Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

BACHELOR DEGREE THESIS

TFG TITLE: ANALYSIS OF OSPFv3 IN LEO SATELLITE NETWORKS

DEGREE: Double bachelor's degree in Aerospace Systems Engineering and Telecommunications Systems

AUTHOR: Daniel Román Martín

DIRECTOR: Sergio Machado Sánchez

CODIRECTOR: Jorge Mata Díaz

DATE: July 24th, 2023

Títol: ANÀLISI D'OSPFv3 A XARXES DE SATÈL·LITS LEO

Autor: Daniel Román Martín

Director: Sergio Machado Sánchez

Codirector: Jorge Mata Díaz

Data: 24 de juliol de 2023

Resum

La comunicació via xarxes de satèl·lits esta en continua investigació i desenvolupament, ja que ofereix múltiples millores respecte a les xarxes terrestres clàssiques com per exemple cobertura global, però tenen un inconvenient principal que s'ha de solucionar, el problema d'encaminament punt a punt.

En aquest treball s'ha desenvolupat un emulador de xarxes de satèl·lits mitjançant contenidors Linux, que han permès analitzar el comportament del protocol d'encaminament IP OSPFv3 en aquest tipus de xarxes. Concretament, s'ha analitzat el seu comportament a la constel·lació Iridium, ja que es àmpliament coneguda i utilitzada en aquest tipus d'estudis. Amb aquesta intenció s'han fet servir arxius de la topologia d'aquestes xarxes al llarg del temps, generats amb el propagador orbital HypatiaSeam, una modificació de Hypatia realitzada pel grup d'investigació SeamSAT de la UPC.

Aquest projectes es troba dins d'un altre més gran, que té com a objectiu poder fer servir una xarxa de satèl·lits LEO per a la comunicació entre avions i els centres de control de l'espai aeri. Això podria permetre centralitzar els diferents centres de control, ja que no es necessitaria que els avions tinguessin que estar en rang directe per a comunicar-se amb aquests centres, sinó que gràcies a la cobertura global que proporciones aquestes xarxes, es podrien comunicar des de qualsevol punt del planeta.

Concretament, en aquest projecte s'ha desenvolupat una plataforma d'emulació que ha permès realitzar un anàlisi del comportament del protocol OSPFv3 per a trobar rutes optimes, es a dir, de distancia mes curta en termes de la funció de cost del protocol.

Presentarem el disseny i la implementació de la plataforma d'emulació, així com l'anàlisi del rendiment d'OSPFv3 en termes de temps de convergència del protocol degut als canvis de topologia, nombre de salts entre un satèl·lit i una estació terrestre, el retard i la taxa de pèrdues.

Título: ANÁLISIS DE OSPFv3 EN REDES DE SATÉLITES LEO

Autor: Daniel Román Martín

Director: Sergio Machado Sánchez

Codirector: Jorge Mata Díaz

Fecha: 24 de julio de 2023

Resumen

La comunicación vía redes de satélites está en continua investigación y desarrollo ya que ofrece múltiples avances respecto a las clásicas redes terrestres como por ejemplo cobertura global, pero tiene un inconveniente principal a solucionar, el problema de encaminamiento punto a punto.

En este trabajo se ha desarrollado un emulador de redes de satélites utilizando contenedores Linux, que ha permitido analizar el comportamiento del protocolo de encaminamiento IP OSPFv3 en este tipo de redes. Concretamente, se ha analizado su comportamiento en la constelación Iridium, la cual es ampliamente conocida y usada en este tipo de estudios. Para ello se ha utilizado archivos de la topología de estas redes a lo largo del tiempo generados con el propagador orbital HypatiaSeam, una modificación de Hypatia realizada por el grupo de investigación SeamSAT de la UPC.

Este proyecto se enmarca dentro de otro más global cuyo objetivo es poder utilizar una red de satélites LEO para la comunicación entre los aviones y los centros de control del espacio aéreo. Esto podría permitir centralizar los diferentes centros de control, ya que no seria necesario que los aviones tuviesen que estar en rango directo para comunicarse con estos centros, sino que gracias a la cobertura global que proporcionan estas redes, se podrían comunicar desde cualquier punto del mundo.

Concretamente, en este proyecto se ha desarrollado una plataforma de emulación que ha permitido realizar un análisis del comportamiento del protocolo OSPFv3 para encontrar rutas óptimas, es decir, de distancia más corta en términos de la función de coste del protocolo.

Presentaremos el diseño y la implementación de la plataforma de emulación, así como el análisis del rendimiento de OSPFv3 en términos de tiempo de convergencia del protocolo ante cambios en la topología, número de saltos entre un satélite y una estación terrestre, el retardo y la tasa de pérdidas.

Title: ANALYSIS OF OSPFv3 IN LEO SATELLITE NETWORKS

Author: Daniel Román Martín

Director: Sergio Machado Sánchez

Codirector: Jorge Mata Díaz

Date: July 24th, 2023

Overview

Communication via satellite networks is under continuous research and development as it offers many advances over traditional terrestrial networks such as global coverage, but has a major drawback to be solved, the problem of point-to-point routing.

In this work we have developed a satellite network emulator using Linux containers, which has allowed us to analyze the behavior of the IP routing protocol OSPFv3 in this type of networks. Specifically, its behavior has been analyzed in the Iridium constellation, which is widely known and used in this type of studies. For this purpose, we have used files of the topology of these networks over time generated with the HypatiaSeam orbital propagator, a modification of Hypatia made by the SeamSAT research group of the UPC.

This project is part of a more global project whose objective is to be able to use a network of LEO satellites for communication between aircraft and airspace control centers. This would make it possible to centralize the different control centers, since it would not be necessary for aircraft to be in direct range to communicate with these centers, but thanks to the global coverage provided by these networks, they could communicate from anywhere in the world.

Specifically, in this project we have developed an emulation platform that has allowed us to analyze the behavior of the OSPFv3 protocol to find optimal routes, i.e., shortest distance in terms of the cost function of the protocol.

We will present the design and implementation of the emulation platform as well as the analysis of OSPFv3 performance in terms of protocol convergence time to topology changes, number of hops between a satellite and a ground station, delay and loss rate.

CONTENTS

INTRODUCTION

This work is part of the SeamSAT project [1], developed by a research group of the telematics engineering department of the Universitat Politècnica de Catalunya (UPC) and funded by the Agencia Estatal de Investigación (AEI).

The goal of SeamSAT project is to use the new Low Latency Low Loss Scalable Throughput Internet Service (L4S) architecture for terrestrial networks, recommended by the Internet Engineering Task Force (IETF), and its application in large Low Earth Orbit (LEO) communications networks. With this objective, the SeamSAT project will produce a proposal for algorithms and protocols to ensure the seamless integration of dense LEO communications networks with terrestrial networks. The project will give special consideration to the integration of aeronautical telecommunication services.

Satellite networks are networks capable not only of overcoming the geographical limitations of terrestrial networks, but also of providing a secure communication channel. These networks are built on a spatial platform for the acquisition, emission and analysis of spatial information in real time and are characterized by high transmission delays and losses, and dynamic topology changes, which distinguishes them form conventional terrestrial networks.

The first satellite networks were implemented in the Geostationary Earth Orbit (GEO) using the bent-pipe method. This method consists of a satellite receiving the signal form the ground station, amplifying it, changing its frequency and retransmitting it to another base station, which means that initial networks consisted of a transmitter, a receiver and a relay, which was the satellite that received and retransmit the information.

Fig. 0.1. Bent-pipe method.

Nowadays, satellite networks are composed of multiple satellites that communicate with each other via Inter-Satellite Links (ISLs) and with the ground stations via the Ground to Satellite Links (GSLs), forming satellite constellations, such as Iridium. Iridium constellation will be explained later in Section 1.1 This implies that there are a large number of paths for communication between ground stations, which together with the large delays

and the dynamic topology changes appears the main problem of these networks, i.e., the point-to-point routing problem.

In this work we are going to study and analyze the use of Open Shortest Path First version 3 (OSPFv3) routing protocol, a well-studied solution for terrestrial networks. The main objective is to describe the performance of the protocol *as-is*, with no modifications, only adjusting some protocol configuration parameters. The analysis will be done using an emulation framework that has been developed on this work and that can be used in the study of other IP routing protocols.

This work is intended to study the implementation of a new way of communication between aircraft and control centers, since thanks to the global coverage of satellite networks they could communicate with each other regardless of the position of each one.

The work is organized as follows. In Chapter 1, LEO satellite networks are introduced by explaining a typical satellite constellation, such as the Iridium constellation, and typical LEO satellite constellation distributions such as the Walker constellations. Chapter 2 presents the emulation framework developed. In Chapter 3, the OSPFv3 protocol is explained. Chapter 4 presents the results of the study. Finally, in Chapter 5 we conclude and present the future lines of this work.

CHAPTER 1. LEO SATELLITE NETWORKS

LEO satellites orbit at altitudes between 250 km and 2000 km above the Earth's surface. The orbital period, which is the time interval between two consecutive passes of a satellite over a characteristic point of the orbit, varies from 90 to 120 min. This is shown by Kepler's third law, Equation 1.1, where *G* is the gravitational constant, M_{earth} is the mass of the Earth, R_{earth} is the radius of the Earth and *h* is the height of the orbit. These satellites are only observable for 20 min by an observer who is stationary on the Earth's surface, as would be the case for base stations, and orbit at a rotational speed of more than 25000 km/h.

$$T^2 = \frac{4\pi^2 \cdot (R_{earth} + h)^3}{G \cdot M_{earth}} \quad \textbf{(1.1)}$$

A LEO satellite network consists of M satellites evenly distributed in the 360º range in N orbital planes, located at a given height and inclination. For example, a constellation consisting of 12 satellites uniformly distributed in 3 orbital planes would be a 3x4 satellite constellation. The typical inclination in LEO satellite networks is usually less than 90º. Figure 1.1 shows an example of a LEO satellite constellation.

The satellites that form these networks are cube-shaped, or a combination of 10x10x10 cm cubes. Each face of the cube usually has an antenna, except for the one opposite to the face that looks the center of the Earth that does not have any antenna. The side facing the surface of the Earth is used to connect to base stations and form the GSLs.

Two of the other faces point to the satellite preceding it or succeeding it in the same orbit. These satellites, which are in the same plane, maintain the distance between them by being in the same orbit. Therefore, there is always a link between them if the electrical or optical system is working properly. These links are known as intra-plane ISLs.

The remaining two faces observe satellites in adjacent co-rotating planes that are in a similar angular phase. These satellites modify the distance between them and even the position, so that when a certain phase of the orbit is exceeded, depending on the velocity vector of a satellite, the satellite on the right moves to the left and vice versa. The links formed between these satellites are called inter-plane ISLs.

Depending on the orbital plane in which the satellites are located, some of them orbit to the north and others to the south. When two adjacent planes rotate in opposite directions, also called counter-rotating planes, a phenomenon named seam appears, which consists in the fact that no link is formed between these two planes.

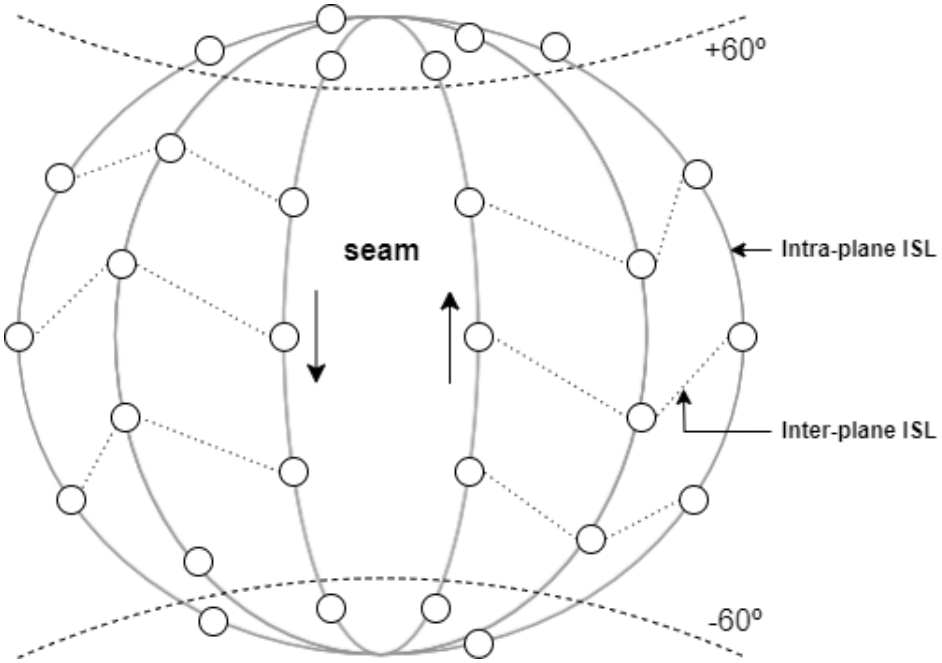

Fig. 1.1. LEO satellite constellation.

To establish links between satellites, antenna beamwidth is modified or electronic guidance system is used, since satellite faces are not perfectly aligned with those of other satellites when creating the links. When inter-plane ISLs reach a high latitude, +60º in the northern hemisphere or -60º in the southern hemisphere, are interrupted, because it is not feasible to maintain them due to the high inclination of the antenna beam.

The advantages of using LEO networks compared to satellites networks located in higher orbits are:

- Smaller satellites.
- Lower transmission losses.
- Less propagation delays.
- Higher throughput.
- Require less transmission power.
- Use of smaller antennas.
- Higher frequency reuse.
- Higher system capacity.

The main drawbacks of LEO satellite networks are:

- Larger number of satellites are needed to provide global coverage.
- Large number of topology changes.
- The interval in which a satellite is visible is smaller.
- Shorter life span.

1.1. Iridium constellation

Iridium is a satellite constellation network consisting of 66 satellites. These satellites are uniformly divided into 6 orbital planes which means that in each plane there are 11 satellites. Figure 1.2 shows an example of the Iridium satellite constellation.

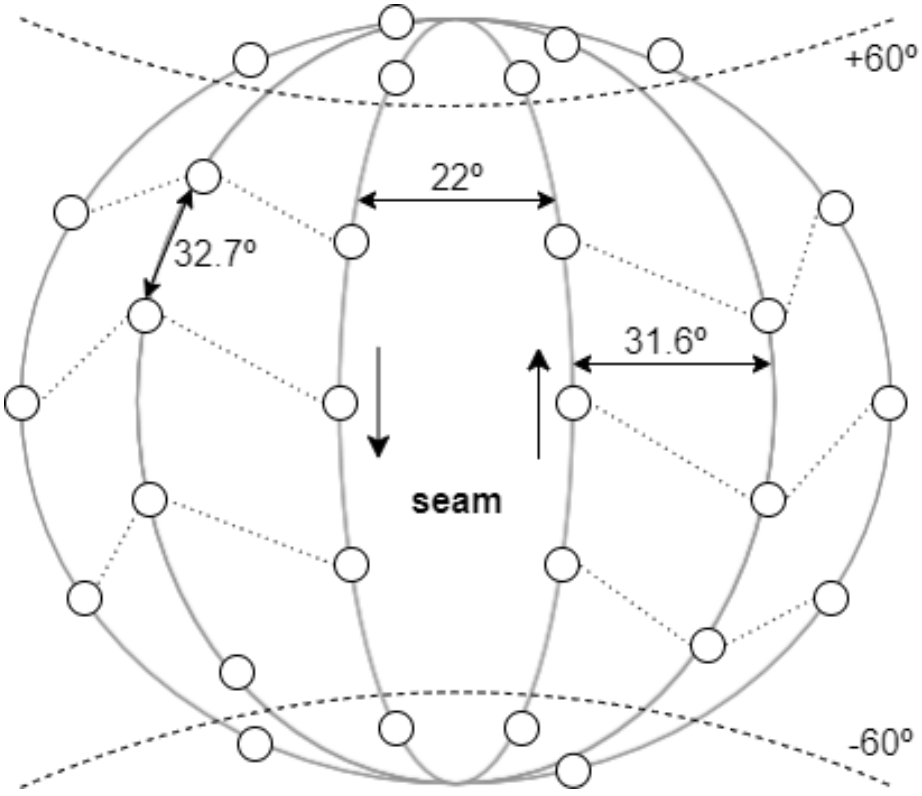

Fig. 1.2. Iridium satellite constellation.

