memory requirements, which can be optimized but will increase the computational efforts of the algorithm.

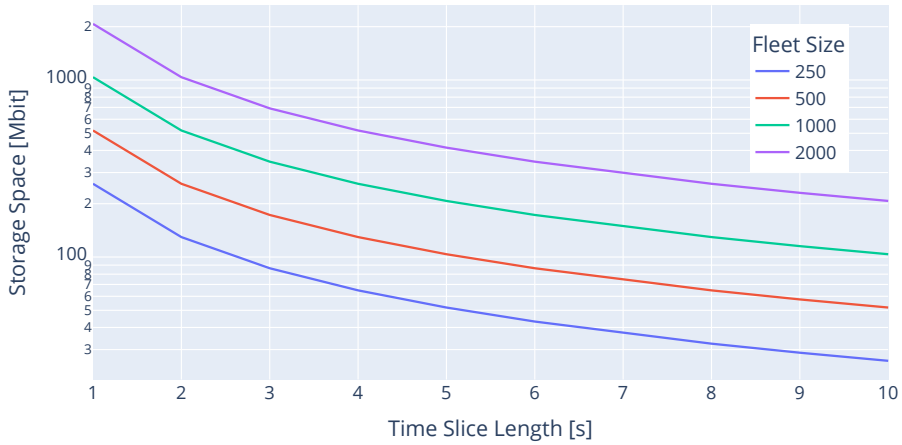


Figure 32: DDRA: Required storage space for time slice lengths on different fleet sizes.

Are storage requirements influenced by algorithm parameters? The storage requirements of Directed Routing are only influenced by constellation properties but not by algorithm parameters. In contrast, the requirements of DDRA are correlated with the selected time slice length, as shown in fig. 32, since halving the time slice length roughly doubles the storage requirements due to the doubled number of time slice representations that must be stored.

C. Algorithm Negotiation Computational Overhead

Directed Routing has no methods for detecting or handling either failures or congestion, so there is no computational overhead for negotiation and no further analysis of this algorithm is required. DDRA behaves differently, as this algorithm can react to detected failures and congestion of both itself and its immediate neighbors.

How often is the algorithm required to reroute? With DDRA, after a failed link or node is detected, the satellite deletes these edges from the currently loaded connection matrix and a recalculation of the routes is required. Furthermore, the satellite will report the detected failure to all of its still reachable neighbors, who will also delete the edge from their connection matrices. When a satellite is congested, the only difference is that the congested satellite itself does not have to recalculate the routes, only the neighbors have to remove the satellite from the connection matrix after receiving the feedback packet.

Is a full recalculation required? This depends on the implementation variant of DDRA. In an eager implementation, the satellite will calculate and store the next hop for all possible destination satellites as soon as possible, so that if a deleted edge or congested satellite becomes known, the satellite will immediately recalculate all possible routes. In a lazy implementation, the satellite clears the routing table after receiving a congestion or failure feedback and only recalculates a next hop for a given destination satellite when a packet with that destination arrives.

How much overhead is created? For the lazy variant, it is difficult to determine the actual overhead because it depends on the packets arriving during that time slice. Nevertheless, it can be stated that any incoming packet with a previously unseen destination satellite (in this time slice) will result in one execution of DSPA, whose runtime was discussed in **A** for different constellation sizes. The eager variant, executed on a constellation of N_S satellites, results in a single DSPA execution without early abort and $(N_S) - 1$ executions of path reconstructions to determine the next hop for all possible destination satellites on each affected satellite (except the congested satellite in the case of congestion handling).

6.2.5 Evaluation Summary

During regular operation scenarios, the used lazy DDRA implementation performs better than Directed Routing in terms of path finding with lower E2E delays, fewer hops, and smaller distances. In addition, the delays are on par with Directed Routing, with one outlier being is the queuing delay in large constellations. The error handling methodologies allow for an better response to failures with less packet loss for both small and large Walker-Star constellations by providing the ability to return to normal operation. Congestion handling, on the other hand, becomes a problem when traffic is high enough to congest several adjacent satellites or all first satellites. Then the packet loss rate becomes unreasonably high, and the result is a much better performance of Directed Routing, which does not implement any mechanism to handle congestion. Regarding algorithm efforts, it was found that DDRA comes with much longer run times than Directed Routing for all constellation sizes, which indicates poor scalability. Therefore an application in large constellations is unfeasible and in LEO mega constellations even impossible with run times of multiple seconds. Directed Routing on the other hand scales much better in terms of run times with average execution times below 100 ms for 20000 satellites.
In summary, neither Directed Routing nor DDRA are economically feasible in large constellations for the reasons stated above. In particular, growing traffic or user bases can have a similar effect on the constellation as a DDoS attack, as too much traffic can trigger the congestion handling problems described above. Furthermore, the maximum reasonable number of satellites is limited and below the number of upcoming mega constellations due to long routing run times. For example, in a constellation of 20000 satellites, it takes about three seconds from the time a congested satellite notification is received until the calculation of alternative routes is complete. Directed Routing, on the other hand, has no capabilities to handle failures and would require manual intervention and adding of exceptions to the pathfinding algorithm to compute routes that do not pass through failed links or nodes. This would require a permanent monitoring of the constellation, and once such a failure is found, it takes time until the changes are made. If ISLs or GSLs devices are very reliable, an implementation is conceivable, but with worse latencies, hop counts,

and distances than precomputed routing tables using DSPA would provide. Given a choice between the two algorithms, this work would suggest using DDRA without the algorithms proposed congestion control mechanism. The recalculations take a long time, but then the alternative routes provide less packet loss for the remaining time slice.

6.3 Walker-Delta: DDRA and DISCO

In the previous subsection a comparison between two algorithms suitable for Walker-Star constellations was performed. Here, the focus is shifted to a comparison of DDRA and DISCO running on Walker-Delta constellations.

6.3.1 Regular Operation

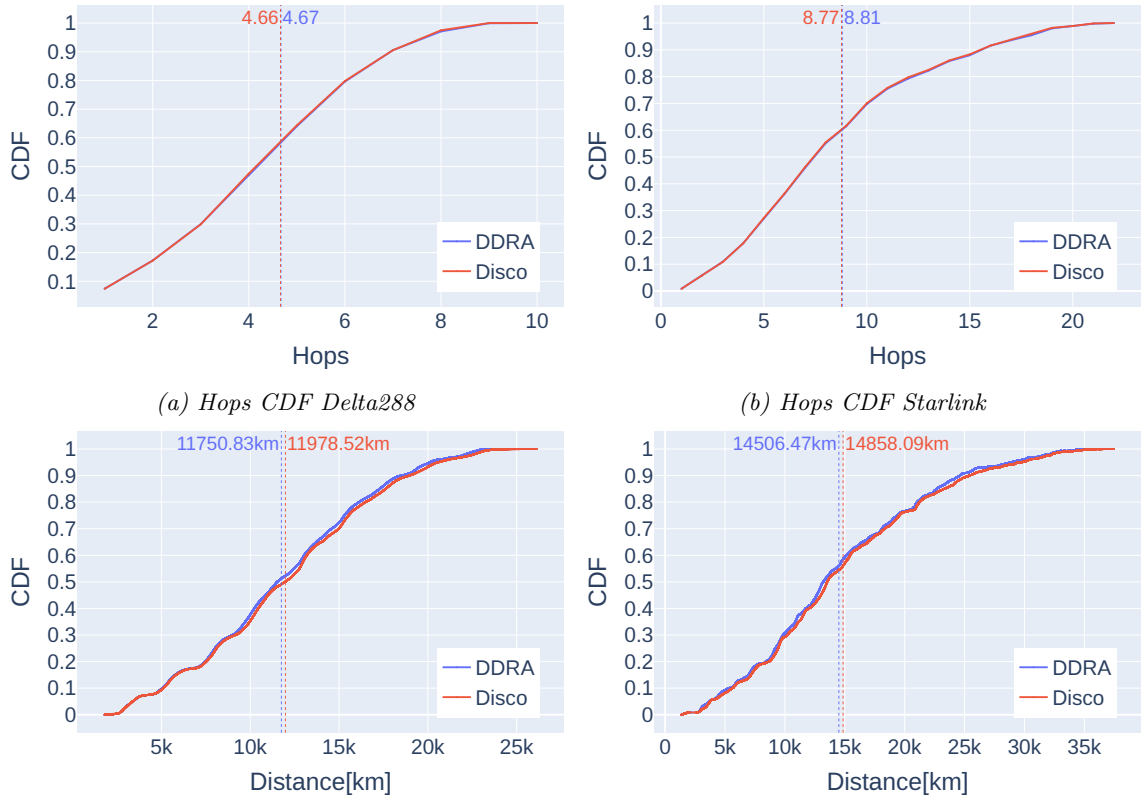
For investigating the regular operation, this benchmark was configured as shown in tab. 13. Most parameters are similar to Walker-Star except for the minimum elevation and maximum hops, which were adjusted due to the lower altitude and higher number of satellites.

Parameter	Common	Delta288	Starlink
Simulation time	6000s		
Random seeds	23543, 84242		
Data rate	50 Mbps		
Orbit base day	1.5, 2		
Maximum hops		75	120
Processing overhead	1ms		
Queue capacity	40 packets		
Maximum GSL per Sat./Gs.	40		
Minimum elevation		15°	30°

Table 13: Walker-Delta: Regular Operation Simulation Parameters.