Iridium constellation is located in the LEO orbit, specifically at a height of 778 km above the Earth's surface, and follows the Walker-star constellation distribution with an inclination of 86.4º, which is explained in Section 1.2. The separation angle between the co-rotating planes is 31.6º. The initial plane and

the final plane are counter-rotating, which as mentioned before forms a seam. The separation angle between these two planes is 22º.

The angle of separation between the satellites that are located in the same orbit plane is 32.7º, which corresponds to an interval distance of about 9 minutes, which is the average time that a satellite is visible to the same ground station. With 11 satellites per plane the orbital period is about 100 minutes. This can also be checked by applying the Equation 1.1.

The Iridium system was first developed in 1987. Initially it consisted of a network of 77 satellites, hence the name Iridium, as it was the 77th element of the periodic table. It began to be deployed in 1997 and became operational in 1998. Iridium's main feature is that it is the first satellite network to offer global coverage. It provides voice and data, paging and fax services.

The first generation of Iridium satellites was in operation until 2019, when the IridiumNEXT constellation came into use. IridiumNEXT has the same services as Iridium. It has 66 active satellites, 9 backup satellites orbiting in case one needs to be replaced and 6 satellites on the ground ready to be deployed in case of need.

Table 1.1 shows the frequency plan of the different communication links of the Iridium satellite network. The operational frequency range for communication between satellites though ISLs is form 23.18 GHz to 23.38 GHz, frequencies belonging to the K-band, which implies a bandwidth of 200 MHz.

Communication between ground station and satellites is separated into two different channels. The downlink (satellite to base station), which within the K-band, uses the operational frequency range from 19.4 GHz to 19.6 GHz. The uplink (base station to satellite) uses the operational frequency range from 29.1 GHz to 29.3 GHz, which belongs to Ka-band. The two channels have a bandwidth of 200 MHz. The ground stations connect the Iridium network to the Public Switched Telephone Network (PSTN), whose function is to enable communication between Iridium devices and the rest of the world's devices.

Table 1.1. Iridium frequency plan.

Inter-Satellite links	Ground to satellite links	Satellite to Iridium devices
K-band 23.18-23.38 GHz	Downlink: K-band 19.4-19.6 GHz Uplink: Ka-band 29.1-29.3 GHz	L-band 1.616-1.626 GHz

Communication between satellites and Iridium devices does not need to go through ground stations. It is done through the L-band using the frequency range from 1.616 GHz to 1.626 GHz, which implies a bandwidth of 10 MHz. To

communicate with these devices, the Iridium satellites have 3 sets of antennas with 16 spot beams, for a total of 48 communication cells as can be seen in Figure 1.3. As there are 66 satellites in total there are 3168 cells, but due to the satellites at high latitudes, there will only be 2150 cells active at a time. This implies that the maximum capacity of the Iridium network will be 172000 users connected at the same time, where in each cell there will be a maximum of 80 users.

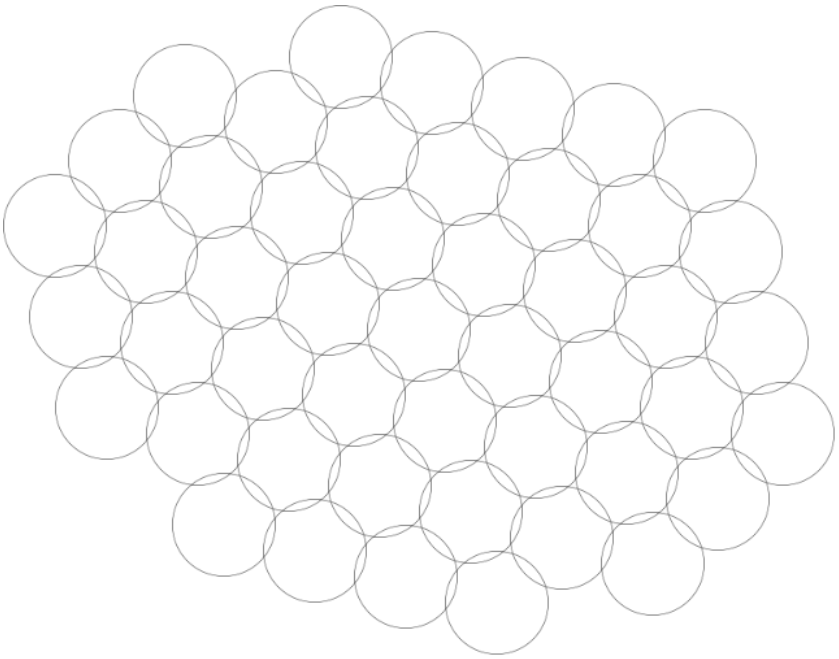

Fig. 1.3. Iridium spot beams.

1.2. Walker constellation

The Walker constellation is a common organization of LEO satellite networks and is the one on which Iridium is based. In this constellation all the satellites have the same inclination, same argument of perigee, same semi-major axis, follow a circular orbit (zero eccentricity) and have two different distributions.

Walker-star constellation or polar orbit constellation is the variant used in Iridium satellite network. In this type of constellation all satellites have an inclination close to 90º. It is characterized by an angle between adjacent planes equal to $\Delta\Omega = \pi/N$ (π-type constellation) and an angle between two neighboring satellites located in the same plane equal to $\omega = 2\pi/M$, where N is the number of orbital planes of the constellation and M is the number of satellites in each plane. Figure 1.4 shows a Walker-star constellation structure.

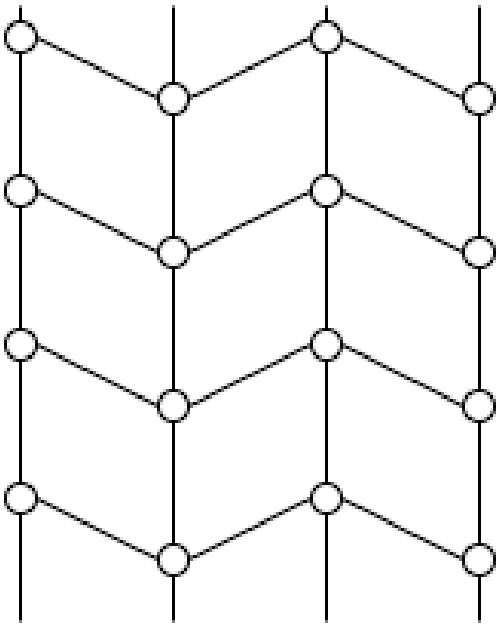

Fig. 1.4. Walker-star constellation.

Walker-delta constellation or Ballard rosette constellation is the variant used in Globalstar satellite network. In this type of constellation all satellites have an inclination smaller than 60º. It is characterized by an angle between adjacent planes equal to $\Delta\Omega = 2\pi/N$ (2π-type constellation) and an angle between two neighboring satellites located in the same plane equal to $\omega = \pi/M$. Figure 1.5 shows a Walker-delta constellation structure.

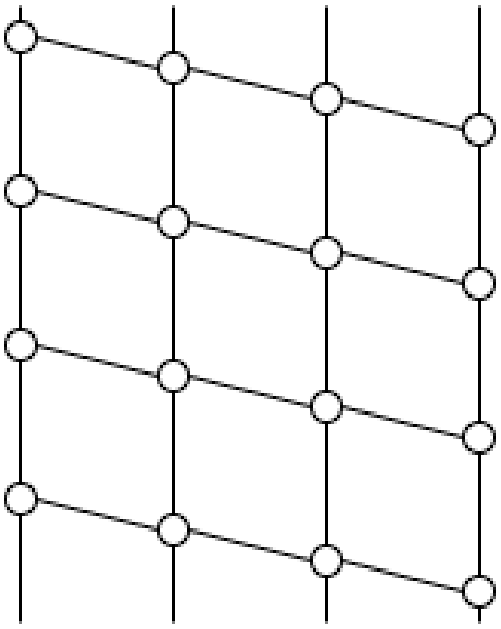

Fig. 1.5. Walker-delta constellation.

1.3. State of the art

In [13] the authors analyze the application of the OSPF protocol in Integrated Terrestrial-Satellite Networks (ITSN) by studying and analyzing networks characteristics such as transmission packet loss, end-to-end delay, among other characteristics. The results obtained in this publication are that as the number of satellites in the satellite network increases, the efficiency of the protocol decreases. This is due to the fact that the number of topology changes increase considerably causing higher delays and higher losses as the satellite network increases.

In [14] the authors propose a modification of OSPF called Orbit Prediction Shortest Path First (OPSPF) for LEO satellite networks. This protocol periodically calculates the routes between satellites to generate the routing tables instantaneously and provides a routing algorithm, which is based on flooding the network with updates to detect network irregularities, for when a link fails or recovers. The results obtained have been to have zero path convergence overhead for the scheduled changes and a reduction of more than 80% of the convergence time when there are unexpected changes (link failure) compared to OSPF.

In [15] the authors present an optimized version of OSPF called Optimized OSPF with Link Plan (OOWLP) in order to minimize OSPF convergence, packet loss and delay by periodically updating the link state, thus eliminating the network flooding process. The Constrained Shortest Path First (CSPF) method is applied to improve the throughput capacity of satellite networks in case where the network is congested. In this study, a reduction of more than 15% in packet loss, a reduction of 50 milliseconds in average delay and a throughput capacity improvement of almost 10 Mbps were achieved.

In [16] the authors introduce a new method known as Cross-Domain Aggregation Routing based on Lightweight OSPF (CDAR-L) with the objective of minimize the overhead produced with the classical OSPF, while maintaining performance in regard to delay and throughput. The results obtained by this study are that they manage to maintain the performance of OSPF and reduce overhead by 66.7% compared to OSPF. Initially this study is performed on GEO satellite networks, but they conclude that it is a first step to develop techniques applicable to LEO and multilayer satellite networks.

In [17] the authors propound to apply and analyze an OSPF-based congestion method to decrease packet loss in LEO satellite networks by designating new routing cost and modifying the interface state update procedure. In this paper the author concludes that the main purpose of reducing satellite network congestion is achieved, in addition to enhancing resource usage and global performance.

In [18] the authors postulate a method called OSPF-based Predictive Update Routing (OSPF-PUR) with the objective of reducing the protocol convergence time when a topology change occurs. To achieve this goal, the forwarding tables is updated based on local information regarding handovers and without

using the flooding feature of OSPF. The results confirm that the convergence time is reduced even while maintain full network connectivity. In terms of network performance, both delay and packet loss are reduced by applying this method.

All the aforementioned publications agree that there is still a lot of research to be done to solve the routing problem of satellite networks. In this thesis we will study and analyze the behavior of the OSPFv3 protocol applied to LEO satellite networks. As a novelty, we study the OSPF version for IPv6, whereas, these papers use the version for IPv4.

CHAPTER 2. EMULATION FRAMEWORK

This chapter explains the design and implementation of the LEO satellite network emulation framework, on which we will perform the analysis of OSPFv3 as an IP routing protocol. Figure 2.1 shows a schematic of the scenario that is developed and analyzed through this work.

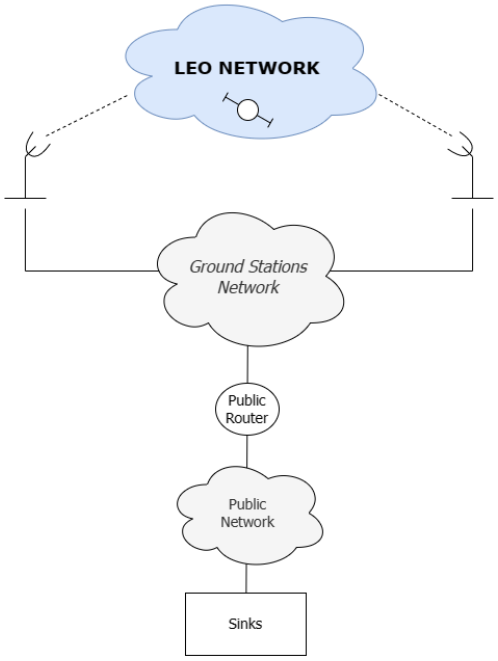

Fig. 2.1. Schematic of the scenario to be analyzed.

Network emulation is performed using Linux containers (LXC) to emulate satellites, ground stations, a public router and sinks. ISLs, GSLs and public networks emulation is done using Linux bridges.

Linux Container Daemon (LXD) [19] is a container and virtual machine management system for Linux and it is the tool that we use to create and manage containers and bridges. It also provides a Python library, pylxd [24], which we use to programmatically manage the containers and virtual machines.

At the beginning of this work, we used the IPMininet [32] Python library for network emulation. IPMininet is an extension of Mininet [33] that allows the emulation of complex IP networks and implements classes that emulate routers and links. It also contains the implementation of routing protocols such as OSPFv3. The main reason why this method did not work for us is that, when executing the topology events, they could not all be executed at the same time, but had to be executed sequentially. This implies that, for large satellite constellations, it would take a long time to produce the changes and results such as the convergence time of OSPFv3 would be unrealistic.

For this reason, using LXD and several libraries provided by Python, we have implemented a completely new way of emulating satellite networks, which will be explained and detailed throughout this chapter.

2.1. LXD – Linux Container Daemon

LXD is a management tool for Linux containers that makes it possible to automate management and ease control of them using a collection of predefined commands. LXD also offers images for many different Linux distributions, such as Ubuntu and Alpine. In addition to Linux containers, LXD also gives you the possibility to manage virtual machines. We will focus on containers, as this is the instance type used for the implementation of this project.

Figure 2.2 shows a comparison between virtual machines and containers. LXC is a product of virtualization, but instead of virtual machines that can run a different operating system than the host, LXC uses operating system-level virtualization techniques. There is only one kernel which, through various functions, provides some processes with partial views so that containers believe they are operating on a separate system. The host and all containers use the same kernel, but while the host can see all processes, the containers can only see their own.

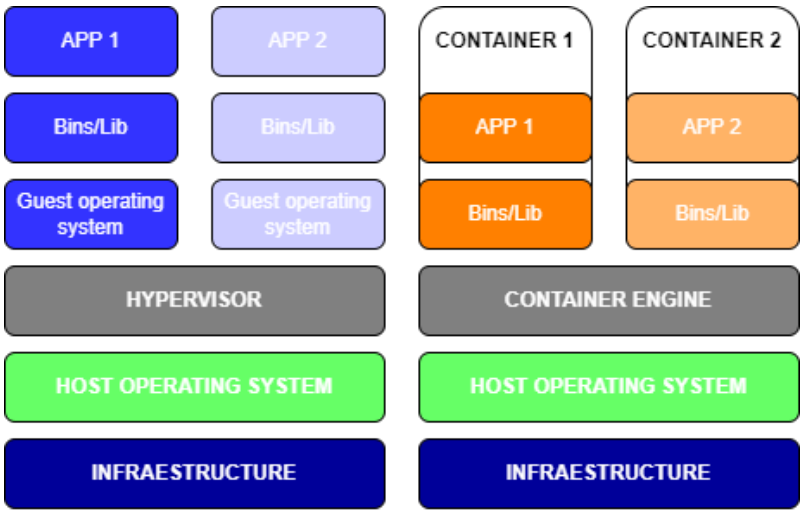

Fig. 2.2. Comparison between virtual machines and containers.

The main advantage of containers over virtual machines is efficiency. By not needing to run a different operating system or replicate the virtual machine's hardware, a container is substantially lighter. After all, the processes running in a container are native to the host computer. As a result, performance improves and more containers can run simultaneously on a single computer than virtual machines.

In the following, we detail the steps we have followed for the installation of LXD in an Ubuntu host system.

The LXD package installation is done via snap[1].

```
sudo snap install lxd
```
(2.1)

The installation of LXC creates a system group named `lxd`. A non-root user must be added to this group in order to execute `lxd` commands.

```
sudo adduser [username] lxd
```
(2.2)

LXD allows several levels of configuration. For the purpose of this project the simplest configuration of all is enough.

```
lxd init --minimal
```
(2.3)

The initialization process creates a virtual bridge interface named `lxdbr0` that performs Layer 3 functionalities such as Network Address Translation (NAT[2]) and Dynamic Host Configuration Protocol (DHCP[3]). A container with an interface connected to the `lxdbr0` bridge can access the public network.

LXD command `lxc network list` shows information of all the host devices. As can be seen in Figure 2.3, after the installation it shows the `lxdbr0` bridge, and in this case, it also shows the host physical interface `enp0s3`.

```
+--------+----------+---------+----------------+---------------------------+
|  NAME  |   TYPE   | MANAGED |      IPV4      |           IPV6            |
+--------+----------+---------+----------------+---------------------------+
| enp0s3 | physical | NO      |                |                           |
+--------+----------+---------+----------------+---------------------------+
| lxdbr0 | bridge   | YES     | 10.72.162.1/24 | fd42:e487:c61c:aea5::1/64 |
+--------+----------+---------+----------------+---------------------------+
```

Fig. 2.3. Default LXD configuration.