A. How do generated routes differ? Fig. 33 shows the results of regular operation. DISCO and DDRA achieve almost the same average number of hops in Delta288 and Starlink, with small advantages for DISCO. Looking at the hop CDFs, it can be seen that on both constellations DSPA and DISCO have approximately the same function. In terms of distance, DDRA, which is based on the shortest path algorithm, is able to achieve slightly shorter average routes and is able to deliver a few extra packets for the same distance, even though the hop count CDFs are almost equal. This means that the route with the least number of hops is not always the shortest path, which is considered by DDRA but not by DISCO.

B. Are there noticeable problems with pathfinding? The packet loss and drop reasons reported in tab. 14 show that DISCO was able to deliver all but one packet, which was transmitted over a GSL channel that was disabled immediately after routing. DDRA, on the other hand, again exhibits a low average packet loss, similar to the findings in Walker-Star. The main reason for the majority of the dropped packets is due to the unavailability of a route. Observing fig. 34, it becomes evident that packet losses emerge regularly close to ground stations, suggesting possible issues with the first or last satellites. Since each ground station is continuously connected, all packets are expected to be routable. However, it appears that the congestion control mechanism is again overreacting, which suggests that the algorithm will run into the same problems discovered in the



(a) Hops CDF Delta288

(b) Hops CDF Starlink

(c) Distances CDF Delta288

(d) Distances CDF Starlink

Figure 33: Walker-Delta: Hops and Distance CDF for DDRA and Disco.

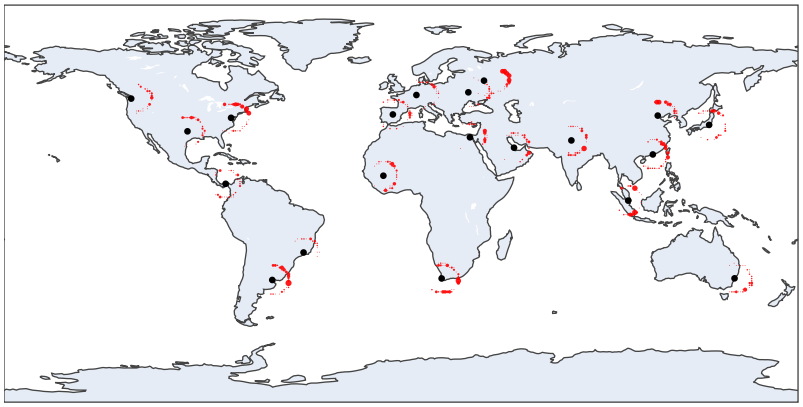


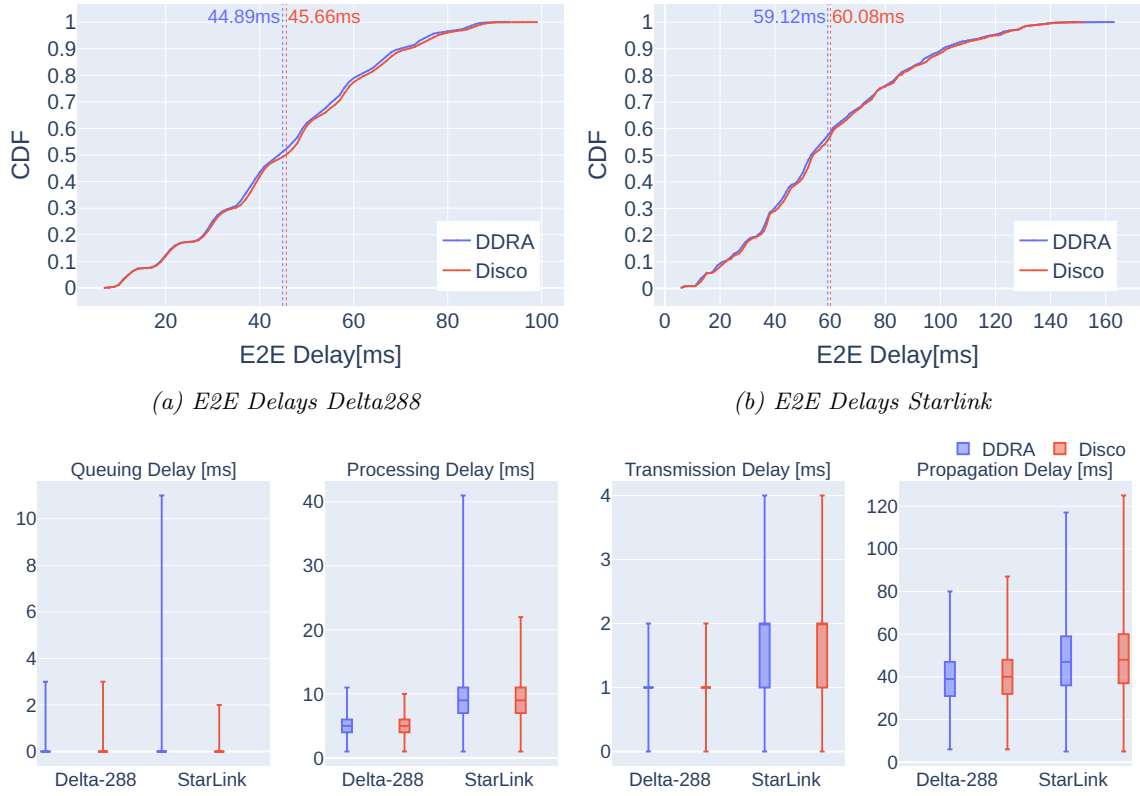
Figure 34: Walker-Delta: Packet drop locations of DDRA in Starlink.

congestion benchmark for Walker-Star (see 6.3.3), with very high packet loss and scalability issues stemming from this mechanism. The remaining packets were dropped for the same reason as the packet dropped with DISCO. For DDRA, the packet loss is low and still acceptable, although it is desirable to have no packet loss during regular operation.

C. How do the enabled packet delays differ? Fig. 35 (a) and (b) visualize the CDF for the E2E delay in Delta288 and Starlink, while (c) shows the contribution of the delay

	Delta288		Starlink	
	DDRA	DISCO	DDRA	DISCO
avg. packet loss	0.005%	0.0%	0.018%	0.0%
Packet drops	161	0	535	1
→ If. down	3	0	10	1
→ Unroutable	158	0	525	0
avg. E2E delay	44.89ms	45.66ms	59.12ms	60.08ms
Percentile				
→ 1% limit	86.0ms	87.0ms	135.0ms	136.0ms
→ 1% avg.	87.78ms	89.14ms	140.41ms	140.73ms
→ 0.1% limit	89.0ms	91.0ms	145.0ms	144.0ms
→ 0.1% avg.	90.13ms	93.80ms	150.076ms	147.05ms

Table 14: Walker-Delta: Regular operation numerical results.



(a) E2E Delays Delta288

(b) E2E Delays Starlink

(c) Delay category distribution

Figure 35: Walker-Delta: E2E delay and delay category contribution.

subtypes as box plots. In both constellations, DDRA is able to achieve slightly lower average E2E delays and is able to deliver a few more packets with the same delay. Regarding the delay subtypes, the median and the lower ($Q1$) and upper ($Q3$) quartiles are approximately equal for all subtypes with small lower values for DDRA in propagation delay. The propagation delay shows slightly longer whiskers for DISCO in all constellations. A higher whisker is the result of a larger maximum outlier in the range $[Q3, Q3 + 1.5 * IQR]$, where IQR is the interquartile range that is calculated as $Q3 - Q1$. Therefore, considering the

results of **A.**, the higher value is not surprising, since the routes of DISCO are longer on average. The more significant differences are in the upper whiskers for the queuing delay and the processing delay, where DDRA has higher values, especially in Starlink. This is due to the longer runtime to find a next hop using DSPA with a lazy implementation of DDRA than using DISCO, which is explored in 6.3.4. A longer processing time favors a longer queuing delay, because if a satellite is still processing a packet, another packet must be queued until the processing is finished. The reason that the quartile and median values are not noticeably affected by this higher runtime of DDRA is that the computed next hops are cached by the satellite for the rest of the time slice, resulting in many fast packet switches similar to DISCO as opposed to a small number of slow ones.

6.3.2 Failure Response

For the failure response comparison, we will keep the probabilities for link and node failures and recoveries, as well as the simulation time from the Walker-Star comparison. All other parameters are configured similarly to the regular operation scenarios, and an overview is shown below in tab. 13.

Parameter	Common	Delta288	Starlink
Simulation time	3000s		
Random seeds	23543, 84242		
Data rate	50 Mbps		
Orbit base day	1.5, 2		
Maximum hops		75	120
Processing overhead	1ms		
Queue capacity	40 packets		
Maximum GSL per Sat./Gs.	40		
Minimum elevation		15°	30°
Failure seeds	43236, 62276		
Prob. failure	0.01, 0.005, 0.0025		
Prob. recovery	0.6		

Table 15: Walker-Delta: Failure Response Simulation Parameters.

A. How capable are the algorithms in handling failures? Looking at fig. 36, it can be seen that DDRA is able to slightly reduce the packet loss rate compared to DISCO across all scenarios on Starlink, with an average reduction of 4.2%. In contrast, the packet loss rates on Delta288 remain unchanged in the low probability scenario, while the medium scenario shows an even higher packet loss rate for DDRA. Only the high probability scenario exhibits a smaller packet loss rate for DDRA. On average, when comparing the raw number of dropped packets, this actually results in a slight increase in packet loss rather than a decrease. As discussed in 6.2.2, this comparison provides only an approximation of the differences in failure handling, since both algorithms may compute different routes, resulting in potentially unequal numbers of packets affected by link or node failures due to the random distribution of failures in the constellation. This study attempted to minimize this effect by using multiple random seeds for the failure scenario generation and multiple base days for the satellite orbits. However, the influence cannot be completely eliminated.