[1] Snap: Package manager designed to work with Linux to make it easier to install and run applications.
[2] NAT: IP address translation mechanism that connects the Internet to private addressing networks and thus saves public IP address.
[3] DHCP: Protocol that allows clients to obtain their IP configurations and avoids duplication of IP addresses on the same subnet. Facilitates network growth and management.

The `lxdbr0` bridge is automatically assigned an IPv4 address and an IPv6 address. The assigned IPv4 address belongs to the private network range 10.0.0.0/8. The IPv6 address is a Unique Local Address (ULA), which is part of the private range fc00::/7. Two other important aspects are that `lxdbr0` is a bridged network and that it is managed. These two properties will be explained in Section 2.1.3. Note that the host physical interface is not managed by LXD.

2.1.1. Images

LXD creates containers from Linux images stored in images repository servers. Different containers can be launched from the same image. Containers can only be created from images based on Linux, while virtual machines can run any operating system. In addition, LXD enables you to specify aliases for each image to simplify its management.

LXD provides a list of images available on the repository servers of each Linux distribution for the creation of containers, that can be retrieved with the command:

```
lxc image list images:[Linux_distro] type=container
```
(2.4)

Figure 2.4 shows a list of all the Alpine distribution images available for the creation of containers.

ALIAS	FINGERPRINT	PUBLIC	DESCRIPTION	TYPE	SIZE
alpine/3.17 (3 more)	8b1027a3af57	yes	Alpine 3.17 amd64	CONTAINER	2.93MB
alpine/3.17/arm64 (1 more)	eb43065cb004	yes	Alpine 3.17 arm64	CONTAINER	2.71MB
alpine/3.17/armhf (1 more)	a1c0cf89d6c2	yes	Alpine 3.17 armhf	CONTAINER	2.58MB
alpine/3.17/cloud (1 more)	8c1c7b779f67	yes	Alpine 3.17 amd64	CONTAINER	19.47MB
alpine/3.17/cloud/arm64	60c41fbe7de5	yes	Alpine 3.17 arm64	CONTAINER	18.81MB
alpine/3.17/cloud/armhf	a6d04a212112	yes	Alpine 3.17 armhf	CONTAINER	18.35MB
alpine/3.17/cloud/i386	1968a71b4c26	yes	Alpine 3.17 i386	CONTAINER	20.06MB
alpine/3.17/cloud/ppc64el	e8d1f2c503f5	yes	Alpine 3.17 ppc64el	CONTAINER	19.50MB
alpine/3.17/i386 (1 more)	b6c24ae6c9c2	yes	Alpine 3.17 i386	CONTAINER	2.99MB
alpine/3.17/ppc64el (1 more)	09d0f43bd47e	yes	Alpine 3.17 ppc64el	CONTAINER	2.82MB
alpine/3.17/s390x (1 more)	5ff54062a5db	yes	Alpine 3.17 s390x	CONTAINER	2.60MB

Fig. 2.4. Alpine distribution images.

Apart from using images from remote servers, there is also the possibility to create images by yourself. To create the new image from a container it is only

necessary to create and configure the container as explained in Section 2.1.2. Once you have the container, create the image with an alias to simplify later procedures and a description to make it easier to recognize each image when you have more than one.

```
lxc publish [container] --alias [alias] description=" "
```
(2.5)

LXD images can be exported executing the command:

```
lxc image export [alias] [filename]
```
(2.6)

Exporting the image creates a tar.gz file, which allows the image to be distributed and imported:

```
lxc image import [filename].tar.gz --alias [alias]
```
(2.7)

For the development of this work, four different images based on Alpine distribution have been created.

- seamsat-router: Image intended for the creation of the containers emulating satellites and the public router. This image has the FRRouting protocol suite (FRR[4]) and a traffic generator installed.
- seamsat-eb: Image intended for the creation of the containers emulating ground stations. This image has only FRR installed.
- seamsat-sink: Image intended for the creation of the containers emulating sinks. A sink is a receiver of the generator traffic.
- alpine-compiler: Image intended for compiling c files. This image has the make package, the C compiler gcc and C libraries such as libevent library installed. This image has been created in order to not increase the weight of the other images with the compiler and C libraries. So, if we need to build any source package, we will build it in this image, and then install the executables or libraries in the corresponding seamsat-router, seamsat-eb or seamsat-sink image.

[4] FRR: Open-source routing protocol suite for Linux and other Unix platforms, which contains routing protocols as OSPFv3.

2.1.1.1. Alpine Linux distribution

Alpine distribution is a free, standalone, general-purpose Linux distribution intended for advanced users who highly value security, simplicity and resource efficiency.

This distribution is compiled with Musl libc, which is a standard C library for Linux kernel operating systems and is about 8 kb in size. This small size is due to the fact that Musl has few localization data and no network services.

Alpine distribution implements BusyBox, which is a software suite that provides multiple Unix tools in a single executable file. More than 300 commands have their basic functionality replaced by this file.

Alpine allows you to create images up to 2.60 MB size, which is one aspect that makes it faster to create instances than Ubuntu, whose smallest images size is about 85.97 MB.

In Figure 2.5, you can see a comparison of an Ubuntu image and an Alpine image size.

```
+--------+-------------+--------+--------------------+-----------+---------+
| ALIAS  | FINGERPRINT | PUBLIC |     DESCRIPTION    |   TYPE    |  SIZE   |
+--------+-------------+--------+--------------------+-----------+---------+
| Alpine | 5ff54062a5db | no     | Alpine 3.17 s390x  | CONTAINER | 2.60MB  |
+--------+-------------+--------+--------------------+-----------+---------+
| Ubuntu | ea35540608dc | no     | Ubuntu jammy s390x | CONTAINER | 85.97MB |
+--------+-------------+--------+--------------------+-----------+---------+
```

Fig. 2.5. Comparison between Alpine and Ubuntu images sizes.

Initially, the image used in this project was based on Ubuntu, but due to the large number of instances that had to be created, it was decided to switch to Alpine distribution, as it considerably reduced the computation time by reducing the size of the containers.

2.1.2. Containers

Containers are the devices we use to emulate the nodes that form the networks analyzed in this project, such as satellites and base stations. There are two different ways to create a container.

When using the `lxc init` command, a container is created but not started.

```
lxc init [alias] [container] --profile [profile]
```
(2.8)

When using the `lxc launch` command, the container is created and also started.

```
lxc launch [alias] [container] --profile [profile]
```
(2.9)

In both cases, it is necessary to specify the profile name as a command line parameter, otherwise it will be created with the default profile.

A LXD profile groups several configuration options that can be applied to a single container or to multiple containers. Furthermore, several profiles can be applied successively to the same container. During this process, the defined configuration values can be overwritten. In this way, families of containers can be easily created.

```
lxc profile create [profile]
```
(2.10)

You can check the setting of any profile created at any time.

```
lxc profile show [profile]
```
(2.11)

Figure 2.6 shows the profile that the `lxc profile create` command creates, which is the default profile.

```
config: {}
description: Default LXD profile
devices:
  eth0:
    name: eth0
    network: lxdbr0
    type: nic
  root:
    path: /
    pool: default
    type: disk
name: default
used_by:
- /1.0/instances/compiler
```

Fig. 2.6. Default profile configuration.

The default profile connects the `eth0` interface directly to the `lxdbr0` bridge, creating a `nictype` network. This type of network device is explained in Section 2.1.3.

This profile can be edited to have the appropriate configuration.

```
lxc profile edit [profile]
```
(2.12)

Containers can have their associated profile removed.

```
lxc profile remove [container] [profile]
```
(2.13)

And they can be associated with another profile.

```
lxc profile add [container] [profile]
```
(2.14)

The `lxc list` command is used to obtain a list of the containers that have been created. This list contains the name of the containers, the state of the containers, whether they are stopped or started, and the IP addresses assigned to them in the `lxdbr0` network.

Figure 2.7 shows two containers that have been created, one of them with `lxc init` and the other with `lxc launch`.

```
+----------+---------+---------------------+-----------------+-----------+
|   NAME   |  STATE  |        IPV4         |      IPV6       |   TYPE    |
+----------+---------+---------------------+-----------------+-----------+
| Rinit    | STOPPED |                     |                 | CONTAINER |
+----------+---------+---------------------+-----------------+-----------+
| Rlaunch  | RUNNING | 10.72.162.77 (eth0) |                 | CONTAINER |
+----------+---------+---------------------+-----------------+-----------+
```

Fig. 2.7. Comparison between `lxc init` and `lxc launch`.

LXD allows commands to be executed without the need to be directly attached to the container, a feature that allows changes to the interface characteristics of each container to be made from code.

```
lxc exec [container] --[command]
```
(2.15)

Containers created for the emulation of the nodes will be ephemeral, which means that once they are stopped, they will also be automatically deleted. The command for stop a container is:

```
lxc stop [container]
```
(2.16)

2.1.3. Networking

LXD contains two different types of network devices, which are used to connect to containers. These two types of devices are `network` device and `nictype` device, which are mutually exclusive, meaning that only one of them can be chosen.

When using the `nictype` option, an interface not managed by LXD can be used. To use this interface the exact data required by LXD must be defined. This option cannot be changed once the device has been created.

When using the `network` option, LXD will manage the networks created, so it will be not necessary to provide any data to LXD.

```
lxc network create [network] --type=[network_type]
```
(2.17)

If the network type is not indicated, the bridge type applies by default.

When a bridge network is created in LXD, a Layer-2 bridge is generated, creating a Layer-2 segment with all containers attached to this bridge. This allows communication between different containers connected to this bridge, which will be very useful for implementing and simulating ISLs and GSLs.

Bridge network can be defined as a `nictype` or `network` device. Within managed networks, it is a type of fully managed network, which includes most of the features of LXD.

Once the network has been created, its configuration can be modified using a file editor, such as nano or vi.

```
lxc network edit [network]
```
(2.18)

For the network to be useful and functional, it is not enough to create and configure it, the network must also be attached to the interface of the chosen container.

```
lxc network attach [network] [container] [interface]
```
(2.19)

2.2. Topology builder

The network framework is divided into two parts. The topology builder, which is explained in this section, and the network emulation, which is explained in Section 2.3. The purpose of the topology builder block is to generate a JSON object with the network topology and the required configuration, such as OSPFv3 related parameters.

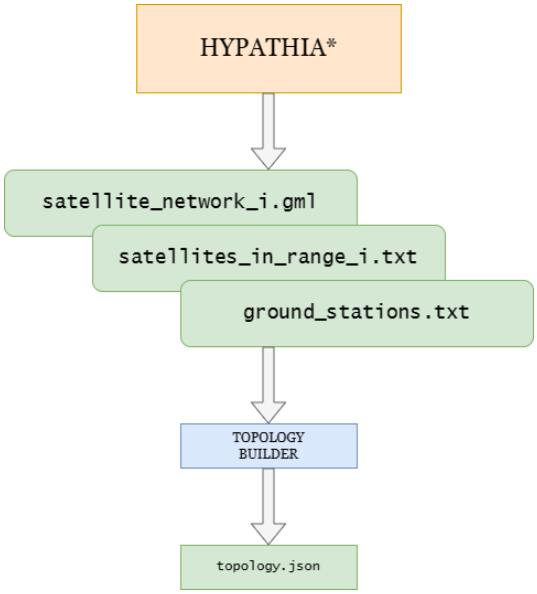

Fig. 2.8. Topology builder schematic.

The first step in the construction of the topology consists of obtaining the files generated by the Hypatia orbital propagator. Hypatia is a public framework, which can be found on GitHub [22][23], intended for the simulation of satellite networks in LEO orbit. Hypatia allows ns-3[5] simulation at the packet level, gives visualizations for ease of understanding and pre-computes the state of the network over time. It models the satellite network as an undirected weighted graph in which GSLs and ISLs act as edges, and satellites and ground stations as nodes. This tool also displays visualizations of the trajectories, the evolution of link usage and the amount of the accessible bandwidth on the routes, among other aspects.

The Hypatia framework used to obtain the files is not the original one, but some modifications have been implemented. These modifications have not been

[5] ns-3: Free software network simulator, whose main objective is to create an easy-to-use simulator for the study of any type of network.

made by me, but by the research group in charge of the SeamSAT project. The Hypatia version of the SeamSAT group is called HypatiaSeam.

The modifications made are focused on the `satgenpy` module, which is a Python-based tool that allows you to create undirected weighted graphs. These modifications make it possible to calculate ISLs and GSLs parameters, to calculate the visibility between satellites and ground stations and, finally, to generate a sequence of GML files representing the evolution of the interconnection of these elements.

Table 2.1 shows the input parameters required by Hypatia Seam.

Table 2.1. HypatiaSeam input parameters.

Related to simulation date	• Simulation epoch date.
Related to constellation parameters	• Constellation altitude. • Inclination of orbital planes. • Number of orbital planes in the constellation. • Number of satellites in an orbital plane. • Phase difference between neighboring satellites. • Separation of orbital planes/arc of RAAN.
Related to communication links	• Ground station antenna elevation angle. • Number of antennas at ground stations. • Minimum communication altitude. • ISL operating limit latitudes.

The files that model Iridium orbital propagation had already been generated for another project, so we used them directly as input to generate our topology. From all the files obtained from HypatiaSeam only three are used:

- `satellite_network_i.gml`: provides information on the satellites and ISLs that form the LEO constellation.
- `satellites_in_range_i.txt`: provides information on the satellites within range of each ground station, as well as the distance between the satellite and the ground station.
- `ground_stations.txt`: provides information on the location of earth stations.

Figure 2.9 shows a flowchart depicting the modules that constitute the topology builder block and the files it uses.

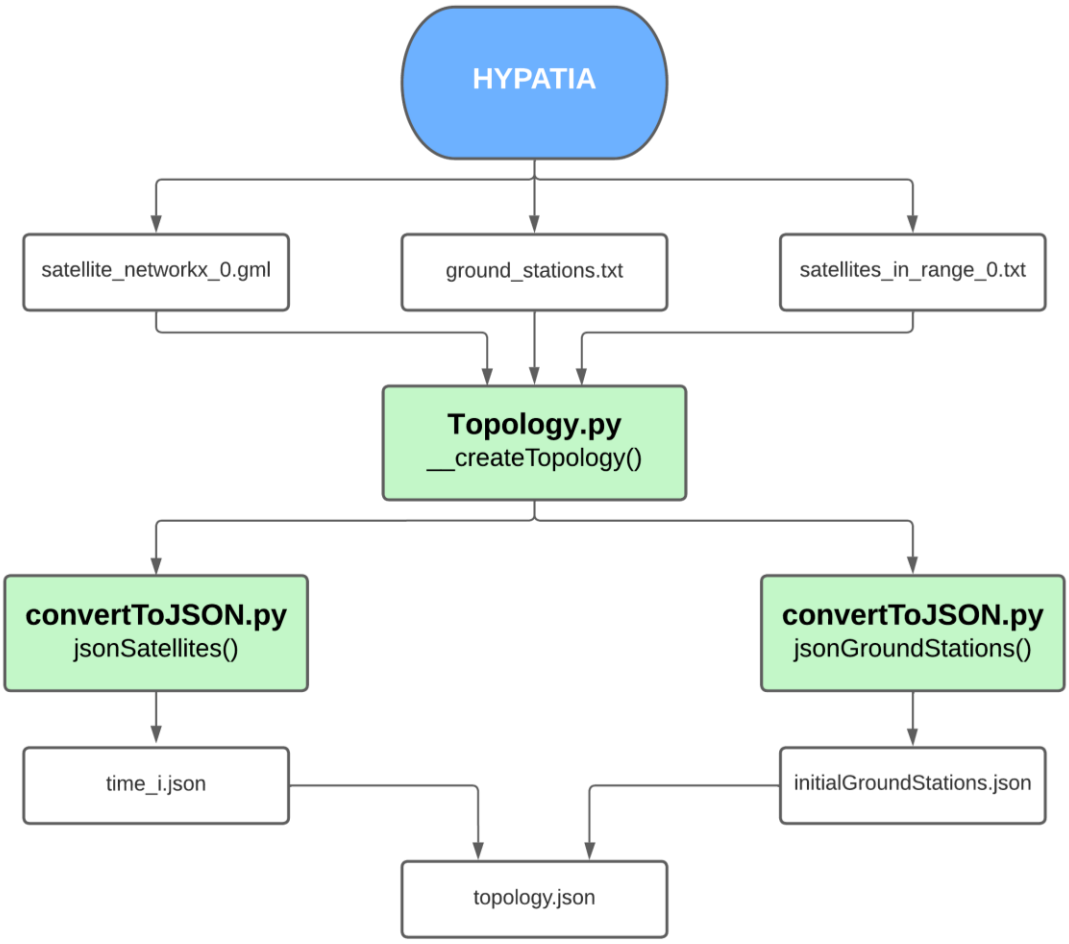

Fig. 2.9. Topology builder flowchart.

Blank rectangles represent files and colored rectangles state the function called in the Python source file. Now, we are going to explain the format files and each one of the modules of the topology builder.

The `satellite_network_i.gml` files contain the description of the nodes (satellites) and the edges between them (ISLs). Figure 2.10 shows the format of a node description.

```
node [
  id 1
  label "1"
  sat_lat_deg "81.99874325032089"
  sat_lon_deg "170.03242308521158"
  sat_alt_m "8013414.0"
]
```

Fig. 2.10. Description of nodes in GML format.

Each node is identified by the integer `id`. The `label` attribute is the representation of the `id` as a string. The attributes `sat_lat_deg`, `sat_long_deg` and `sat_alt_m` correspond to the latitude coordinate in degrees, the longitude coordinate in degrees and to the altitude above sea level in meters, respectively.

Figure 2.11 shows the format of an edge description.

```
edge [
  source 1
  target 2
  weight 20339875.685218588
  ifa 1
  ifb 0
]
```

Fig. 2.11. Description of edges in GML format.

Each edge is represented by two integers, `source` and `target`, which identify the nodes that form the link. In spite of source and target, remember that a link is bidirectional, so `source` and `target` only define both nodes connected through the link. The `weight` property is a double that represents the link distance in meters. The `ifa` and `ifb` attributes represent the interfaces of the nodes through which the link is interconnected.

The `satellites_in_range_i.txt` files contain the description of the possible links to be established between satellites and ground stations (GSLs). Figure 2.12 shows the content of one of these files.

```
GS:0 - [(10929068.0, 4), (11053459.0, 5), (10885853.0, 9)]
GS:1 - [(9413133.0, 8)]
GS:2 - [(11130172.0, 1), (9915706.0, 8)]
```

Fig. 2.12. Satellites in range in txt format.

The example topology consists of three ground stations. For example, the first line states that the base station, whose integer `id` is 0, observes the satellites with integer `id` 4, 5 and 9. In addition, it gives the distance to each of them in meters.

The `ground_stations.txt` file contain the location of the ground stations. For each earth station an integer `id` is given, the city where it is located, the latitude, longitude an elevation of its position, and, finally, its cartesian coordinates.

Figure 2.13 shows the content of this file.

```
0,Madrid,40.416500,-3.702560,0.000000,4852700.063425,-314027.764315,4113303.836106
1,Brisbane,-27.467940,153.028090,0.000000,-5047168.968040,2568544.970425,-2924317.876680
2,Tokyo,35.689500,139.691710,0.000000,-3954843.592378,3354935.154958,3700263.820217
```

Fig. 2.13. Ground stations in txt format.

The `satellite_network_i.gml`, `satellites_in_range_i.txt` and `ground_stations.txt` files are the input to the topology builder block, which will generate the `topology.json` file.

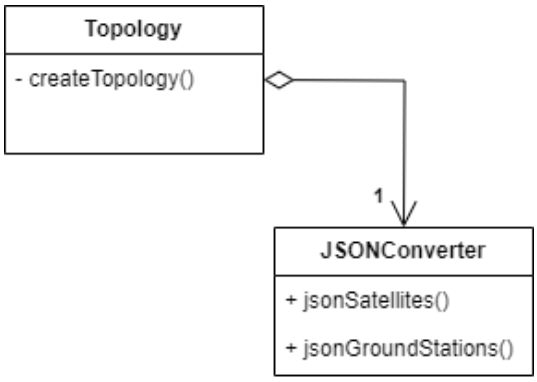

Fig. 2.14. Topology UML class diagram.

Figure 2.14 shows the UML class diagram of the two classes that we have implemented in order to generate the file `topology.json`. This file represents

the LEO satellite network as a single JSON object with two properties: `networks` and `routers`. Both properties are dictionaries.

```
"networks": {
  "isl1": {},
      .
      .
      .
}
```

Fig. 2.15. Networks' property definition.

In the case of `networks`, each key is a label that identifies each of the links, and the value is an empty JSON object. As it is an empty JSON object, when creating the network, it will be created with the default configuration. This implies that a bridge network type will be created for each defined network, which as seen in Section 2.1.3 is the type of network that we use to emulate ISLs and GSLs. Figure 2.15 shows partially the `networks` property definition where `isl1` is the name of a bridge network that will emulate an ISL between two satellites.

```
"routers": {
    "R1": {
      "ephemeral": true,
      "source": {
        "type": "image",
        "alias": "seamsat-router"
      },
      "profiles": [
        "no-nic"
      ],
      "devices": {
        "eth0": {
          "network": "isl1",
          "type": "nic",
          "host_name": "R1-eth0"
        }
      },
      "frr": {
                .
                .
                .
        },
        "router": {
          "ospf6": {}
        }
      }
    }
  }
```

Fig. 2.16. Routers' property definition.

In the case of `routers`, each key is a label that identifies each one of the satellites. For convenience this label is the char “R” concatenated with the node `id` in the `satellite_network_i.gml` file plus one. The value is a JSON object with multiple properties that are used in the configuration of the container that emulates a satellite. OSPFv3 configuration parameters are include in the `frr` JSON object that will be explained in Chapter 4. Figure 2.16 shows how to define a router object.

In the case of the ground stations, they are also defined within the `routers` dictionary. The label that identifies each earth station is the string “EB” concatenated with the `id` attribute in the `satellites_in_range_i.txt` and in the `ground_stations.txt` files plus one. Also included in this dictionary are the public router, whose identifying label is `publicRouter`, and sinks, which will be identified by the `sink_i` label.

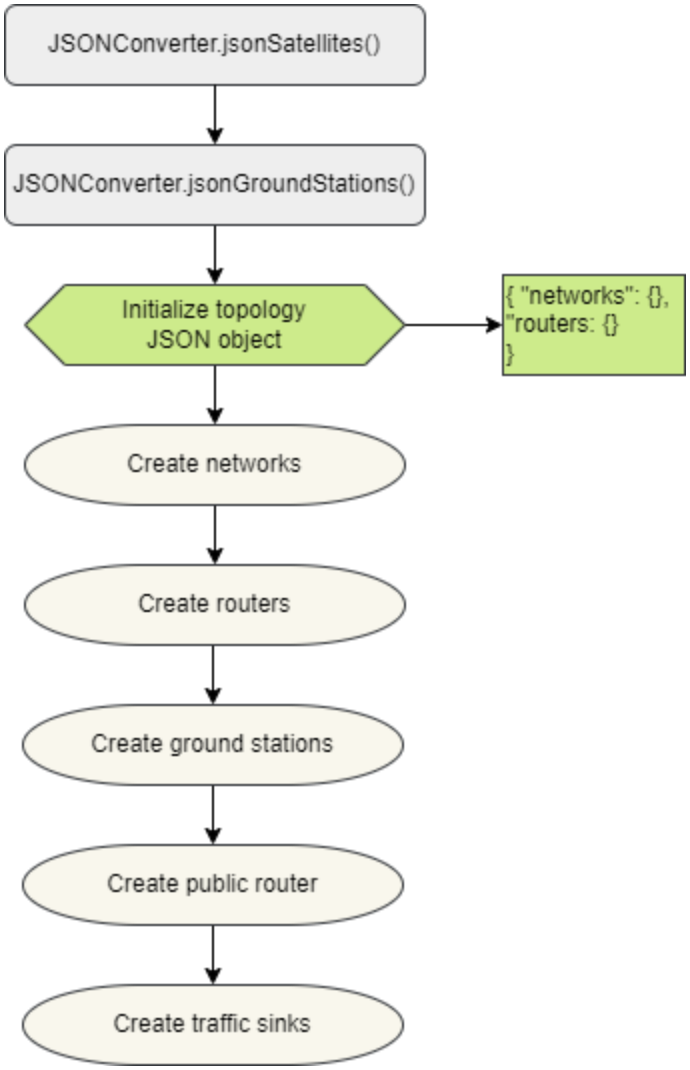

Fig. 2.17. createTopology() flowchart.

Figure 2.17 shows the process followed in the `createTopology()` method to build and obtain the `topology.json` file.

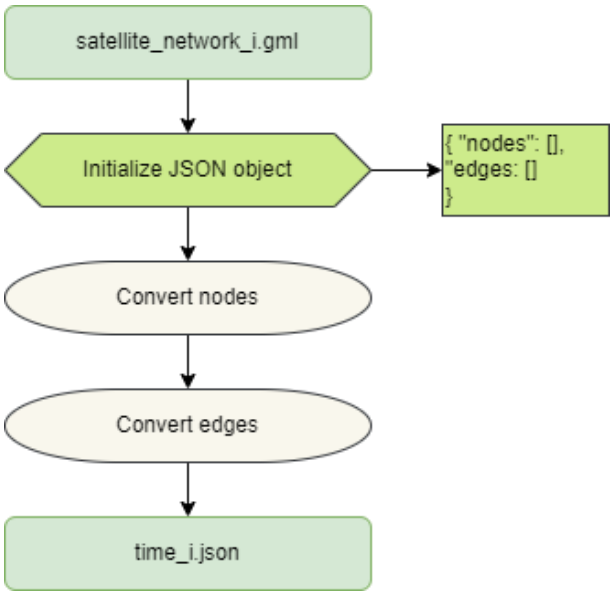

Fig. 2.18. jsonSatellites() flowchart.

The first step is to call the `jsonSatellites()` function of the `JSONConverter` class. This method converts all `satellite_network_i.gml` files to `time_i.json` files. Figure 2.18 shows the process followed to obtain them. The `time_i.json` files contain the description of the nodes (satellites) and edges (ISLs) that form the satellite constellation in JSON format.

In each of `time_i.json` files a JSON object is initialized, which has as properties the dictionaries `nodes` and `edges`. Figure 2.19 shows the format of a node description.

```
{
  "id": 1,
  "label": "1",
  "lat": "32.55980205649051",
  "lon": "-97.6636045268267",
  "alt": "785245.5"
}
```

Fig. 2.19. Description of nodes in JSON format.

Each node is identified by the integer `id`. The `label` attribute is the representation of the `id` as a string. The attributes `lat`, `long` and `alt`

correspond to the latitude coordinate in degrees, the longitude coordinate in degrees and to the altitude above sea level in meters, respectively.

Figure 2.20 shows the format of an edge description.

```
{
  "source": 1,
  "target": 2,
  "weight": "13.46793907067334",
  "ifa": 1,
  "ifb": 0
}
```

Fig. 2.20. Description of edges in JSON format.

Each edge is represented by two integers, `source` and `target`, which identify the nodes that form the bidirectional link. The `weight` property represents the link delay in milliseconds. Note that the information in the gml files was the distance in meters, here it is converted to a delay in milliseconds. The `ifa` and `ifb` attributes represent the interfaces of the nodes through which the link is interconnected.

The second step is to call the `jsonGroundStations()` function of the `JSONConverter` class. This method reads the `satellites_in_range_0.txt` and applying the hard handover selection method creates the `initialGroundStations.json` file. The hard handover method consists of selecting the closest node to the base station to choose the GSL to be formed. Figure 2.21 shows the process followed to obtain this file.

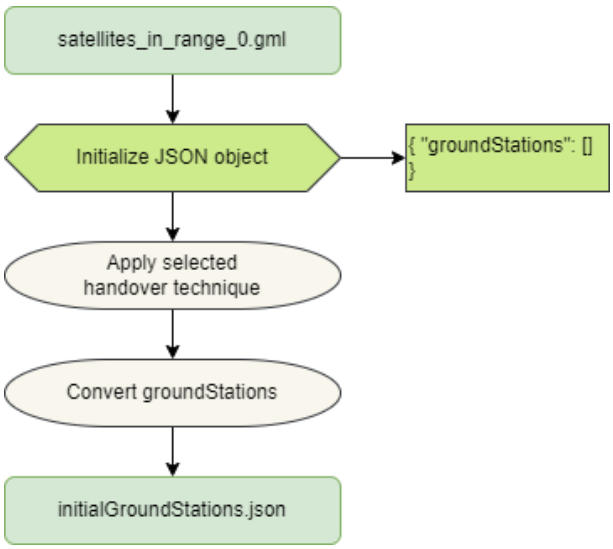

Fig. 2.21. jsonGroundStations() flowchart.

The `initialGroundStations.json` file consist on a JSON object that has as property the dictionary `groundStations`. Figure 2.22 shows the format of a GSL description.

```
{
  "groundStation": 0,
  "weight": "5.388781328181379",
  "node": 1
}
```

Fig. 2.22. Description of GSLs in JSON format.

For each GSL there is an integer `id` representing the earth station and another integer `id` representing the node, which form the link. The `weight` attribute represents the link delay in milliseconds.

The last step to obtain the `topology.json` file is to read the files `time_0.json` and `initialGroundStations.json`, and build the networks, routers, public router and sinks objects that constitute the `networks` and `routers` dictionaries presented in Figure 2.15 and in Figure 2.16.

Once the `topology.json` file is obtained, the topology builder block is completed and the network emulation part will start.

2.3. Network emulation

This section explains the network emulation block of the emulation framework. Figure 2.23 shows a schematic of the network emulation block, which is divided into two parts.

The first part consists of creating and initializing the containers that act as routers, base stations and sinks, and the bridges that emulate ISLs and GSLs with the corresponding end-to-end delay, from the `topology.json`, `time_0.json` and `initialGroundStations.json` files.

In the second part, events representing dynamic topology changes are added. These topology modifications will be programmed with the APScheduler Python library, explained in Section 2.4, and will be executed throughout the emulation every certain time interval.

The result of this emulation is a traffic capture containing the Link State Advertisements (LSAs) generated by OSPFv3, files that will allow us to obtain the path characteristics between each router to the nearest ground station on different interval times, and finally, a file that will allow us to observe packet losses, transmission delay, jitter, among other aspects.

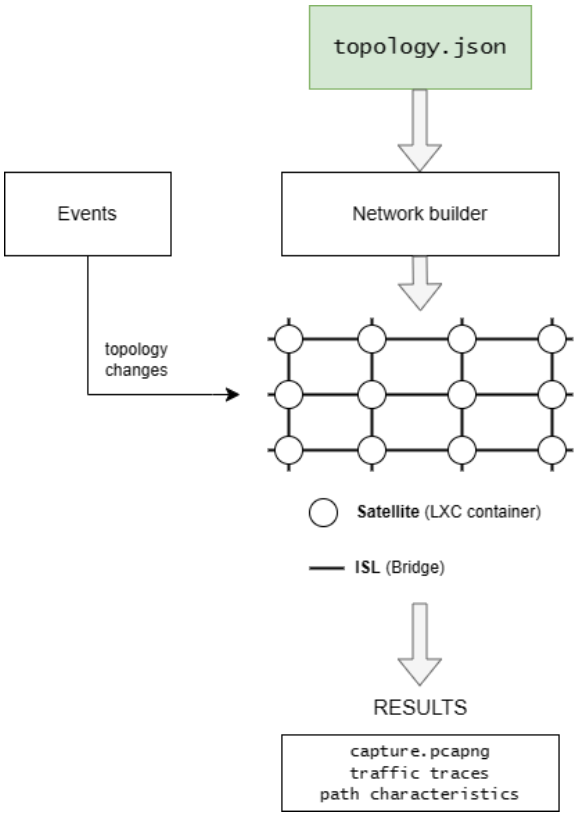

Fig. 2.23. Network emulation schematic.

Figure 2.24 shows the flowchart with the procedures to follow to create and initialize the network topology.