(a) Packetloss Delta288

(b) Packetloss Starlink

Figure 36: Walker-Delta: Packet loss comparison for failure scenarios.

B. Are the algorithms able to recognize recoveries? Looking again at the packet loss rate after 2500s into the simulation, it can be seen that the packet loss rate for both algorithms and constellations decreases towards 0% when all link and node failures are resolved. Furthermore, since link and node failures may recover during the transition from one phase to the next for each of the 30 phases, in which the scenario between 100s and 2500s is divided, the algorithms are either unable to recognize failures in the first place, or must be able to recognize recoveries. If they were unable to recognize recoveries, the shown packet loss rates would be inconceivable, since the effects of removing links or nodes from the routing topology without recovering some of them would result in a fragmented and potentially unconnected graph and an associated high number of packets dropped due to unroutability or expiry because satellites are either unable to find a route or packets will loop between a set of satellites until the TTL reaches 0.

C. Does the failure handling enable a still acceptable system performance? DISCO has no failure handling capabilities, and without them it cannot achieve a packet loss rate below 5% except for the low failure scenario in Delta288. The explanation for DISCO being able to achieve a lower packet loss rate for the medium failure scenario in Delta288 is that the algorithm distributes the hops of the route in such a way that it passes through links that are not affected by failures. However, this is purely random and not the result of an algorithm mechanism. Similar to DISCO, DDRA is only able to achieve a rate below the threshold in the low failure scenario of Delta288. With the exception of

the Delta288 medium failure scenario, the algorithm successfully reduces packet loss rates compared to DISCO, but the gains are minimal and not high enough to reduce packet loss in any scenario below the threshold.

6.3.3 Congestion Response

The final algorithmic capability is congestion control, which will be examined here. Again, similar to Walker-Star, the maximum queue size is reduced to increase the need for congestion handling, and three scenarios are created for each sending interval using a 0.03s+1s *burst+idle* pattern. Other simulation parameters are set to those shown in tab. 15.

Parameter	Common	Delta288	Starlink
Simulation time	1500s		
Random seeds	23543, 84242		
Data rate	50 Mbps		
Orbit base day	1.5, 2		
Maximum hops		75	120
Processing overhead	1ms		
Queue capacity	25 packets		
Maximum GSL per Sat./Gs.	40		
Minimum elevation		15°	30°
Burst+Idle Duration	0.03s + 1s		
Sending Interval	1ms, 0.5ms, 0.25ms		

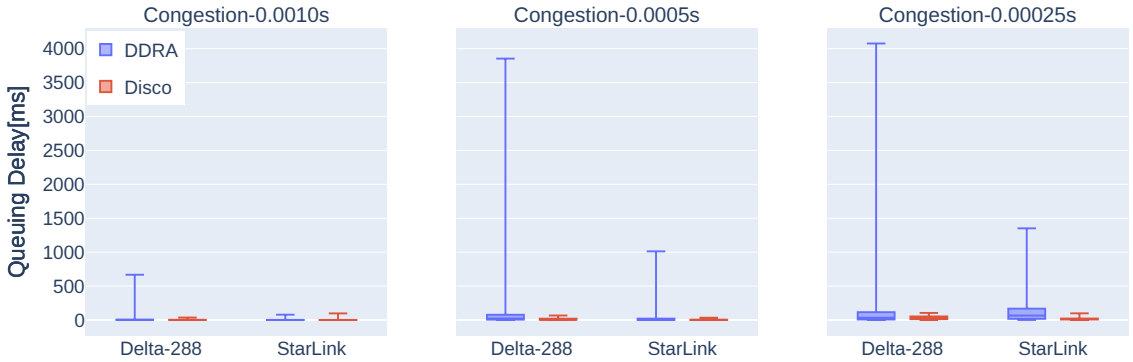
Table 16: Walker-Delta: Congestion Response Simulation Parameters.

A. How capable are the algorithms in handling congestion? Looking at the queuing delays shown in (a) fig. 37, it is clear that DISCO outperforms DDRA with lower quartiles and much smaller whiskers. Since DISCO does not come with a congestion control mechanism, this shows that DDRA's mechanism not only does not reduce the congestion that leads to longer queue times, but actually aggravates the observed delays. Except on Starlink in the lowest traffic scenario, some of the recorded queuing delays of DDRA exceed the upper limit of 500ms of the maximum acceptable E2E delay by far, and this does not include other delay subtypes such as propagation delay, which can add another 100ms. More critically, the packet loss rates are also much higher than DISCO for each scenario on both constellations. Looking at the exact numbers shown in tab. 17 for Delta288 DDRA is able to reduce the number of dropped packets due to full queues from $557,601$ to $308,658$, but therefore creates an unacceptably high number of drops due to expiration and non-routable which cannot be found with DISCO. On Starlink, the situation is worse, as there are $75,853$ packets dropped due to full queues, compared to $46,896$. There are also an additional $551,624$ packets dropped due to expiration, and $1,725,741$ packets that are unroutable.

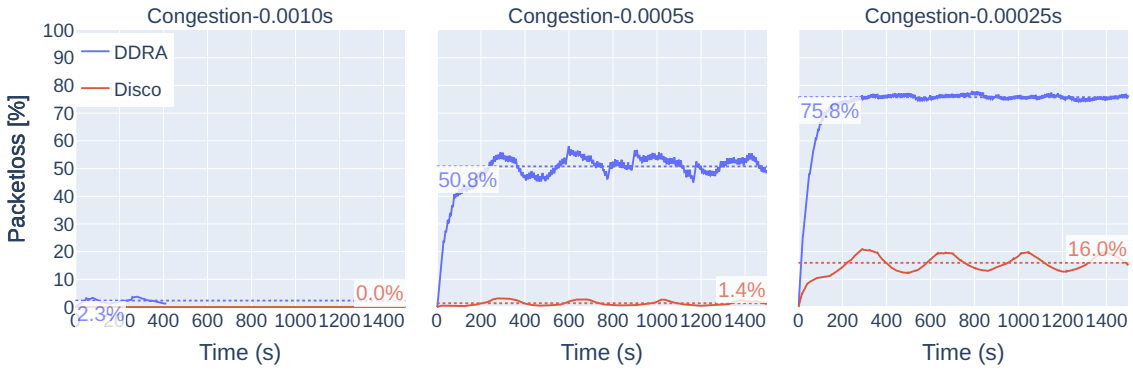
B. Which drawbacks are created by the mechanisms? Similar to Walker-Star, the mechanisms of DDRA lead to a massive increase in queuing delay and packet loss. Once again, it is evident that burst traffic causes overflow in the first and last satellites, resulting in a fragmented routing topology. This leads to either packets being sent in loops until the TTL expires or being dropped directly due to the satellite's inability to compute an alternative route. When comparing the difference in packet loss between Delta288 and Starlink, it appears that this found effect is stronger with fewer first or last satellites.

	Delta288		Starlink	
	DDRA	DISCO	DDRA	DISCO
avg. packet loss	75.8%	16.0%	57.1%	1.3%
Packet drops	3,023,044	557,601	2,353,218	46,896
→ Expired	252,880	0	551,624	0
→ Full Queue	308,658	557,601	75,853	46,896
→ Unroutable	2,461,506	0	1,725,741	0

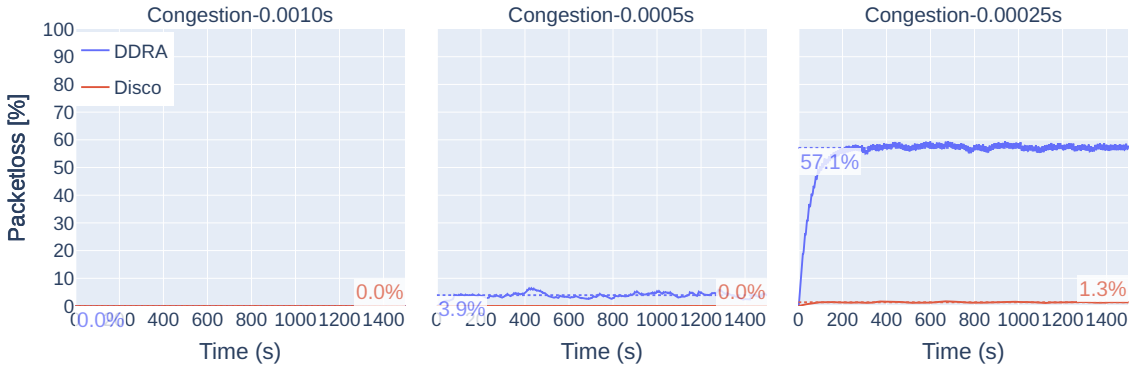
Table 17: Walker-Delta: Congestion-0.00025s scenario numerical results.



(a) Queuing Delays



(b) Packetloss Delta288



(c) Packetloss Starlink

Figure 37: Walker-Delta: Queuing delay boxplots for Delta288 and Starlink.

6.3.4 Algorithm Efforts

A. Computational Efforts

In contrast to DDRA, which operates on a graph representing the current routing topology, DISCO utilizes constellation properties for routing decisions based on a calculated minimal hop count without the need for constructing a connection matrix or graph. This method was proposed with scalability and low runtime in mind, which will be investigated compared to DDRA in the following.