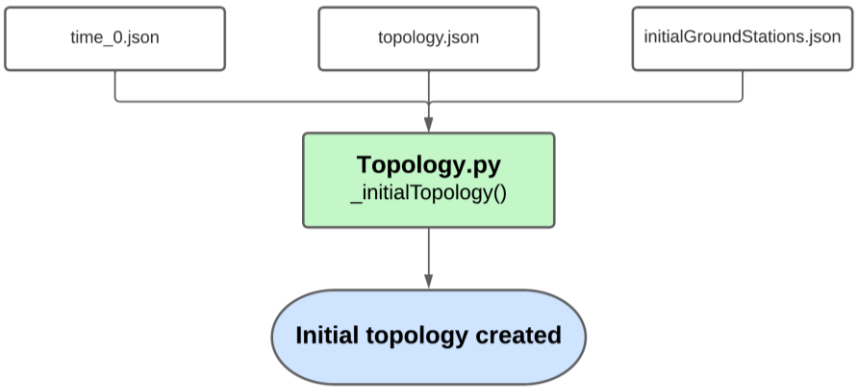

Fig. 2.24. Network topology initialization flowchart.

From the `topology.json` file, the containers and bridges necessary for network emulation are created and initiated. To perform the emulation correctly it is necessary to add the distance (delay) between each of the nodes of the network. In the case of the ISLs, the initial delay is contained in the `time_0.json` file. The delay is applied with the Command 2.20, where `host_name` is the char “R” concatenated with the node `id` plus one,

concatenated with the symbol “-”, concatenated with the interface that is part of the link (ethX), and `delay` is the one-way delay of the link in milliseconds.

```
tc qdisc add dev [host_name] root netem delay [delay]ms
```
(2.20)

In the case of GSLs, a new interface is added to the container emulating the satellite to be connected to the container emulating the ground station. Command 2.21 and Command 2.22 are needed to create the new interface in the container, where `node` is the char “R” concatenated with the node `id` plus one, `intf` is the interface that is part of the link, `isleb` is the bridge emulating the specific GSL, and `host_name` is the `node` parameter concatenated with the symbol “-” concatenated with `intf`. The delay of the GSLs is applied with the Command 2.20 from the `initialGroundStations.json` file.

```
lxc config device add [node] [intf] nic network=[isleb]
host_name=[host_name]
```
(2.21)

```
lxc exec [node] -- ip link set dev [intf] up
```
(2.22)

Figure 2.25 shows a flowchart depicting the modules that constitute the part where events are added and the files it generates.

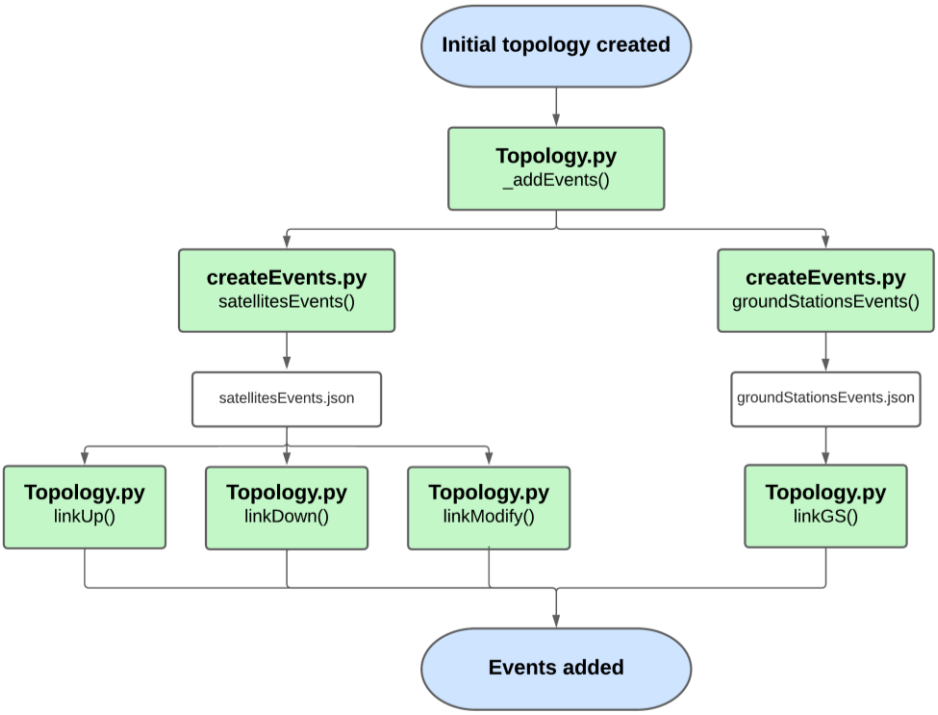

Fig. 2.25. Events flowchart.

The first step is to call `satellitesEvents()` function of the `Events` class. This method creates from the `time_i.json` files the `satellitesEvents.json` file, which contains all the satellite events that have to be programmed to happen during the network emulation. Figure 2.26 shows the process followed to obtain this file.

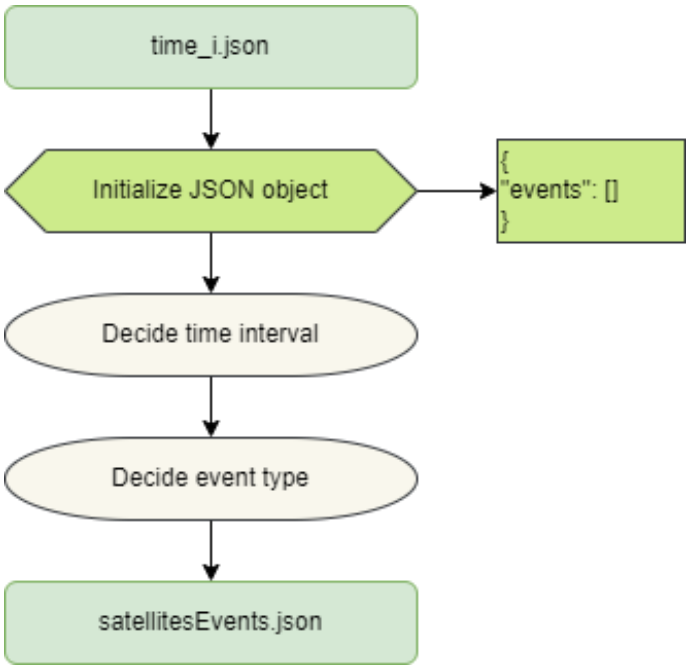

Fig. 2.26. satellitesEvents() flowchart.

The `satellitesEvents.json` file consist on a JSON object that has as property the dictionary `events`. Each event has an attribute `time`, which is the time interval that must elapse from the start of the emulation until the event is executed.

For each event, a JSON object named `actions` is defined. This object contains the integer `id` of the nodes that form the link on which the event is applied, as well as the interfaces contained in the link. It also contains another JSON object called `action`, where the type of event that takes place is defined. There are three types of events defined in the `satellitesEvents()` method:

- linkup: Event executed when recovering a link that was down.
- linkdown: Event executed when a link is no longer operational.
- linkmodifiy: Event executed when the distance between the satellites that form the link is modified.

Figure 2.27 shows the format of an ISL event description. In this example, the time to pass between each snapshot is `20` seconds. It should be noted that the event type is `linkup`. The `weight` attribute represents the one-way delay of the link in milliseconds. If the event type were `linkdown`, the `weight` attribute would not be defined.

```
{
  "time": 20.0,
  "actions": [
    {
      "node1": 9,
      "node2": 20,
      "ifa": 2,
      "ifb": 0,
      "action": [
        {
          "type": "linkup",
          "weight": "10.781447881552157"
        }
      ]
    }
  ]
}
```

Fig. 2.27. Description of ISL event in JSON format.

The second step is to call the `groundStationsEvents()` method of the `Events` class. From the `satellites_in_range_i.gml` files the `groundStationsEvents.json` file is obtained, which contains the events related to base stations. Figure 2.28 shows the process followed to obtain this file.

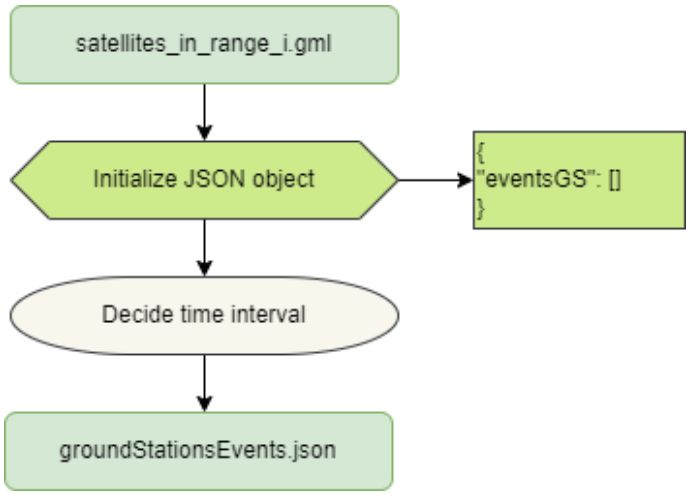

Fig. 2.28. groundStationsEvents() flowchart.

The `groundStationsEvents.json` file consist on a JSON object that has as property the dictionary `eventsGS`. The `groundStation` attribute is the earth station that forms the link with the satellite that has as integer `id` the value of the `node` attribute. Each event has an attribute `time`, which is the time interval that must elapse from the start of the emulation until the event is executed. The `weight` attribute represents the one-way delay of the link in milliseconds.

Figure 2.29 shows the format of a GSL event description.

```
{
  "groundStation": 1,
  "weight": "4.284516557117658",
  "time": 20.0,
  "node": 58
}
```

Fig. 2.29. Description of GSL event in JSON format.

In this case there is no event type in the JSON object, since there is only one defined for events related to ground stations.

- linkGS: Event executed when there is a modification of the topology in relation to the ground stations.

Figure 2.30 shows the UML class diagram of the two classes that we have implemented in order to generate the files containing the events and the methods used to schedule and execute them.

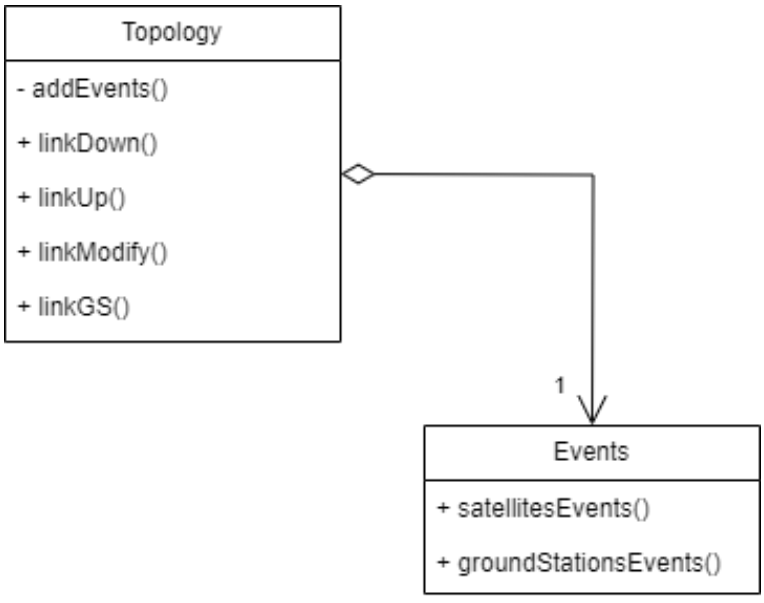

Fig. 2.30. Events UML class diagram.

2.4. Python libraries

For the emulation of the LEO satellite network, apart from the use of LXD containers and networks, several Python libraries have been required. Table 2.2 contains the most important libraries used in this project and what they have been used for. Each of these Python libraries is explained throughout this section.

Table 2.2. Python libraries used.

Pylxd	Used to simplify the creation of LXD containers and LXD networks.
APScheduler	Used to program the dynamic topology changes as events.
NetworkX	Used to read and analyze gml files coming from HypatiaSeam.
Pyshark	Used to read and analyze the captures made in each of the interfaces that emulate satellites and ground stations.

2.4.1. Pylxd

Pylxd is a Python library that simplifies the use and implementation of LXD and allows you to perform operations with containers and bridged networks, as well as to program and automate processes such as instance creation and networking.

The Pylxd Python module can be found in its GitHub repository [25] and installed with `pip`[6].

```
pip install pylxd
```
(2.23)

[6] Pip: Package management system used to install and manage software packages written in Python.

2.4.2. Advanced Python Scheduler

Advanced Python Scheduler, known as APScheduler [26][27], is a Python package that allows you to schedule your code to run later, once or multiple times. Moreover, new jobs can be added or old ones deleted at any moment.

There are two main types of schedulers. The first is the Blocking Scheduler, which blocks the main thread until the assigned job is finished. On the other side, there is the Background Scheduler, which runs the assigned jobs in the background as if they were a separate thread. The latter is the one used in this project to execute the events.

There are three types of triggers, which are responsible for determining the logic to correctly calculate the time instant at which the job is to be executed. The first is the `interval` type, which is used when you want to run a job in a specific time interval. The second is the `cron` type, used to execute code at certain times of the day. And finally, the `date` type trigger, which is the one used in this project. This trigger allows you to run the job only at a specific time instant.

To correctly determine the time instant to be executed, the `datetime` and `timedelta` functions provided by the Python datetime library are used. The `datetime` function stores the time at which the code has started, just after the initial topology has been created and started. The `timedelta` function adds to this initial time the elapsed time defined in each event.

For the installation of APScheduler and datetime libraries, the Python `pip` package is used.

```
pip install apscheduler
```
(2.24)

```
pip install datetime
```
(2.25)

2.4.3. NetworkX

NetworkX [28][29] is one of the most widely used Python libraries for working with graphs and networks. This library allows you to create, manipulate and analyze graphs efficiently.

One of the main advantages of NetworkX is its ability to work with large and complex graphs, allowing you to handle graphs of millions of nodes and links. The library has a wide variety of functionalities that allow you to create, import and export graphs in multiple formats, as well as to analyze the properties of these networks.

To install NetworkX library the Python `pip` package is used.

```
pip install networkx
```
(2.26)

2.4.4. PyShark

PyShark [30][31] is a Python library that performs the container function for Tshark, which is the command line version of Wireshark.

PyShark has two methods for analyzing capture packets. LiveCapture() allows you to capture one of the local host interfaces and then analyze the capture packets. FileCapture() imports the packets to be analyzed from a previous capture.

To install PyShark library the Python `pip` package is used.

```
pip install pyshark
```
(2.27)

CHAPTER 3. OSPFv3 ANALYSIS IN LEO SATELLITE NETWORKS

This chapter explains how OSPFv3 has been implemented in this project, and analyzes a small topology to describe how the exchange of LSAs and the OSPFv3 database works.

3.1. OSPFv3

OSPF is a free-to-use link state (LS[7]) interior routing protocol for the TCP/IP family. It was developed by the Internet Engineering Task Force (IETF) in 1988, but was not formalized until 1991.

OSPFv3 that is the OSPF version for IPv6 was published in 1999 and updated in 2008. It is specified in the RFC[8] 5340 [34] and it is the Interior Gateway Protocol (IGP) recommended by the IETF due to it fast convergence and scalability. In this version of OSPF, the main methods that characterize this protocol are maintained, such as the transmission of information by means of the flooding algorithm, the implementation of the shortest path algorithm, as well as other procedures. But there have been some changes in relation to the previous version, OSPFv2, which was used for IPv4 routing, such as modifications in the format of addresses and identifiers, as in the case of router IDs, and authentication is deleted due to IPv6 implementation.

In the OSPF protocol, each router in the domain is in charge of describing a part of the network and transmitting the LSs, through the LSAs, to its neighboring routers. LS refers to the information related to a certain network part, while LSA is the message that contains this information. LSAs are the records of a database that represents the topology of the network, each router has its own database that it is the same for all routers in the same OSPF area.

OSPF is organized in so called areas. An OSPF router belongs to a single area, except for area border routers, which belong to an area and to a special area called backbone. Due to the characteristics of this type of network, such as dynamic topology changes, no method has been found to divide these satellite networks into areas and thus take profit of this function offered by the OSPF protocol.

[7] LS protocol: Type of protocol where routers send information about the links to which they are connected to all other routers in the topology. It has a faster convergence than distance vector protocols, since in this type of protocol each router sends its entire routing table to all its neighbors, which slows down the convergence of the network.

[8] RFC: Request for Comments is a document that describes and defines Internet protocols, methods and programs. It is managed by the IETF.

The flooding algorithm for distributing LSAs ensures that all routers of the same OSPF area have an identical database except during periods when the protocol is converging.

With the database information, each router can calculate the optimal routes to each possible destination using the Dijkstra algorithm, which is the most optimal algorithm for calculating the shortest path from one node to all other nodes in the network.

OSPF routers communicate through messages directly over the IP network layer. In total, there are five different types of OSPF messages.

- Hello message (type 1): It serves to discover and maintain the relationship between neighboring OSPF routers.