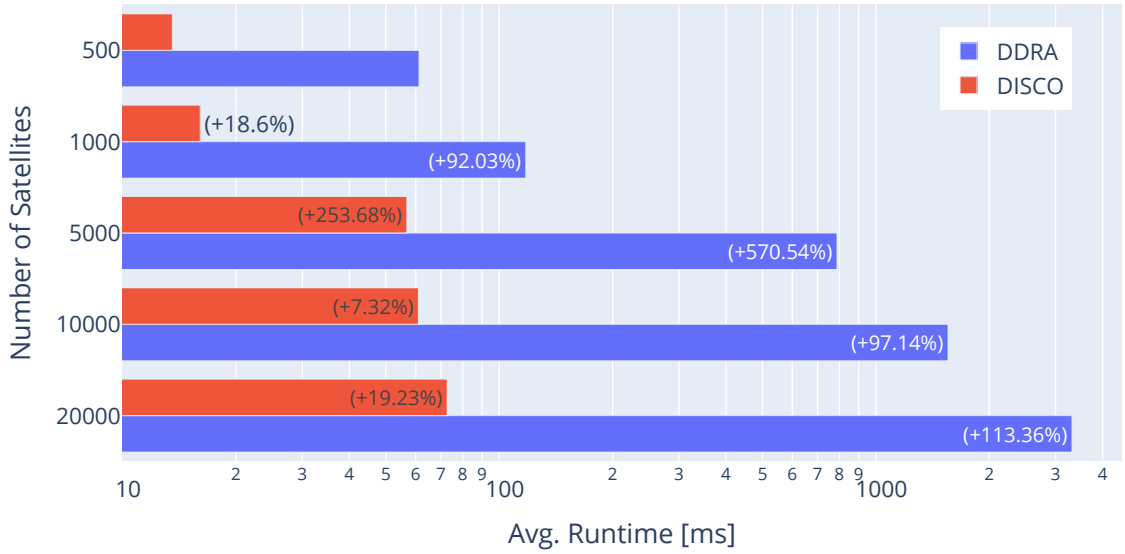


Figure 38: Walker-Delta: Average runtime for pathfinding between random pairs in different sized graphs.

How expensive are the algorithms? Fig. 38 shows the average runtime of finding a single route between a random pair for DISCO and DDRA in Walker-Delta constellations with different numbers of satellites on a logarithmic scale. It can be seen that DISCO terminates much faster than DDRA in all constellation sizes. For smaller constellation sizes, DISCO is even fast enough not to need to cache of routes and compute them on packet arrival, which is not possible with DDRA without causing long processing and subsequent queuing delays. From this it is can be concluded that DISCO offers better runtime performance characteristics and is the less expensive algorithm.

How scalable are the algorithms? The runtime baseline given by the time for routing decisions in the smallest constellation of 500 satellites is much faster for DISCO with 15ms vs 60ms. In addition, the subsequent relative increases in runtime per step in constellation size are smaller. The runtime remains below 100ms even for the largest constellation, while DDRA already exceeds this threshold with slightly less than 1000 satellites and ends up with a runtime of more than 3s for 20000 satellites. Another look at fig. 38 shows that when using DDRA, a doubling of the number of satellites on average leads to about a doubling of the runtime with an increasing tendency, while DISCO leads to an increase of about 20%, again with an increasing tendency. Considering the base and the increase in runtime together, it can be concluded that DDRA offers worse scalability than DISCO.

Are the algorithms suitable for LEO mega constellations? Again, this evaluation uses the lower bound for mega constellations of 10000 satellites mentioned above (see 6.2.3). For these constellations, the corresponding minimum average routing runtimes are about 1500ms with DDRA and about 60ms with DISCO, which is a difference of 25x. While both algorithms could compute routing tables prior to operation, DDRA will run into problems if rerouting becomes necessary during operation due to failures or congestion. If it takes about 1500ms for alternative routes to become available after learning of a congested satellite or failed link, the congestion control and fault recovery mechanisms are delayed and their effectiveness is reduced or even non-existent. For example, a congested satellite will continue to receive traffic for the next 1500ms, which may cause the queue to overflow, and the satellite may be in the non-congested state by the time the new routing tables take effect, resulting in subsequent reroutes. The 60ms of DISCO are feasible even for calculations during operation, in addition there are no reactive mechanisms which could be influenced by this duration.

B. Memory Utilization

The storage capacity requirements of DDRA are unchanged from those of Walker-Star, the only difference being that longer time slices are possible because the topology remains the same, since only the distances of the inter-plane hops change. A constellation could leverage DISCO in two ways, first by dividing the orbital period into time slices, pre-computing and storing the next hop to each destination, or second by running the algorithm on the fly. In the first case, assuming that the orbital period T of a constellation with N_S satellites is divided into slices of length S, the required storage capacity for each satellite can be calculated as $(T/S) * N_S * (16\text{bit} + 2\text{bit})$, where 16bit are reserved for the identifier and 2bit for the next ISL direction. In the second case, the satellite must store the general constellation parameters such as RAAN difference, phase difference, and phase offset in order to perform DISCO during operation. In addition, the initial mean anomaly and RAAN for each satellite must be known, and the current latitude must be computable, which can be done by determining the orbit using the already known parameters along with the inclination and eccentricity of the constellation. This results in a memory requirement of $5 * 16\text{bit}$ for the constellation parameters and $N_S * (16\text{bit} + 16\text{bit})$ for the initial orbital parameters per satellite.

How storage expensive are the algorithms? Assuming an orbital period of $T = 90$ min, a time slice length of $S = 5$s, and a constellation with $N_S = 500$ satellites, the required storage for DDRA remains 103.68 Mbit. The first variant for DISCO requires, by also using S as time slice length, $1080 * 500 * 18$ bit = 9.72 Mbit, while the second variant requires 80 bit + $500 * 32$ bit = 16.08 kbit. Comparing these values, it can be seen that even the more storage efficient variant of DDRA requires at least 10 times the storage of DISCO for the given setting, and if the second variant of DISCO is used, which has higher runtime requirements, the difference increases further to about 6448 times.

Are storage requirements influenced by algorithm parameters? Again, the effect of the algorithm parameters on DDRA is unchanged. In the first variant, the storage requirement of DISCO also depends on the time slice length like DDRA, but the data to be stored for each time slice is smaller by a factor of about 10. The second variant, on the other hand, does not depend on algorithm parameters.

C. Algorithm Negotiation Computational Overhead

Similar to Directed Routing in Walker-Star, DISCO does not come with reactive methods, so there is no computational overhead for negotiation and no further analysis of this algorithm is required. An analysis of DDRA is not necessary due to the unchanged behavior compared to Walker-Star, which was already discussed in **C** of 6.2.4.

6.3.5 Evaluation Summary

During regular operation, DDRA and DISCO deliver almost the same performance, with small advantages for DDRA in terms of enabled E2E delays, but at the cost of an existing low packet loss rate. The error handling technique of DDRA works, but the obtained gains are minimal when compared to DISCO, and almost negligible because the packet loss rate is not reduced enough to allow for an acceptable system performance.
Similar to Walker-Star, the problems found with the congestion control mechanism are also present with DDRA in Walker-Delta. DISCO, without any implemented mechanism for handling congestion, is able to handle much more traffic without exceeding acceptable packet loss rates or incurring high queuing delays, making the algorithm much more scalable in terms of traffic and user base.
The runtimes of the two algorithms are very different. While DDRA's runtime is unchanged between Walker-Star and Walker-Delta, DISCO can make use of its really scalable and fast execution time on Walker-Delta constellations with average runtimes of 70 ms in mega constellations with 20000 satellites. In terms of memory, neither algorithm requires unacceptably large amounts of memory if the time slice length is not too small. Since the topology remains constant in Walker-Delta, the length of the time slices can be increased to reduce memory usage in exchange for some accuracy of the shortest routes if needed.

Again, DDRA in this form, with this proposed congestion control mechanism, is neither practical nor economical. In terms of economic efficiency, DISCO performs better than DDRA. DISCO can be used with a large number of satellites due to its excellent runtime scalability and small memory footprint, while allowing for small E2E delays and routes whose length is close to the shortest possible route. In addition, this algorithm is fast enough to handle high traffic volumes by causing less congestion than DDRA. DDRA, on the other hand, has weak runtime scalability and weak scalability of supported user and traffic capacity, because similar to the findings of Walker-Star, a growing user base or traffic demand can affect the constellation in a way comparable to a DDoS attack. This effect can cause unacceptably high delays and packet loss rates, resulting in poor system performance and a poor user experience.
Thus, if a choice must be made between the two algorithms, DISCO is the better choice, since DDRA's failure and congestion handling mechanisms either do not lead to much improvement or even degrade performance. However, if DDRA is still the preferred solution, it should be implemented without the proposed performance-threatening congestion control mechanism.

7 Discussion

7.1 Research Questions

The objective of this work is to contribute to the ongoing research on the performance evaluation of state-of-the-art LEO routing algorithms that can be used with single-shell Walker constellations by proposing applicable benchmark scenarios and developing an extensible simulation platform. To demonstrate the comprehensiveness of the benchmarks and the capabilities of the developed platform, a comparison between two Walker-Star and two Walker-Delta applicable algorithms was performed for a small and a large constellation, respectively. The main research question was:

RQ Is it possible to develop a general applicable methodology that allows the research of various LEO routing algorithms for single-shell Walker constellations by enabling a comprehensive and reproducible comparison of their performance?

To answer this question, the research question was divided into three smaller research questions, the answers to which are summarized below.

RQ1 What are suitable general applicable benchmarks which allow a representative exploration of features and characteristics of LEO routing algorithms?

Through a literature review, several scenarios and evaluation questions were found and combined with own contributions to create a set of five comprehensive benchmark scenarios. First, a parameter study was proposed to test the influence of constellation parameters on algorithm performance. Then, by selecting one small and one large constellation per Walker type, the algorithms are exposed to the remaining benchmarks. The benchmarks test, first, the day-to-day performance of the algorithms, second, the error handling capabilities by running against multiple link and node failure scenarios with different failure probabilities, third, the congestion handling by applying *burst+idle* patterns with different transmission intervals that cause an increasing number of queued packets, and, finally, the overhead of the algorithms in terms of runtime, memory requirements, and protocol computation negotiation overhead.

RQ2 How can a simulation tool be developed that allows the dynamic creation of Walker-Star and Walker-Delta satellite constellations, the addition of LEO routing algorithms, and the exploration and evaluation of their performance using simulation and visualization techniques?

This platform was created by extending the omnetpp-based FLoRaSat framework. We added support for creating arbitrary single-shell constellations according to given Walker parameters, the dynamic set up and tear down of ISLs and GSL according to constraints such as maximum latitude and elevation. In addition, packet and satellite statistics extraction capabilities have been added, as well as an extensible API that allows arbitrary LEO routing algorithms to be added to the platform using multiple exposed callbacks. Finally, a CLI was developed to automatically process the extracted metrics into visualizations.

RQ3 Can the developed platform be used to perform a comprehensive comparison of the performance of three representative LEO routing algorithms by leveraging the established benchmark scenarios for both Walker-Delta and Walker-Star constellations?

Using the proposed generic benchmark scenarios, this work has demonstrated the capabilities of the developed simulation platform by performing a comparison between the algorithms DDRA and Directed Routing for Walker-Star, as well as DDRA and DISCO for Walker-Delta. This was done by using the extensive metrics collection of the simulator as input for the created FLoRaSat CLI, allowing the automatic generation of visualizations used for an evaluation and final conclusion about the algorithms performance, economic operating costs and suitability for deployment in tomorrow's upcoming LEO mega constellations. During the comparison, we identified problems with the congestion control mechanism utilized by DDRA under conditions of high traffic. Excluding entire nodes appears ineffective for handling congestion due to restricted routing topology to reach certain destination nodes and GSs. We also found that DISCO is far ahead of DSPA-based routing in terms of runtime and scalability.

7.2 Findings and Contributions

A list of findings in related work is presented below, along with an explanation of how this work can contribute to those findings.

1. Difficult comparability. Related work, in particular algorithm proposals, do not use comparable benchmark scenarios and evaluation questions and create one-time simulation platforms that are developed specifically for the evaluation of this algorithm. The benchmarks proposed in this work and the simulation platform developed allow for a more comparable evaluation since the constellation, satellite positions, and available ISL and GSL are created and computed consistently and the same comprehensive set of evaluation questions is assumed to be answered.

2. Missing selection of first and last satellites. Some algorithm proposals seem to miss an important part of routing in LEO, which is the selection of the first and last satellites during routing. The selection of an optimal first and last satellite is a non-trivial routing task and can strongly influence the performance of the constellation by reducing required hops or distribute congestion. By enabling researchers to implement their own methodology for calculating the first and last satellites, this working platform points them to this issue and requires them to think about it, or at least be aware of it, when using the default DSPA-based variant provided.

3. Repeating simulator and visualization creation effort. Creating simulators with metric extraction capabilities and subsequent visualization techniques is a non-trivial task and takes a long time, which slows down the ongoing research. The developed simulation platform replaces the necessity of manual simulator creation during the research of novel algorithm proposals by adding an algorithm implementation to the already existing platform. Therefore, the main contribution of this work with the help of CLI for the subsequent metric analysis is to enable researchers to dedicate their time to improving their novel algorithms instead of spending it on implementing a test and evaluation environment. This will be beneficial for future related work.

4. Malfunctioning reactive methods. The congestion control issues discovered in DDRA due to the exclusion of certain nodes from the routing topology demonstrate that not every reactive approach is also an effective method to enhance the system performance. DISCO delivered good performance even under significantly higher traffic compared to DDRA, despite not utilizing any approach for handling congestion. It is possible that DDRA may have achieved even better results by not implementing the congestion control mechanism at all. On the other hand, while the error handling employed in DDRA is promising, its state resets too frequently. If the state would not reset at each new time slice, there would not be the possibility of three new packet being discarded and subsequent wait on the non-arriving acknowledgements for every broken satellite link.
This shows, that the developed platform can identify malfunctions or areas for improvement, which demonstrates our success in achieving our research objective of contributing to the ongoing study of LEO routing algorithms.

5. Small constellations are not future-proof. It has been shown that on larger constellations, congestion has much less impact on performance. This implies that large constellations will be better equipped to handle future increases in user-bases or traffic. The higher number of first and last satellites results in better distribution of ground traffic across the constellation, which is not the case with Iridium and similar satellite systems where only two or three satellites cross densely populated areas such as Europe at the same time.
The methodology of this work permits the exploration of the impact on constellations of varying sizes, and enables improved economic planning and modeling of forthcoming constellations that are currently in the design phase.

6. Satellite removal mechanisms are inefficient. The results for the congestion control mechanism of DDRA indicate that the method of removing satellites from neighbor routing tables, even if they are still reachable, can degrade performance, especially in smaller constellations. This occurs because GSs usually have contact with only a limited number of satellites, which serve as the first and last satellites for packets originating from and destined for these GSs. If all connected satellites of a GS are removed from the routing tables of their respective neighbors, there will no longer be a valid path from and to this station resulting in packet drops, leading to high packet loss rates.

7.3 Limitations

1. Required Omnetpp and C++ knowledge. This work provides a user-friendly and extensible API upon which new algorithmic additions can be based. Nevertheless, knowledge of omnetpp is essential for basic processes such as adding the new routing module to the simulation kernel, writing and understanding the domain-specific language used in configuration files, or understanding and using the lifecycle hooks called by the kernel. Additionally, having a comprehensive understanding of essential C++ features such as pointers, data structures, is crucial to avoid disrupting simulations. Moreover, it will inevitably be necessary to debug a running simulation using debuggers such as gdb to understand the multi-module structure and execution flow, which further highlights the necessity for more than a beginner's level of C++ knowledge, since the omnetpp base does not make use of expressive error messages.

2. Simulation speed and algorithm runtime correlation. Omnetpp is a DES that uses sequential execution of scheduled events. This works well unless the execution time of individual events becomes very high, which results in a freezing simulation time until all events scheduled for a particular simulation time have completed. Considering the poor scalability of algorithms such as DSPA for large numbers of satellites, this will have a negative impact on the simulation time experienced. For example, if an algorithm requires frequent recalculation of routing tables on each satellite, an event is scheduled and executed for each satellite, which can take several seconds and even minutes in mega-constellations if these events are executed sequentially.
Multithreading could be a solution to this problem, but due to the single-threaded nature of omnetpp, it was not designed to support parallelism. As a result, fundamental modules that are essential to FLoRaSat lack required methodologies such as locking or atomic state. Fig. 39 illustrates the simulation speed in terms of simulated seconds per second of runtime for DDRA using growing constellation sizes. It is evident that the simulation speed decreases heavily with the increase in constellation sizes, and falls below one simulated second per second of runtime with 1400 satellites. For instance, simulating one hour would take: 3600 simsec / 0.847 simsec/$s \approx 4252s$.

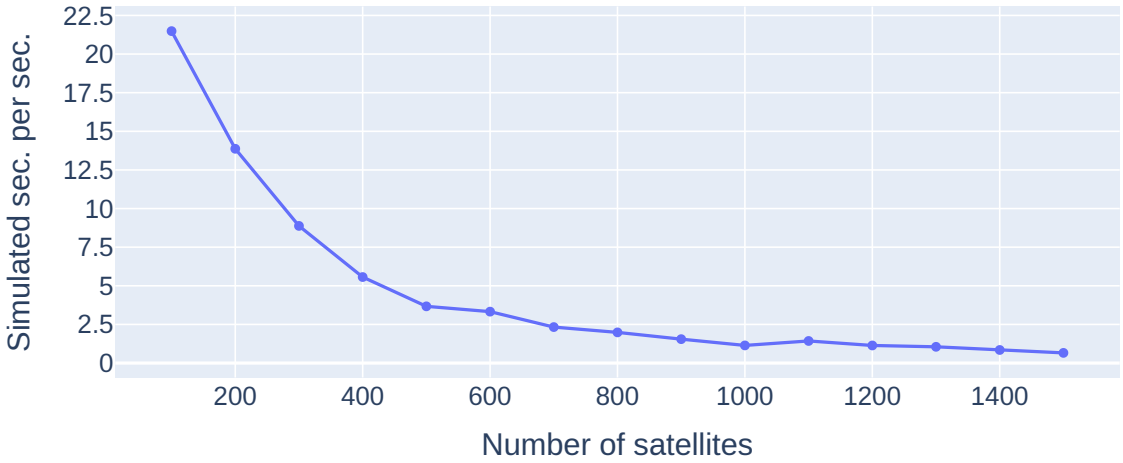


Figure 39: DDRA simulation speed on different constellation sizes.