- Database Description (type 2): Describe the content of the router's database.

- Link State Request (type 3): Used to request information about the link status.

- Link State Update (type 4): Contain the LSAs with the LS information. Are sent when there is an update of the information or in response to type 3 messages.

- Link State Acknowledgement (type 5): Sent in response to type 4 messages to confirm that the information has been received by the router.

By default, Hello messages are sent every 10 seconds, informing of their presence across interfaces where OSPF is enabled. This time interval, known as the Hello interval, defines the period of time between two consecutive Hello messages and it is configurable. The Hello message also contains the OSPF id of the routers used to identify neighbor relationship, and the Dead interval, which is the time that must elapse before a neighbor router is considered down. The Dead interval consists of an interval of 40 second by default, and it is also configurable.

The remaining four types of messages are dedicated to database synchronization, which is important to correctly calculate routes and avoid loops. Synchronization occurs when a new router appears or when an update occurs in any LSA. This process consists of each router sending a sequence of packets with a summary of the LSAs it contains, so that the other routers can compare them and identify the missing LSAs.

The algorithm for transmitting LSAs in this type of networks is known as reliable flooding. It starts when a router updates an LSA created by it and send the information in a type 4 message through all its interfaces. Besides, this type of network does not need a designated router.

This LSA update is received by the neighbors, who compare it with their OSPFv3 database and if it is more recent than the ones they have, they perform the following steps:

- Update the OSPFv3 database.
- Send a type 5 message to the router that sent the LSA update, through the interface on which the message was received.
- Through the rest of the interfaces, it sends a new LSA update to its neighbors.

LSA updates will continue to be sent until all acknowledgement messages are received by all interfaces of the router.

Each OSPF router creates one or more LSA messages based on its knowledge of the network. The LSA header contains the following fields to distinguish the LSAs from each other.

- Type: sets the LSA type.
- Link State ID: identifies the LSA. It varies according to the LSA type.
- Advertising router: is the OSPFv3 identifier of the router that created the LSA.
- LS Sequence number: is a 32-bit signed integer used to detect old or redundant LSAs.

In the OSPFv3 version, different types of LSAs are defined.

- Router LSA (type 1): contains information about the router interfaces.
- Network LSA (type 2): contains all routers attached to the same link.
- Inter-Area Prefix LSA (type 3): contains IPv6 prefixes of other areas.
- Inter-Area Router LSA (type 4): contains the path to an OSPFv3 router of other area.
- AS External LSA (type 5): contains external prefixes from other protocols,
- NSSA LSA (type 7): contains the route to an external prefix.
- Link LSA (type 8): contains a list of link local addresses[9].
- Intra-Area Prefix (type 9): contains the IPv6 prefixes of the area.

[9] Link local address: is defined and allows communication on a single link, outside that subnet the address is useless.

Of all these types of LSAs generated by OSPFv3 protocol, in this project we focus only on three of them: Router LSAs, Link LSAs and Intra-Area Prefix LSAs, because these are the only types of LSA that will be generated.

Router LSAs are generated one per OSPFv3 router. In these LSAs the `Link State ID` attribute is always 0.0.0.0. All routers in the same area have the same Router LSAs. For each router interface, except for the loopback interface, the type of network, the cost of sending the packet and the interface label is indicated. Finally, the router identifier (`Neighbor Router ID`) of the designated router of the network and its interface label (`Neighbor Interface ID`) are indicated.

Link LSAs are generated one per interface and neighbor connected directly to the router. This type of LSA is used to know the IPv6 link local address that can be used as next-hop. The link local address is specified for each router and interface. Not all routers have the same Link LSAs as they depend on the networks to which each router is connected.

Intra-Area Prefix LSAs are generated one for each network in the area. There are two types of networks. Stub networks, where the `Link State ID` is always 0.0.0.0, and transit networks, where this field is the interface label, 32-bit address, of the designated router of the network. As the satellite network only has point-to-point links, there are neither transit nor stub networks. These LSA types indicate the IPv6 prefixes associated with the network. All routers in the same area have the same Intra-Area Prefix LSAs.

In large networks with multiple nodes, as in the case of LEO satellite networks, two main drawbacks can arise.

- The volume of LSAs to be transmitted would be very high.
- The computation time to find the shortest path increase by 2^n, where n is the number of nodes in the topology.

3.2. Example topology

This section shows an example of a topology from which its database will be displayed and explained. Specifically, we will show the three types of LSAs that are useful in this project, which are Router LSAs, Link LSAs and Intra-Area Prefix LSAs.

Figure 3.1 shows an example of a small topology with four satellites and two ground stations, with an IP address configuration that it is explained in Chapter 4.

Network 2001:2001::/64 is a terrestrial network in which all routers emulating base stations are connected to router T, which represents a central ground station from which information is transmitted to the sinks.

Network 2001:2002::/64 is also a terrestrial network and its function is to connect the sinks, which represent the connected clients, to the central ground station.

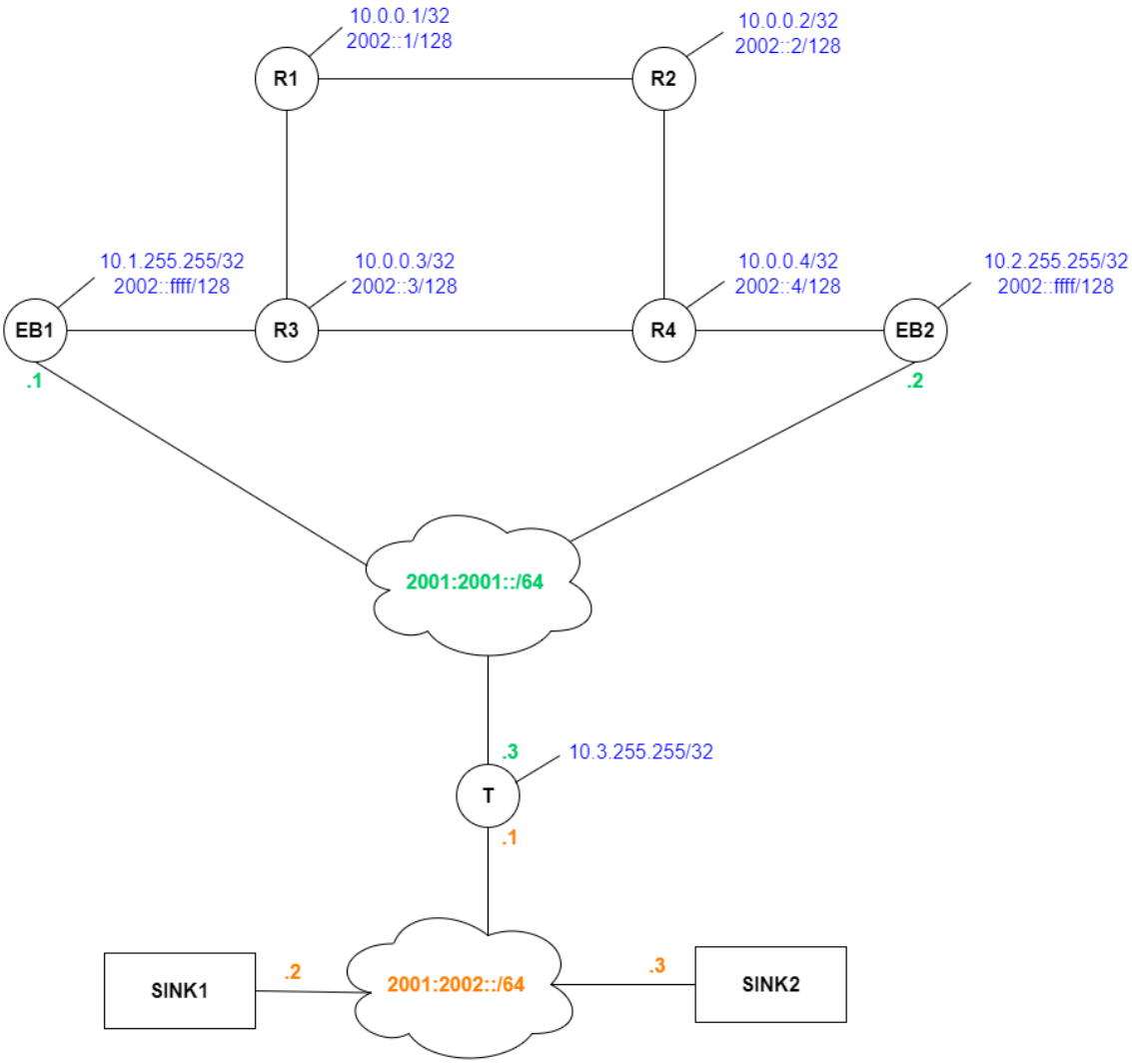

Fig. 3.1. Topology example.

Figure 3.2 shows the Router LSAs in the database. In total there are seven LSAs, one for each OSPF router emulating the four satellites, the two ground stations and the central base station.

```
Area Scoped Link State Database
(Area 0.0.0.0)

Age:  491 Type: Router
Link State ID: 0.0.0.0
Advertising Router: 10.0.0.1
LS Sequence Number: 0x80000002
CheckSum: 0x2b65 Length: 56
Duration: 00:08:10
    Bits: -------- Options: --|-|--
|-|-|--|R|-|--|E|V6
    Type: Point-To-Point Metric: 10
    Interface ID: 0.0.0.14
    Neighbor Interface ID: 0.0.0.18
    Neighbor Router ID: 10.0.0.2
    Type: Point-To-Point Metric: 10
    Interface ID: 0.0.0.16
    Neighbor Interface ID: 0.0.0.22
    Neighbor Router ID: 10.0.0.3

Age:  490 Type: Router
Link State ID: 0.0.0.0
Advertising Router: 10.0.0.2
LS Sequence Number: 0x80000002
CheckSum: 0x2661 Length: 56
Duration: 00:08:08
    Bits: -------- Options: --|-|--
|-|-|--|R|-|--|E|V6
    Type: Point-To-Point Metric: 10
    Interface ID: 0.0.0.18
    Neighbor Interface ID: 0.0.0.14
    Neighbor Router ID: 10.0.0.1
    Type: Point-To-Point Metric: 10
    Interface ID: 0.0.0.20
    Neighbor Interface ID: 0.0.0.26
    Neighbor Router ID: 10.0.0.4

Age:  481 Type: Router
Link State ID: 0.0.0.0
Advertising Router: 10.0.0.3
LS Sequence Number: 0x80000003
CheckSum: 0x8680 Length: 72
Duration: 00:07:59
    Bits: -------- Options: --|-|--
|-|-|--|R|-|--|E|V6
    Type: Point-To-Point Metric: 10
    Interface ID: 0.0.0.22
    Neighbor Interface ID: 0.0.0.16
    Neighbor Router ID: 10.0.0.1
    Type: Point-To-Point Metric: 10
    Interface ID: 0.0.0.24
    Neighbor Interface ID: 0.0.0.28
    Neighbor Router ID: 10.0.0.4
    Type: Point-To-Point Metric: 10
    Interface ID: 0.0.0.46
    Neighbor Interface ID: 0.0.0.30
    Neighbor Router ID: 10.1.255.255

Age:  477 Type: Router
Link State ID: 0.0.0.0
Advertising Router: 10.0.0.4
LS Sequence Number: 0x80000003
CheckSum: 0x3abc Length: 72
Duration: 00:07:54
    Bits: -------- Options: --|-|--
|-|-|--|R|-|--|E|V6
    Type: Point-To-Point Metric: 10
    Interface ID: 0.0.0.26
    Neighbor Interface ID: 0.0.0.20
    Neighbor Router ID: 10.0.0.2
    Type: Point-To-Point Metric: 10
    Interface ID: 0.0.0.28
    Neighbor Interface ID: 0.0.0.24
    Neighbor Router ID: 10.0.0.3
    Type: Point-To-Point Metric: 10
    Interface ID: 0.0.0.48
    Neighbor Interface ID: 0.0.0.34
    Neighbor Router ID: 10.2.255.255

Age:  482 Type: Router
Link State ID: 0.0.0.0
Advertising Router: 10.1.255.255
LS Sequence Number: 0x80000004
CheckSum: 0x2b15 Length: 56
Duration: 00:07:54
    Bits: -------- Options: --|-|--
|-|-|--|R|-|--|E|V6
    Type: Point-To-Point Metric: 10
    Interface ID: 0.0.0.30
    Neighbor Interface ID: 0.0.0.46
    Neighbor Router ID: 10.0.0.3
    Type: Transit-Network Metric: 10
    Interface ID: 0.0.0.32
    Neighbor Interface ID: 0.0.0.38
    Neighbor Router ID: 10.3.255.255

Age:  482 Type: Router
Link State ID: 0.0.0.0
Advertising Router: 10.2.255.255
LS Sequence Number: 0x80000005
CheckSum: 0x0c27 Length: 56
Duration: 00:07:58
    Bits: -------- Options: --|-|--
|-|-|--|R|-|--|E|V6
    Type: Point-To-Point Metric: 10
    Interface ID: 0.0.0.34
    Neighbor Interface ID: 0.0.0.48
    Neighbor Router ID: 10.0.0.4
    Type: Transit-Network Metric: 10
    Interface ID: 0.0.0.36
    Neighbor Interface ID: 0.0.0.38
    Neighbor Router ID: 10.3.255.255

Age:  474 Type: Router
Link State ID: 0.0.0.0
Advertising Router: 10.3.255.255
LS Sequence Number: 0x80000005
CheckSum: 0x2f03 Length: 56
Duration: 00:07:50
    Bits: -------- Options: --|-|--
|-|-|--|R|-|--|E|V6
    Type: Transit-Network Metric: 10
    Interface ID: 0.0.0.38
    Neighbor Interface ID: 0.0.0.38
    Neighbor Router ID: 10.3.255.255
    Type: Transit-Network Metric: 10
    Interface ID: 0.0.0.40
    Neighbor Interface ID: 0.0.0.40
    Neighbor Router ID: 10.3.255.255
```

Fig. 3.2. Router LSAs.

From the Router LSA marked in blue, the part of the topology shown in Figure 3.3 can be deduced.

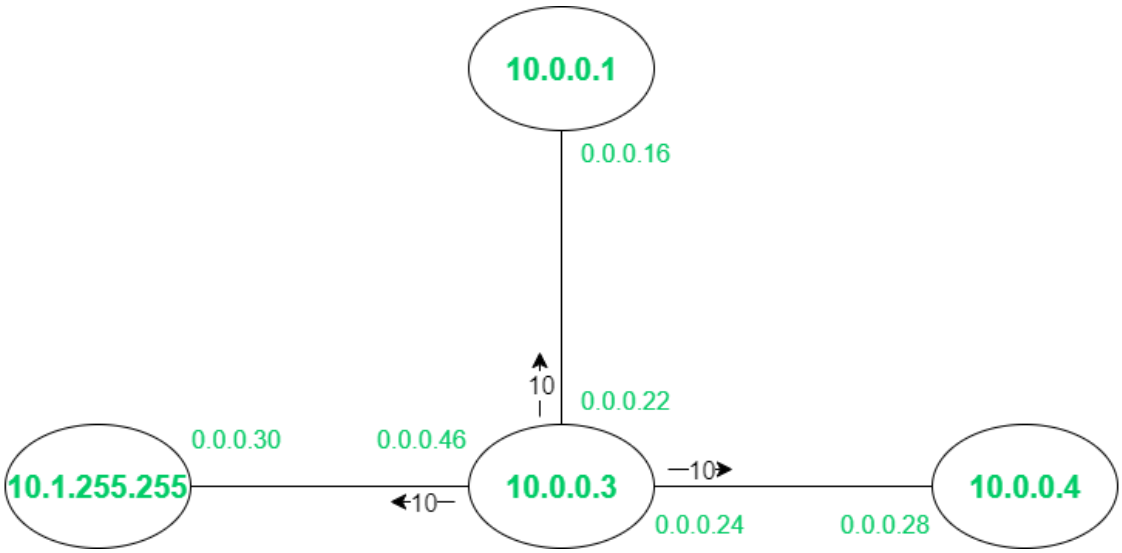

Fig. 3.3. Topology segment from Router LSA.

The router that generates this Router LSA has as ID the IPv4 address 10.0.0.3 (`Advertising Router`), which corresponds to R3. This router is connected through interface 0.0.0.22 (`Interface ID`) to the router with ID 10.0.0.1 (`Neighbor Router ID`), which corresponds to R1, through interface 0.0.0.16 (`Neighbor Interface ID`). Through the interface 0.0.0.24 is connected with the router with ID 10.0.0.4, which corresponds to R4, through interface 0.0.0.28. Finally, through the interface 0.0.0.46 is connected with the node with ID 10.1.255.255, which corresponds to EB1, trough the interface 0.0.0.30. The three links to which R3 belongs are point-to-point (`Type`) with a path cost of 10 (`Metric`).