3. Execution system requirements. The extensive collection of statistics, including metrics for each packet, the recording of each hop of a packet's route, and the state changes for each satellite, allows for detailed analysis of the algorithm's behavior. However, this has the side effect that the files that store these records grow rapidly in size on the executing system's file system. Depending on the algorithm used and the distribution of ground stations, a single simulation run of 18000s may require more than 3 GB of storage capacity. For instance, a vast parameter study can demand hundreds of GBs of available disk space.

As mentioned above, omnetpp does not support multithreading during simulations, but it does support parallel execution of multiple simulation runs. Since each satellite is represented by a module consisting of several submodules, including the routing module, the main memory requirements scale with the number of satellites and the memory requirements for the particular algorithm used. Therefore, the number of possible parallel

executions is limited by the available memory (RAM) of the executing system. In our tests it was impossible to run more than two simulations in parallel using DDRA and Starlink with 32 GB of RAM, while for smaller constellations it is possible to use all six available CPU cores without problems.

4. Deviations from authors intents The compared implementations of DDRA and DISCO were created by us based on our understanding of the published papers, since we had no access to the original authors' source code. This can lead to misunderstandings about certain implementation details, which can lead to much better or, of course, worse results than the actual intended implementation would deliver. In this work, we have tried to implement the proposals in the best possible way and to optimize as much as possible without intentionally deviating from the proposed methodologies. However, this factor cannot be ruled out and should be kept in mind when looking at the comparisons. Especially, the problems with DDRA's congestion control could be based on different queuing behavior assumptions.

7.4 Future Work

The above discussed limitations and impressions gathered while reviewing literature and working on this thesis spur the following future work.

1. Paralellization of CPU bound calculations. One aforementioned limitation is the correlation of algorithm execution speed to the simulation speed. These long running calculations must be executed sequentially due to omnetpp's lack of multithreading support. One potential future work is to find ways to work around this lack of support by combining multiple events into one event which executes slices in parallel. We forsee a rewrite of fundamental modules and the addition of state guarded by mutex and rebuilding of some class based singleton utilities to callbacks, which then allow an usage in parallel. Another idea is to outsource CPU bound calculations by calling libraries, that could be written in other languages that allow a more intuitive approach to multithreading, with the current state as arguments, await the result and after resolving apply the state changes to FLoRaSat.

2. Comprehensive algorithm survey. In sec. 3.1, this work already mentioned that found surveys of LEO algorithms for continuous connectivity are merely a listing and description missing a comparison and classification of their performance. Using the experimental pipeline presented in this work, it would be possible to create a comprehensive overview that includes a comparable performance index, which resolves the problem of different evaluation approaches and poor comparability.

3. Combining DDRA and Directed Routing. When comparing the two algorithms, it was found that both operate on a currently known graph representation and thus have a comparable approach to path finding, while Directed Routing has better scalability due to its utilized heuristics. By replacing DDRA's approach to pathfinding with that of Directed Routing and discarding the faulty congestion control mechanism, a hybrid algorithm could be developed that is much more suitable for actual deployments due to the ability to handle failures and able to be deployed on very large constellations. This would also require the development of a heuristic that is not tailored to Walker-Star constellations, but could

also work with Walker-Delta constellations.

4. Multi-shell constellation support. Currently the simulator only supports the dynamic creation of constellations with a single-shell and inter-plane and intra-plane ISLs. Upcoming and already existing mega constellations such as Starlink, have the approval of their respective supervisory authority for deploying satellites in multiple orbital shells. By extending the simulator to support the creation of multiple shells with different altitudes and the dynamic establishment and teardown of inter-shell satellite links, a whole new research area would be supported by FLoRaSat and therefore be able to leverage the created capabilities of this work.

5. Satellites as traffic source. In the current version of the simulator, traffic is generated exclusively on the ground stations, except for the negotiation packets generated by the algorithm. Omitting the simulation of ground stations could also be used to speed up the simulation, since GSLs do not need to be created, deleted, and updated frequently. E.g., imagine a constellation with 1500 satellites and 20 ground stations. This simulation requires $1500 * 20$ elevation calculations, as well as with distance calculations for all pairs that have elevations higher than the minimum elevation, on each scheduled update. Currently, no support for this behavior is implemented, but a modification to the architecture, as shown in fig. 40, could be implemented that would allow traffic patterns to be used on the satellite itself.

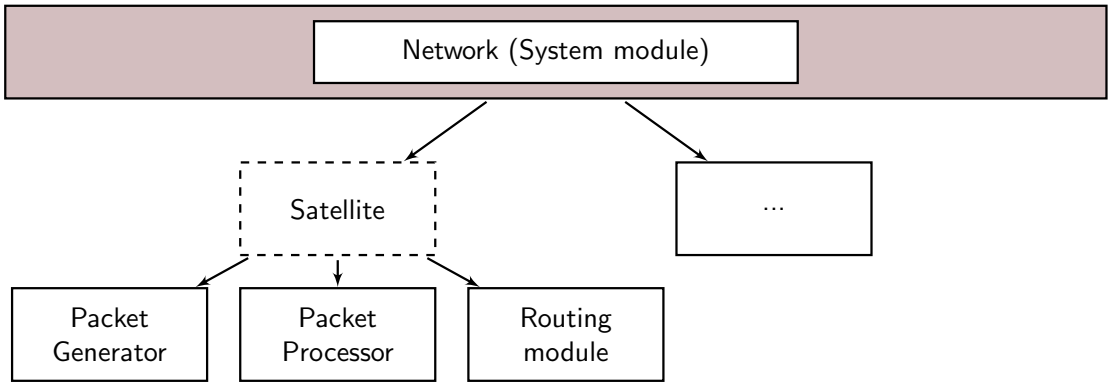


Figure 40: FLoRaSat architecture for on satellite traffic.

This change would only require an adaptation of the packet generator, which would have to forward the generated packets internally in the satellite to the packet processor instead of forwarding them to the satellite via GSL. The mechanism for selecting the first and last satellite is not required for this approach of traffic generation, another point that could lead to simulation speed up.

A Appendix

A.1 Adding new algorithms to FLoRaSat

The following part is not directly related to the methodology of this work, but can be used as a more detailed step-by-step introduction to using the platform for future research on novel algorithms. To this end, this subsection will explain the addition of a simple algorithm called "RandomRouting", which works as follows. Upon arrival of a packet, the algorithm first checks if the satellite has an active connection to the destination ground station, if so, the packet is forwarded to the destination, otherwise the packet is forwarded to a random neighboring satellite that the forwarding satellite is connected to.

A.1.1 Algorithm Implementation

1. File creation. For the addition, the first step is to add the required files to the simulator which will the implement the novel routing module. These files are "RandomRouting.ned", "RandomRouting.h", "RandomRouting.cc", which are placed in their own directory "randomrouting" as shown below.

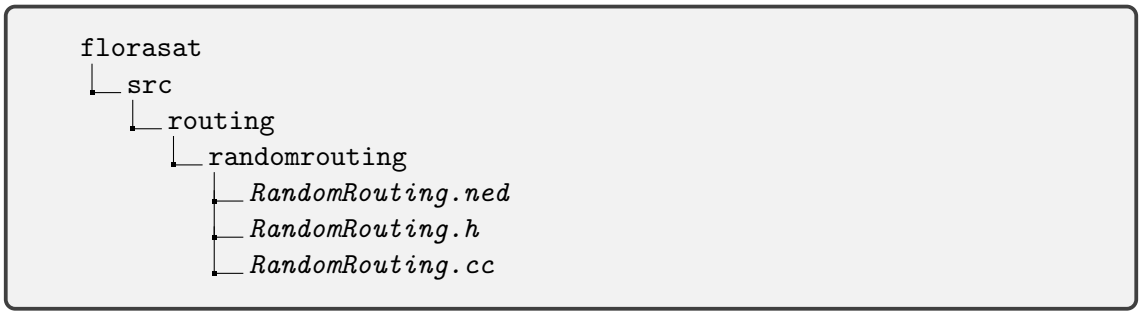


Figure 41: Directory and file tree for novel algorithm files.

2. Module declaration. The next step is to define the new module inside "RandomRouting.ned", the contents of which are shown below. Here, a simple module with name "RandomRouting" that extends "RoutingBase" is declared. Since this module has no submodules, it can be created as a simple module. Extending "RoutingBase", allows FLoRaSat to replace the abstract routing module of the satellites with this implementation during simulations (see 5.5.2 and 5.5.4). The file configures that the implementation of the class is located in a header file with the path "routing::RandomRouting".

```
1  package flora.routing.random;           // namespace of module
2
3  import flora.routing.RoutingBase;
4
5  simple RandomRouting extends RoutingBase {
6      parameters:
7          @class(routing::RandomRouting);  // path to class header file
8  }
```

Listing 5: Exemplary "RandomRouting.ned" file content.

3. Algorithm header file. After the module has been declared and can be found by FLoRaSat, it is time to implement the actual routing logic. The first task, is to declare the handlers of the "RoutingBase" that will be overridden by this module, which is done in "RandomRouting.h". Since this algorithm does not react to queue sizes, dropped packets, or other events, it is enough to override "handlePacket". The namespace of the file is set to "flora::routing", which must match the class path, specified in "RandomRouting.ned", except that "'flora"' is the base bath of the module file, which can therefore be omitted.

```
#ifndef __FLORA_ROUTING_RANDOMROUTING_H_
#define __FLORA_ROUTING_RANDOMROUTING_H_

#include <omnetpp.h>
#include "core/Utils.h"
#include "inet/common/packet/Packet.h"
#include "routing/RoutingBase.h"

namespace flora {
namespace routing {
class RandomRouting : public RoutingBase {
   public:
    virtual void handlePacket(inet::Packet* pkt) override;
};
}  // namespace routing
}  // namespace flora
#endif  // __FLORA_ROUTING_RANDOMROUTING_H_
```

Listing 6: Exemplary "RandomRouting.h" file content.