Figure 3.4 shows the Link LSAs in the R1 database. As can be seen, there are two LSAs for each link directly connected to this router, because they are point-to-point links. Through `eth0` interface it is connected to R2, which has as ID 10.0.0.2 (`Advertising Router`), and through the `eth1` interface it is connected to R3, which has as ID 10.0.0.3.

```
            I/F Scoped Link State Database (I/F eth0 in Area 0.0.0.0)

Age:  541 Type: Link
Link State ID: 0.0.0.14
Advertising Router: 10.0.0.1
LS Sequence Number: 0x80000001
CheckSum: 0x7b81 Length: 44
Duration: 00:09:00
     Priority: 1 Options: --|-|--|-|-|--|R|-|--|E|V6
     LinkLocal Address: fe80::216:3eff:fe77:854e
     Number of Prefix: 0

Age:  539 Type: Link
Link State ID: 0.0.0.18
Advertising Router: 10.0.0.2
LS Sequence Number: 0x80000001
CheckSum: 0x3dc8 Length: 44
Duration: 00:08:57
     Priority: 1 Options: --|-|--|-|-|--|R|-|--|E|V6
     LinkLocal Address: fe80::216:3eff:fe76:6264
     Number of Prefix: 0


         I/F Scoped Link State Database (I/F eth1 in Area 0.0.0.0)

Age:  541 Type: Link
Link State ID: 0.0.0.16
Advertising Router: 10.0.0.1
LS Sequence Number: 0x80000001
CheckSum: 0x42e2 Length: 44
Duration: 00:09:00
     Priority: 1 Options: --|-|--|-|-|--|R|-|--|E|V6
     LinkLocal Address: fe80::216:3eff:fe6a:2591
     Number of Prefix: 0

Age:  537 Type: Link
Link State ID: 0.0.0.22
Advertising Router: 10.0.0.3
LS Sequence Number: 0x80000001
CheckSum: 0xf3d8 Length: 44
Duration: 00:08:54
     Priority: 1 Options: --|-|--|-|-|--|R|-|--|E|V6
     LinkLocal Address: fe80::216:3eff:fe34:7ebe
     Number of Prefix: 0
```

Fig. 3.4. Link LSAs.

From Link LSAs marked in blue, the part of the topology shown in Figure 3.5 can be deduced.

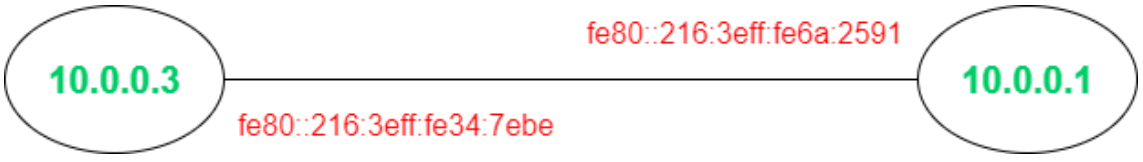

Fig. 3.5. Topology segment from Link LSAs.

From these Link LSAs, the link local addresses of each interface of the link formed between R3 and R1 has been obtained. The link local address of the

interface of R3 is fe80::216:3eff:fe34:7ebe and the link local address of the interface of R1 is fe80::216:3eff:fe6a:2591.

Figure 3.6 shows the Intra-Prefix LSAs in the database. As in the case of Router LSAs, there are also a total of seven Intra-Prefix LSAs in the database, one per each OSPFv3 router.

```
Age:  640 Type: Intra-Prefix
Link State ID: 0.0.0.0
Advertising Router: 10.0.0.1
LS Sequence Number: 0x80000003
CheckSum: 0x2f0a Length: 52
Duration: 00:10:39
     Number of Prefix: 1
     Reference:  Router  Id:  0.0.0.0
Adv: 10.0.0.1
     Prefix Options: --|--|--|--|--
     Prefix: 2002::1/128
     Metric: 10

Age:  641 Type: Intra-Prefix
Link State ID: 0.0.0.0
Advertising Router: 10.0.0.2
LS Sequence Number: 0x80000003
CheckSum: 0x59dc Length: 52
Duration: 00:10:32
     Number of Prefix: 1
     Reference:  Router  Id:  0.0.0.0
Adv: 10.0.0.2
     Prefix Options: --|--|--|--|--
     Prefix: 2002::2/128
     Metric: 10

Age:  630 Type: Intra-Prefix
Link State ID: 0.0.0.0
Advertising Router: 10.0.0.3
LS Sequence Number: 0x80000004
CheckSum: 0x81b0 Length: 52
Duration: 00:10:28
     Number of Prefix: 1
     Reference:  Router  Id:  0.0.0.0
Adv: 10.0.0.3
     Prefix Options: --|--|--|--|--
     Prefix: 2002::3/128
     Metric: 10

Age:  626 Type: Intra-Prefix
Link State ID: 0.0.0.0
Advertising Router: 10.0.0.4
LS Sequence Number: 0x80000004
CheckSum: 0xab83 Length: 52
Duration: 00:10:23
     Number of Prefix: 1
     Reference:  Router  Id:  0.0.0.0
Adv: 10.0.0.4
     Prefix Options: --|--|--|--|--
     Prefix: 2002::4/128
     Metric: 10

Age:  631 Type: Intra-Prefix
Link State ID: 0.0.0.0
Advertising Router: 10.1.255.255
LS Sequence Number: 0x80000005
CheckSum: 0x0533 Length: 52
Duration: 00:10:23
     Number of Prefix: 1
     Reference:  Router  Id:  0.0.0.0
Adv: 10.1.255.255
     Prefix Options: --|--|--|--|--
     Prefix: 2002::ffff/128
     Metric: 10

Age:  631 Type: Intra-Prefix
Link State ID: 0.0.0.0
Advertising Router: 10.2.255.255
LS Sequence Number: 0x80000004
CheckSum: 0x0b2c Length: 52
Duration: 00:10:27
     Number of Prefix: 1
     Reference:  Router  Id:  0.0.0.0
Adv: 10.2.255.255
     Prefix Options: --|--|--|--|--
     Prefix: 2002::ffff/128
     Metric: 10

Age:  634 Type: Intra-Prefix
Link State ID: 0.0.0.38
Advertising Router: 10.3.255.255
LS Sequence Number: 0x80000001
CheckSum: 0x37e5 Length: 44
Duration: 00:10:28
     Number of Prefix: 1
     Reference: Network Id: 0.0.0.38
Adv: 10.3.255.255
     Prefix Options: --|--|--|--|--
     Prefix: 2001:2001::/64
     Metric: 0

Age:  623 Type: Intra-Prefix
Link State ID: 0.0.0.40
Advertising Router: 10.3.255.255
LS Sequence Number: 0x80000002
CheckSum: 0x4bcb Length: 44
Duration: 00:10:20
     Number of Prefix: 1
     Reference: Network Id: 0.0.0.40
Adv: 10.3.255.255
     Prefix Options: --|--|--|--|--
     Prefix: 2001:2002::/64
     Metric: 0
```

Fig. 3.6. Intra-Area Prefix LSAs.

From the Intra-Area Prefix LSA marked in blue, the part of the topology shown in Figure 3.7 can be deduced.

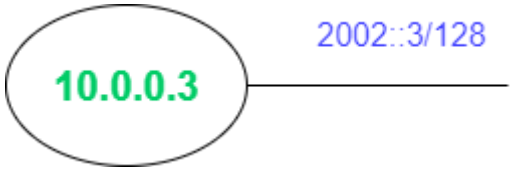

Fig. 3.7. Topology segment from Intra-Area Prefix LSA.

From this Intra-Area Prefix LSA it is deduced that R3 is associated to the prefix 2002::3/128.

Figure 3.8 shows the IPv6 Routing Information Base (RIB) of R1. The following command is used to obtain this table.

```
lxc exec [container] --vtysh -c 'show ipv6 route'
```
(3.1)

This table shows the information of the paths that R1 has to take to reach any router.

```
Codes: K - kernel route, C - connected, S - static, R - RIPng,
       O - OSPFv3, I - IS-IS, B - BGP, N - NHRP, T - Table,
       v - VNC, V - VNC-Direct, A - Babel, F - PBR,
       f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup
       t - trapped, o - offload failure

O>*  2001:2001::/64  [110/30]  via  fe80::216:3eff:fe34:7ebe,  eth1,  weight  1,
00:06:37
O>*  2001:2002::/64  [110/40]  via  fe80::216:3eff:fe34:7ebe,  eth1,  weight  1,
00:06:37
O   2002::1/128 [110/10] is directly connected, lo, weight 1, 00:06:37
C>* 2002::1/128 is directly connected, lo, 00:07:03
O>*  2002::2/128  [110/20]  via  fe80::216:3eff:fe76:6264,  eth0,  weight  1,
00:06:37
O>*  2002::3/128  [110/20]  via  fe80::216:3eff:fe34:7ebe,  eth1,  weight  1,
00:06:37
O>*  2002::4/128  [110/30]  via  fe80::216:3eff:fe34:7ebe,  eth1,  weight  1,
00:06:37
  *                           via  fe80::216:3eff:fe76:6264,  eth0,  weight  1,
00:06:37
O>*  2002::ffff/128  [110/30]  via  fe80::216:3eff:fe34:7ebe,  eth1,  weight  1,
00:06:37
C * fe80::/64 is directly connected, eth1, 00:07:03
C>* fe80::/64 is directly connected, eth0, 00:07:03
```

Fig. 3.8. R1 IPv6 RIB.

Through OSPFv3 (○), R1 has learned and selected (>) paths to reach prefixes 2001:2001::/64, 2001:2002::/64, 2002::3/128, 2002::4/128 and 2002::ffff/128, through the link local address fe80::216:3eff:fe34:7ebe corresponding to the interface of the link it forms with R3. Through the link local address fe80::216:3eff:fe76:6264, which corresponds to R2, R1 reaches the 2002::2/128 prefix. Finally, to reach the prefix 2002::1/128 it learns the route via OSPFv3, but does not select it since R1 is directly connected.

Chapter 4. Network Satellite Emulation Analysis and Evaluation

This chapter explains the OSPFv3 configuration that has been established in the different routers that emulate satellites and ground stations, and analyzes the results obtained from different tests performed, including traffic captures, and characteristics and traces of the path from one router to the nearest ground stations. Figure 4.1 shows the analyzed scenario.

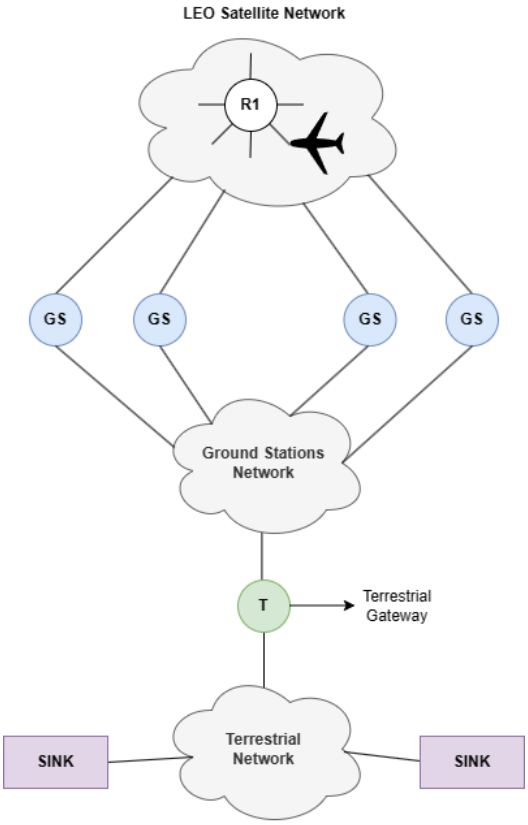

Fig. 4.1. Scenario analyzed.

All tests are done by generating traffic on one of the satellites, the emulation time covers a full lap of the Iridium constellation, which satellite is selected is irrelevant. We have chosen the satellite labeled R1, which starts in the equator. This satellite has a total of six interfaces. Two to connect to satellites that are in the same plane (intra-plane ISLs), one to connect to a satellite in the contiguous plane (inter-satellite ISL), one is in disuse because R1 is in on of the two planes between which the seam is located, another one is the loopback interface to assign IP addresses. Finally, the sixth interface is used to connect to the aircraft (traffic generator) that communicates with the airspace control centers through the ground stations.

The satellite network analyzed has been the Iridium constellation, which consists of 66 satellites and 4 ground stations. These base stations are connected to a network through which they communicate with another station that acts as a terrestrial gateway. This station is responsible for distributing information to the different airspace control centers (sinks). The analysis is a worst-case scenario, i.e., a satellite will not be always sending traffic to Earth, as an airplane will not always be connected to the same satellite during its flight. Anyway, this setup can be interpreted as a satellite that at each time has traffic to route, independently if it is always originated at the same source (plane) or the source changes at any time.

4.1. OSPFv3 Configuration

Figure 4.2 shows the `frr` JSON object of one of the satellites, which is part of the `routers` dictionary in the `topology.json` file, used to set the OSPFv3 configuration on each router.

```
"frr": {
  "interface": {
    "lo": {
      "ip_address": [
        "10.0.0.5/32"
      ],
      "ipv6_address": [
        "2002::5/128"
      ],
      "ospf6_area": "0.0.0.0",
      "ospf6_passive": true
    },
    "eth0": {
      "ospf6_area": "0.0.0.0",
      "ospf6_network": "point-to-point",
      "ospf6_dead_interval": 2,
      "ospf6_hello_interval": 1
    },
                    .
                    .
                    .
  },
  "router": {
    "ospf6": {}
  }
}
```

Fig. 4.2. FRR object definition.

In the case of routers, the loopback interfaces are configured with an IPv4 address in the 10.0.X.X/32 range with 0 < X < 255. In OSPF all the routers are unique identified by a 32 bits number in dotted quad format. This identifier can be configured manually. In OSPFv3 if a router has an IPv4 address configured in its loopback interface, this address becomes also the OSPFv3 router identifier. An IPv6 address is also configured in the 2002::X/128 range with 0 < X < 255, which will be used by OSPFv3.

In the case of ground stations, the IPv4 address is in the 10.X.255.255/32 range with 0 < X < 255. In the case of IPv6 addressing, all base stations have the 2002::ffff/128 anycast address assigned to them.

All nodes that are included of the satellite network are part of the same OSPFv3 area, which is the 0.0.0.0 area. As already mentioned, it is difficult to divide this type of network into areas. If the area 0.0.0.0 is determined as a grid within the non-backbone areas are located, the problem of boundary selection arises. It was not clear whether it was possible to guarantee that routers would always have connectivity to the border, and if they did, it could happen that it would take many hops to reach a router that might be only two hops away.

The loopback interface in all OSPFv3 routers is configured as passive, which means that no messages are sent through this interface, but the network to which it belongs is advertised in the messages sent by the other interfaces of the router.

Bridges emulating ISLs and GSLs are configured as a point-to-point links. Doing this, designated router selection is avoided and no Network LSAs are generated.

The Hello interval and the Dead interval have been modified. Specifically, the Hello interval has been set to 1 second and the Dead interval to 2 seconds. This implies that Hello messages will be sent every second, and that an OSPF neighboring router will be dropped if after two seconds there is no information from it. Both values are the minimum that can be set.

Finally, in the `router` JSON object, the protocol that each router has to apply is established. In this case `ospf6` is set, which indicates OSPFv3.

4.2. Results

In order to analyze and study the performance of LEO satellite networks, specifically the Iridium constellation, the following tests have been done using the created network emulator.

- Distribution of convergence time values of OSPFv3 Router LSAs by analyzing traffic captures (specifically captures in pcapng format) to plot the histogram.

- Average number of events between snapshots.

- Histogram to quantify the number of hops between a satellite and the nearest base station.

- Traffic generation is done using Distributed Internet Traffic Generator (D-ITG) to obtain delay, throughput, jitter and average losses metrics.

  To calculate the jitter, the D-ITG generator applies 4.1 and 4.2 formulas.

$$J_i = (S_i - T_i) - (S_{i-1} - T_{i-1}) = (S_i - S_{i-1}) - (T_i - T_{i-1}) \quad \textbf{(4.1)}$$

$$AverageJitter = \frac{\sum_{i=1}^{n} |J_i|}{n} \quad \textbf{(4.2)}$$

  The previous formulas are based on a traffic sequence as shown in Figure 4.3.