4. Algorithm source file. The corresponding implementation of the handler is shown in lst. 7, which is the content added to "RandomRouting.cc". In summary, the logic can be explained as follows:

- First, the destination ground station identifier is extracted from the packet. Here, we make use of metadata added by the simulator by getting the "CstRoutingTag". For real algorithms, it is more realistic to create a packet on the ground station routing module or the first satellite and load the destination identifier from it.
- Second, it is checked if the satellite is connected to the ground station, and if yes, the sender is instructed to send the packet to the GSL that connects the satellite to the destination ground station identifier and the function returns, otherwise the execution continues.
- Third, all available ISLs to adjacent satellites are collected into a vector. Note, that failed links are also be considered as available by the method call used, since it considers pre-plannable connections. Failed link detection capabilities must be implemented by the routing module itself, e.g., by overriding "handlePacketDrop".
- Forth, if the vector contains no available direction, the sender is instructed to drop the packet, because no route was found, otherwise, a random direction from the vector is chosen and the sender is instructed to forward the packet in that direction.

```cpp
#include "RandomRouting.h"
namespace flora {
namespace routing {
Define_Module(RandomRouting); // announce class to simulation kernel
void RandomRouting::handlePacket(inet::Packet* pkt) {
    auto tag = pkt->getTag<routing::CstRoutingTag>();
    int dstGs = tag->getDstGs();

    // check if connected to destination groundstation
    if (getGroundlinkIndex(satId, dstGs) != -1) {
        sender->sendMessage(pkt, ISLDirection::GROUNDLINK, false, dstGs);
        return;
    }

    std::vector<ISLDirection> availableDirections;
    if (sat->hasLeftSat()) {
        availableDirections.emplace_back(ISLDirection::LEFT);
    }
    if (sat->hasUpSat()) {
        availableDirections.emplace_back(ISLDirection::UP);
    }
    if (sat->hasRightSat()) {
        availableDirections.emplace_back(ISLDirection::RIGHT);
    }
    if (sat->hasDownSat()) {
        availableDirections.emplace_back(ISLDirection::DOWN);
    }

    if (availableDirections.empty()) {
        sender->dropPacket(pkt, PacketDropReason::NO_ROUTE_FOUND, true);
    } else {
        int index = intrand(availableDirections.size());
        ISLDirection dir = availableDirections[index];
        sender->sendMessage(pkt, dir, false, -1);
    }
}
} // namespace routing
} // namespace flora
```

Listing 7: Exemplary "RandomRouting.cc" file content.

A.1.2 Algorithm Configuration and Execution

When this point is reached, the implementation of the algorithm is complete. It would also be possible to repeat the steps in a slightly modified form to override the logic used by the ground stations to select the first and last satellites by creating another module that extends "RoutingBaseGs" instead of "RoutingBase". To use the algorithm during the actual simulation runs, the simulation must be configured to use the added module. The simplest way to use the algorithm is to add a "randomroute.ini" file to the "Routing" folder in the top-level "simulations" folder, the contents of which are shown in lst. 8. Inside the omnetpp IDE it is then possible to right-click the configuration file and run the file as an omnetpp simulation.

```
[General]
include base.ini
sim-time-limit = 6000s                          # simulation duration

# Groundstations
include groundstations/WorldWide-20.ini
# Constellation
include constellations/Iridium.ini
# Traffic pattern
include traffic/UDP-burst-small-sleep.ini

# Recording config
*.packetRecorder.repetition = "0"
*.packetRecorder.simName = "Regular"             # set simulation name

*.groundStation[*].packetGenerator.maxHops = 25  # set TTL

*.packetRecorder.algName = "Random"              # set name for results
*.loRaGW[*].routing.typename = "RandomRouting"   # specify algorithm module
```

Listing 8: Exemplary "randomrouting.ini" file content.

References

[1] Enrico Del Re, R. Fantacci, and Giovanni Giambene. "Characterization of user mobility in Low Earth Orbit Mobile Satellite Systems." In: *Wireless Networks* 6 (July 2000), pp. 165–179. DOI: 10.1023/A:1019181312967.

[2] Ömer Korçak and Fatih Alagöz. "Virtual topology dynamics and handover mechanisms in Earth-fixed LEO satellite systems." In: *Computer Networks* 53.9 (2009), pp. 1497–1511. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2009.01.010. URL: https://www.sciencedirect.com/science/article/pii/S1389128609000346.

[3] Harry W. Jones. "Much Lower Launch Costs Make Resupply Cheaper than Recycling for Space Life Support." In: NASA Ames Research Center Moffett Field, CA, United States. ICES-2017, Oct. 2017.

[4] Gregory Stock, Juan Fraire, and Holger Hermanns. *Distributed On-Demand Routing for LEO Mega-Constellations: A Starlink Case Study*. Aug. 2022. DOI: 10.48550/arXiv.2208.02128. URL: https://arxiv.org/abs/2208.02128.

[5] Haichao Tan and Lidong 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*. 2014, pp. 357–361. DOI: 10.1109/CSE.2014.93.

[6] Ioannis Gragopoulos, Evangelos Papapetrou, and Fotini-Niovi Pavlidou. "Performance study of adaptive routing algorithms for LEO satellite constellations under Self-Similar and Poisson traffic." In: *Space communications* 16 (Jan. 1999), pp. 15–22.

[7] Juan A. Fraire et al. "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)*. 2022, pp. 464–470. DOI: 10.1109/WoWMoM54355.2022.00072.

[8] OpenSim Ltd. *Omnet++ Discrete Event Simulator*. 2022. URL: https://omnetpp.org/.

[9] European Space Agency. *Types of orbits*. URL: https://www.esa.int/Enabling_Support/Space_Transportation/Types_of_orbits (visited on 06/28/2023).

[10] Dan Stillman. "What is a Satellite?" In: *NASA Education. NASA* 12 (2014), pp. 5–8.

[11] Federal Aviation Administration. *Describing Orbits*. URL: https://www.faa.gov/sites/faa.gov/files/about/office_org/headquarters_offices/avs/III.4.1.4_Describing_Orbits.pdf (visited on 06/27/2023).

[12] Michael A. Earle. "Sphere to Spheroid Comparisons." In: *The Journal of Navigation* 59.3 (2006), pp. 491–496. DOI: 10.1017/S0373463306003845.

[13] Xinmeng Liu et al. "A low-complexity probabilistic routing algorithm for polar orbits satellite constellation networks." In: *2015 IEEE/CIC International Conference on Communications in China (ICCC)*. 2015, pp. 1–5. DOI: 10.1109/ICCChina.2015.7448759.

[14] Hongcheng Yan, Qingjun Zhang, and Yong Sun. "A Novel Routing Scheme for LEO Satellite Networks Based on Link State Routing." In: *2014 IEEE 17th International Conference on Computational Science and Engineering*. 2014, pp. 876–880. DOI: 10.1109/CSE.2014.178.

[15] National Aeronautics and Space Administration. *Catalog of Earth Satellite Orbits*. URL: https://earthobservatory.nasa.gov/features/OrbitsCatalog (visited on 06/28/2023).

[16] F. Vatalaro et al. "Analysis of LEO, MEO, and GEO global mobile satellite systems in the presence of interference and fading." In: *IEEE Journal on Selected Areas in Communications* 13.2 (1995), pp. 291–300. DOI: 10.1109/49.345873.

[17] Geoff Huston. *Using LEOs and GEOs*. URL: https://circleid.com/posts/20220428-using-leos-and-geos (visited on 06/29/2023).

[18] C.E. Fossa et al. "An overview of the IRIDIUM (R) low Earth orbit (LEO) satellite system." In: *Proceedings of the IEEE 1998 National Aerospace and Electronics Conference. NAECON 1998. Celebrating 50 Years (Cat. No.98CH36185)*. 1998, pp. 152–159. DOI: 10.1109/NAECON.1998.710110.

[19] Société Européenne des Satellites. *GEO, MEO, AND LEO*. URL: https://www.satellitetoday.com/wp-content/uploads/2021/02/Guide-GEO-MEO-LEO-1.pdf (visited on 06/29/2023).

[20] Israel Leyva-Mayorga et al. "NGSO Constellation Design for Global Connectivity." In: *arXiv e-prints*, arXiv:2203.16597 (Mar. 2022), arXiv:2203.16597. DOI: 10.48550/arXiv.2203.16597. arXiv: 2203.16597 [cs.NI].

[21] Giovanni Palmerini and Filippo Graziani. "Polar elliptic orbits for global coverage constellations." In: *Astrodynamics Conference*. 1994, p. 3720. DOI: 10.2514/6.1994-3720.

[22] J. G. Walker. "Circular Orbit Patterns Providing Continuous Whole Earth Coverage." In: 1970. URL: https://apps.dtic.mil/sti/pdfs/AD0722776.pdf.

[23] J. G. Walker. "Continuous Whole-Earth Coverage by Circular-Orbit Satellite Patterns." In: 1977. URL: https://apps.dtic.mil/dtic/tr/fulltext/u2/a044593.pdf.

[24] E. Ekici, I.F. Akyildiz, and M.D. Bender. "A distributed routing algorithm for datagram traffic in LEO satellite networks." In: *IEEE/ACM Transactions on Networking* 9.2 (2001), pp. 137–147. DOI: 10.1109/90.917071.

[25] Yong Lu, Fuchun Sun, and Youjian Zhao. "Virtual Topology for LEO Satellite Networks Based on Earth-Fixed Footprint Mode." In: *IEEE Communications Letters* 17.2 (2013), pp. 357–360. DOI: 10.1109/LCOMM.2013.011113.122635.

[26] Yonghu Zhu et al. "Software defined routing algorithm in LEO satellite networks." In: *2017 International Conference on Electrical Engineering and Informatics (ICELTICs)*. 2017, pp. 257–262. DOI: 10.1109/ICELTICS.2017.8253282.

[27] Ömer Korçak, Fatih Alagoz, and Abbas Jamalipour. "Priority-based adaptive routing in NGEO satellite networks." In: *Int. J. Communication Systems* 20 (Mar. 2007), pp. 313–333. DOI: 10.1002/dac.823.

[28] Matthias Motzigemba, Herwig Zech, and Philipp Biller. "Optical Inter Satellite Links for Broadband Networks." In: *2019 9th International Conference on Recent Advances in Space Technologies (RAST)*. 2019, pp. 509–512. DOI: 10.1109/RAST.2019.8767795.

[29] Bai Jianjun et al. "Compact explicit multi-path routing for LEO satellite networks." In: *HPSR. 2005 Workshop on High Performance Switching and Routing, 2005.* 2005, pp. 386–390. DOI: 10.1109/HPSR.2005.1503260.

[30] M. Werner. "A dynamic routing concept for ATM-based satellite personal communication networks." In: *IEEE Journal on Selected Areas in Communications* 15.8 (1997), pp. 1636–1648. DOI: 10.1109/49.634801.

[31] Vaibhav Bhosale et al. "A Characterization of Route Variability in LEO Satellite Networks." In: Mar. 2023, pp. 313–342. ISBN: 978-3-031-28485-4. DOI: 10.1007/978-3-031-28486-1_14.

[32] Markus Werner et al. "Analysis of System Parameters for LEO/ICO-Satellite Communication Networks." In: *Selected Areas in Communications, IEEE Journal on* 13 (Mar. 1995), pp. 371–381. DOI: 10.1109/49.345881.

[33] Juan Misael Gongora-Torres et al. "Elevation Angle Characterization for LEO Satellites: First and Second Order Statistics." In: *Applied Sciences* 13.7 (2023). ISSN: 2076-3417. DOI: 10.3390/app13074405. URL: https://www.mdpi.com/2076-3417/13/7/4405.

[34] T.-H. Chan, B.S. Yeo, and L.F. Turner. "A localized routing scheme for leo satellite networks." In: 2003.

[35] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs." In: *Numer. Math.* 1.1 (Dec. 1959), pp. 269–271. ISSN: 0029-599X. DOI: 10.1007/BF01386390. URL: https://doi.org/10.1007/BF01386390.

[36] Jifeng Jin et al. "A Disruption Tolerant Distributed Routing Algorithm in LEO Satellite Networks." In: *Applied Sciences* 12 (Apr. 2022), p. 3802. DOI: 10.3390/app12083802.

[37] George Fishman. *Discrete-event Simulation: Modeling, Programming, and Analysis*. Vol. 5. Jan. 2002. DOI: 10.1007/978-1-4757-3552-9.

[38] Vladimir Rykov and Dmitry Kozyrev. "On Sensitivity of Steady-State Probabilities of a Cold Redundant System to the Shapes of Life and Repair Time Distributions of Its Elements." In: Sept. 2018, pp. 391–402. ISBN: 978-3-319-76034-6. DOI: 10.1007/978-3-319-76035-3_28.

[39] Abdul Halim Zaim and Derya Yiltas. "PERFORMANCE ANALYSIS AND ROUTING TECHNIQUES IN LEO SATELLITE SYSTEMS." In: *IU-Journal of Electrical & Electronics Engineering* 5 (2005), pp. 1363–1372. URL: https://api.semanticscholar.org/CorpusID:12342772.

[40] Mohamed Atef Ali Madni, Saeid Iranmanesh, and Raad Raad. "DTN and Non-DTN Routing Protocols for Inter-CubeSat Communications: A comprehensive survey." In: *Electronics* 9.3 (2020). ISSN: 2079-9292. DOI: 10.3390/electronics9030482. URL: https://www.mdpi.com/2079-9292/9/3/482.

[41] Qi Xiaogang et al. "A Survey of Routing Techniques for Satellite Networks." In: *Journal of Communications and Information Networks* 1.4 (2016), pp. 66–85. DOI: 10.11959/j.issn.2096-1081.2016.058.

[42] Juan A. Fraire. "Inventory of Practice: Topologies and Routing." In: (2022). URL: https://mission-project.eu/deliverables/d31-topologies-routing-inventory.pdf.

[43] Xinmeng Liu et al. "A Low-Complexity Routing Algorithm Based on Load Balancing for LEO Satellite Networks." In: *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*. 2015, pp. 1–5. DOI: 10.1109/VTCFall.2015.7390795.

[44] E. Papapetrou, S. Karapantazis, and F.-N. Pavlidou. "Distributed on-demand routing for LEO satellite systems." In: *Computer Networks* 51.15 (2007), pp. 4356–4376. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2007.05.008. URL: https://www.sciencedirect.com/science/article/pii/S1389128607001582.

[45] Tarik Taleb et al. "SAT04-3: ELB: An Explicit Load Balancing Routing Protocol for Multi-Hop NGEO Satellite Constellations." In: *IEEE Globecom 2006*. 2006, pp. 1–5. DOI: 10.1109/GLOCOM.2006.523.

[46] Manuel Roth, Hartmut Brandt, and Hermann Bischl. "Implementation of a geographical routing scheme for low Earth orbiting satellite constellations using intersatellite links." In: *International Journal of Satellite Communications and Networking* 39.1 (2021), pp. 92–107. DOI: https://doi.org/10.1002/sat.1361. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sat.1361. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sat.1361.

[47] Xuezhi Ji et al. "A destruction-resistant on-demand routing protocol for LEO satellite network based on local repair." In: *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. 2015, pp. 2013–2018. DOI: 10.1109/FSKD.2015.7382259.

[48] Quan Chen et al. "A distributed congestion avoidance routing algorithm in mega-constellation network with multi-gateway." In: *Acta Astronautica* 162 (2019), pp. 376–387. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2019.05.051. URL: https://www.sciencedirect.com/science/article/pii/S0094576518317387.

[49] Ömer Korçak and Fatih Alagöz. "Priority-based Adaptive Shortest Path Routing for IP over LEO Satellite Networks." In: 2005. URL: https://api.semanticscholar.org/CorpusID:16536703.

[50] Hong Seong Chang et al. "FSA-based link assignment and routing in low-earth orbit satellite networks." In: *IEEE Transactions on Vehicular Technology* 47 (1998), pp. 1037–1048. URL: https://api.semanticscholar.org/CorpusID:17881749.

[51] Hui Li and Xuemai Gu. "Adaptive ATM routing in Walker-Delta satellite communication networks." In: *2006 1st International Symposium on Systems and Control in Aerospace and Astronautics*. 2006, 6 pp.–373. DOI: 10.1109/ISSCAA.2006.1627646.

[52] Yixin HUANG et al. "Reinforcement learning based dynamic distributed routing scheme for mega LEO satellite networks." In: *Chinese Journal of Aeronautics* 36.2 (2023), pp. 284–291. ISSN: 1000-9361. DOI: https://doi.org/10.1016/j.cja.2022.06.021. URL: https://www.sciencedirect.com/science/article/pii/S1000936122001297.

[53] L. Breslau et al. "Advances in network simulation." In: *Computer* 33.5 (2000), pp. 59–67. DOI: 10.1109/2.841785.

[54] NS3-maintainers. *Network Simulator 3*. URL: https://www.nsnam.org/ (visited on 07/24/2023).

[55] Magister Solutions. *Satellite Network Simulator 3*. URL: https://satellite-ns3.com (visited on 07/24/2023).

[56] ESA. *Real-time Satellite Network Emulator*. URL: https://artes.esa.int/projects/realtime-satellite-network-emulator (visited on 07/24/2023).

[57] Mengjie Liu et al. "Large-Scale Small Satellite Network Simulator: Design and Evaluation." In: *2020 3rd International Conference on Hot Information-Centric Networking (HotICN)*. 2020, pp. 194–199. DOI: 10.1109/HotICN50779.2020.9350838.

[58] Ari Keränen, Jörg Ott, and Teemu Kärkkäinen. "The ONE Simulator for DTN Protocol Evaluation." In: *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. Simutools '09. Rome, Italy: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009. ISBN: 9789639799455. DOI: 10.4108/ICST.SIMUTOOLS2009.5674. URL: https://doi.org/10.4108/ICST.SIMUTOOLS2009.5674.

[59] Roberto Puddu, Vlad Popescu, and Maurizio Murroni. "An open source satellite network simulator for quality based multimedia broadband traffic management." In: *2022 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. 2022, pp. 01–07. DOI: 10.1109/BMSB55706.2022.9828566.

[60] Marco Cuollo et al. "Interlink and coverage analysis for small satellite constellations." In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 227.7 (2013), pp. 1201–1212. DOI: 10.1177/0954410012453678. eprint: https://doi.org/10.1177/0954410012453678. URL: https://doi.org/10.1177/0954410012453678.

[61] Gottfried Konecny. "Small satellites–A tool for Earth observation?" In: (Jan. 2004).