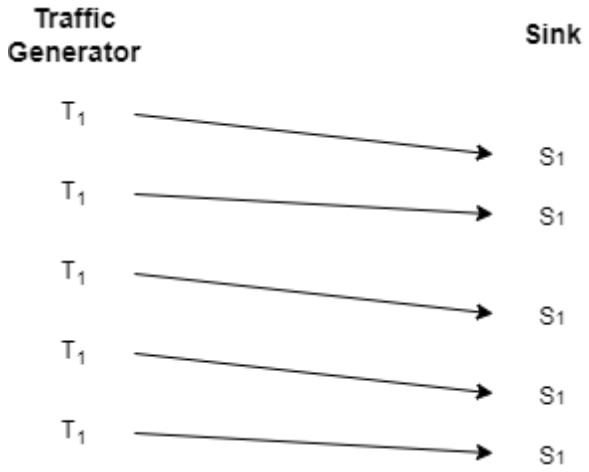

Fig. 4.3. Jitter calculation scheme.

To realize the study, emulations of two hours were performed, which is equivalent to more than one full lap (100 min) of the Iridium constellation, with different interval times between snapshots. In fact, we have analyzed for time intervals of 20 and 5 seconds with two different transmission configurations.

- Configuration 1: 50 packets of 160 bytes per second. Equivalent to a theoretical throughput of 64 kbps. VoIP traffic.

- Configuration 2: 4000 packets of 160 bytes per second. Equivalent to a theoretical throughput of 5.12 Mbps. With such a high packet

transmission rate, we have found a limitation on the D-ITG generator. In this configuration the true packet sending rate drops from 4000 pkt/s to 3100 pkt/s.

In the following, we present the results for each of them.

4.2.1. Δt = 20 s

In this section, the time difference between snapshots is set to 20 seconds, therefore, 360 consecutives snapshots have been emulated to complete the two hours.

Figure 4.4 shows the convergence time distribution of OSPFv3 Router LSAs for this time interval between snapshots. As can be observed, the most predominant convergence time interval, and therefore the time it takes for the routers to learn about the changes that have occurred in the topology, is between 0 and 1 second. The average convergence time is equal to 2.3529 seconds.

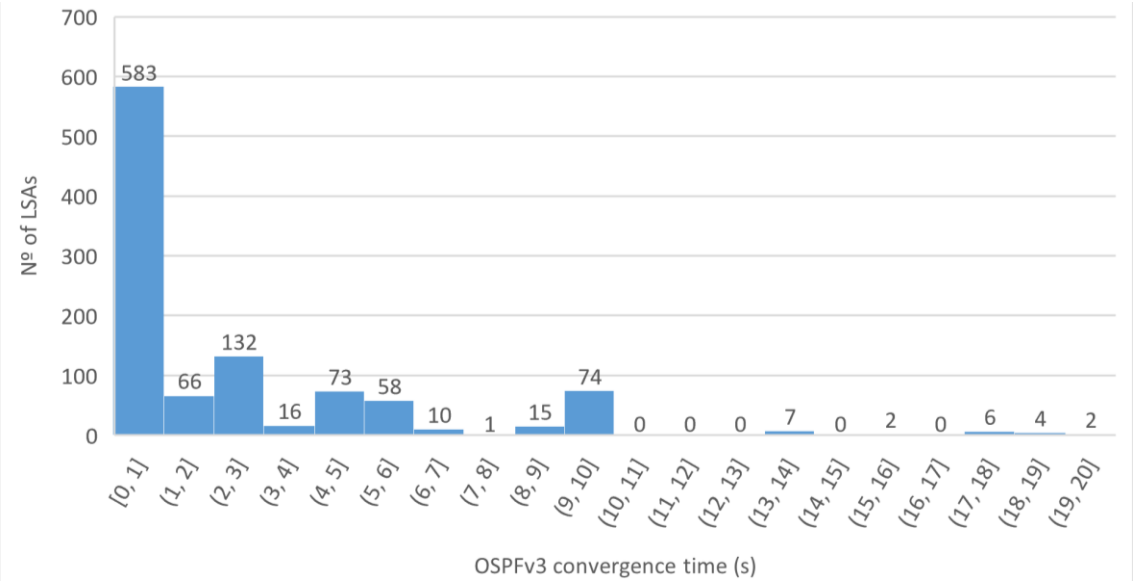

Fig. 4.4. OSPFv3 convergence time histogram. Δt = 20 s.

From `satelliteEvents.json` file and `groundStationsEvents.json` file, it has been calculated an average of 1 event per snapshot. These events include links disconnection and connections, changes in ISL delay and GSL events.

Figure 4.5 shows the number of hops it takes R1 to reach the nearest ground station, in terms of OSPFv3 cost, in each snapshot. With this spacing between snapshots, values of 3, 4, and 5 hops are dominant.

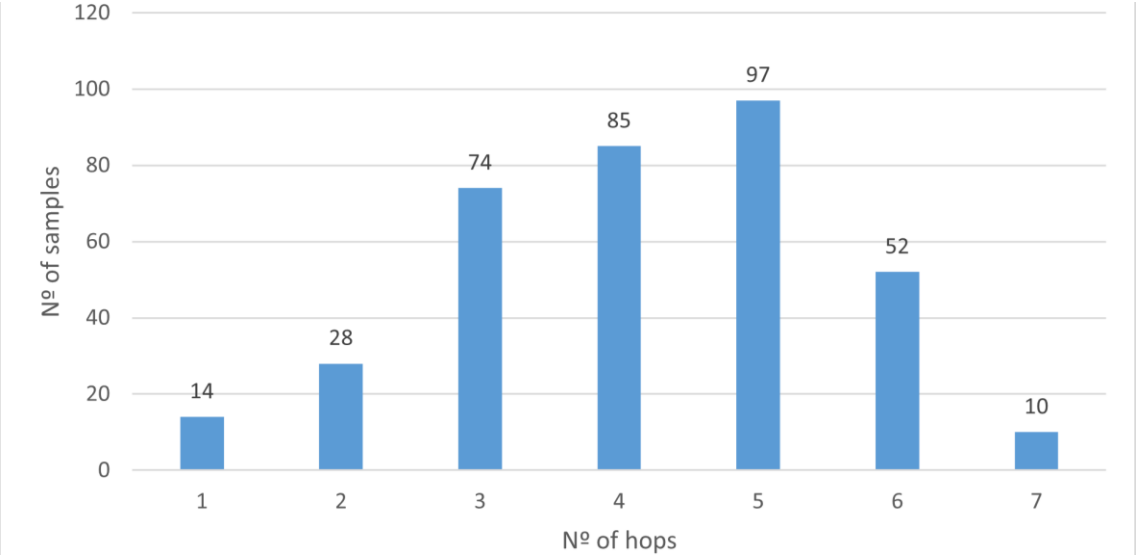

Fig. 4.5. Number of hops histogram. Δt = 20 s.

Figure 4.6 shows the satellite ground track during the two hours of emulation. The satellite starts in northern Brazil, at the equator (latitude 0º), and ends in northern Canada. The trajectory of the satellite affects the number of hops the satellite would have to make to reach the ground stations, as the actual distance between them varies.

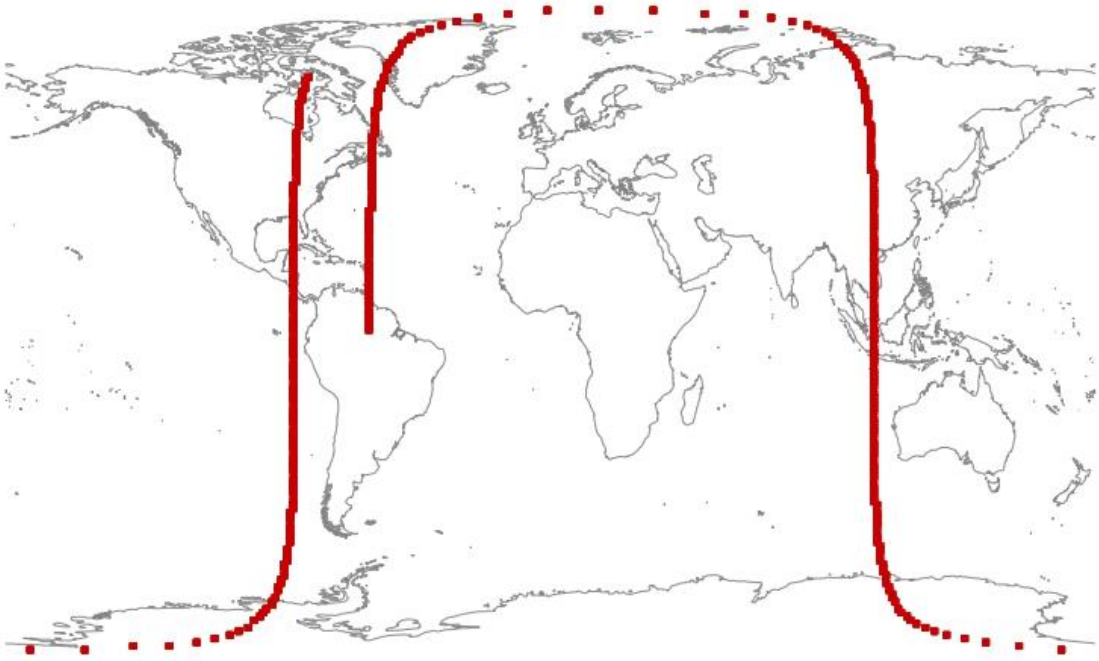

Fig. 4.6. Satellite ground track. Δt = 20 s.

Figure 4.4, Figure 4.5 and Figure 4.6 are independent of the transmission configuration used. The metrics extracted from the D-ITG, that depend on the traffic generator configuration, are show below.

4.2.1.1. Configuration 1

Table 4.1 shows a summary made by D-ITG of the metrics obtained during traffic generation.

Table 4.1. D-ITG summary. Δt = 20 s. Configuration 1.

Total time	7200 s
TRANSMISSION METRICS	
Total packets	357552 pkt
Average packet rate	49.66 pkt/s
RECEPTION METRICS	
Total packets	356733
Minimum delay	4.168 ms
Maximum delay	134.932 ms
Average delay	45.624 ms
Average jitter	324 μs
Delay standard deviation	17.427 ms
Average packet rate	49.5463 pkt/s
Packets dropped	819 (0.23 %)

Figure 4.7 shows a comparison between the evolution of the packet delay and the number of hops at each time step. The shapes of the two graphs are practically the same, showing that the greater the number of hops, the greater the packet delay. It is also worth to mention that packets with zero delay are lost packets.

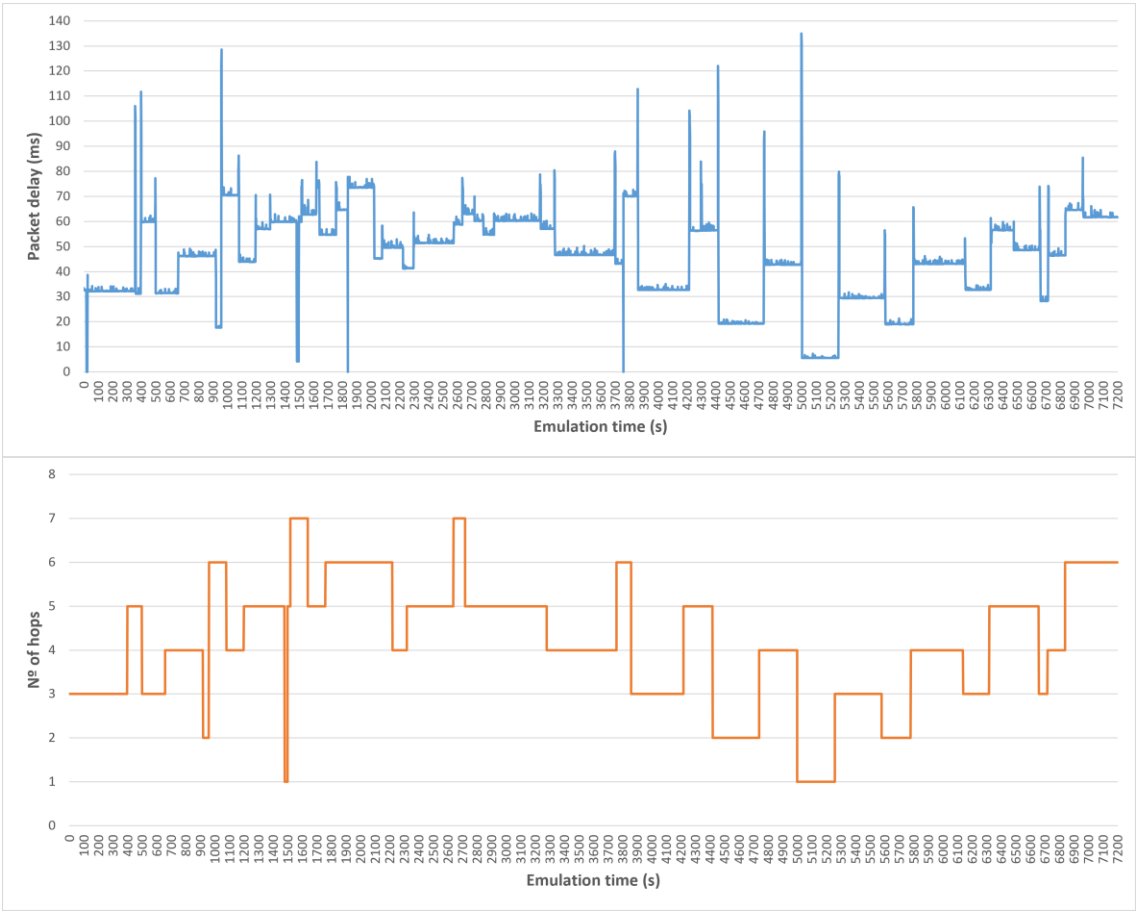

Fig. 4.7. Packet delay and number of hops. Δt = 20 s. Configuration 1.

Figure 4.8 shows the delay histogram of received packets. The packet delay is concentrated between (15, 20] ms and (70, 75] ms, with the (30, 35] ms and (45, 50] ms intervals dominating. From the D-ITG metrics summary, Table 4.1, we obtain that the average delay in this scenario is equal to 45.624 ms.

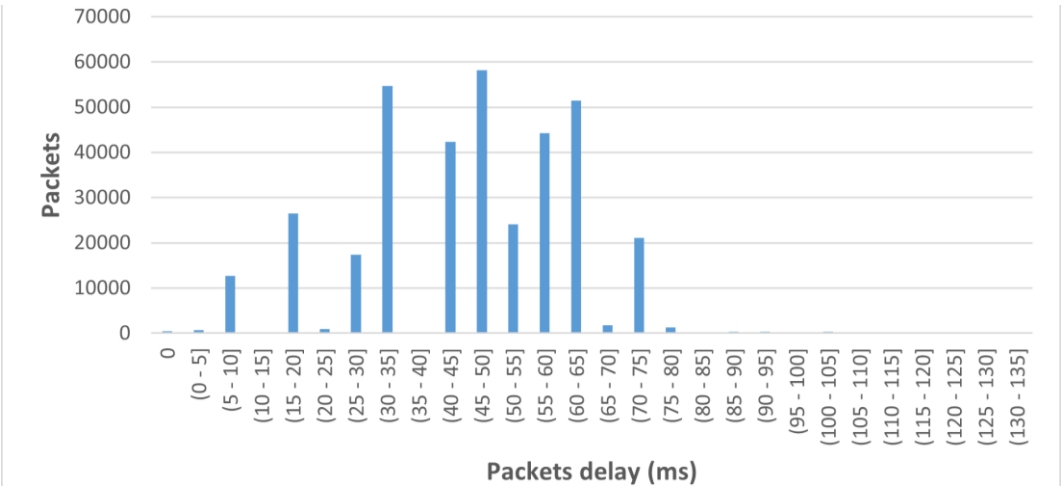

Fig. 4.8. Packet delay histogram. Δt = 20 s. Configuration 1.

Figure 4.9 shows the packet loss rate and compares it to the latitude at which the satellite is located at the same time instant. There are 3 zones or points where all the packets are lost. The first located at 20 seconds is due to the initialization of the GSL events. One point located at 1300 seconds, where the latitude is higher than 60º, an another at 3760 seconds, where takes place the disconnections of the inter-plane ISL, due to satellite is approaching to -60º.

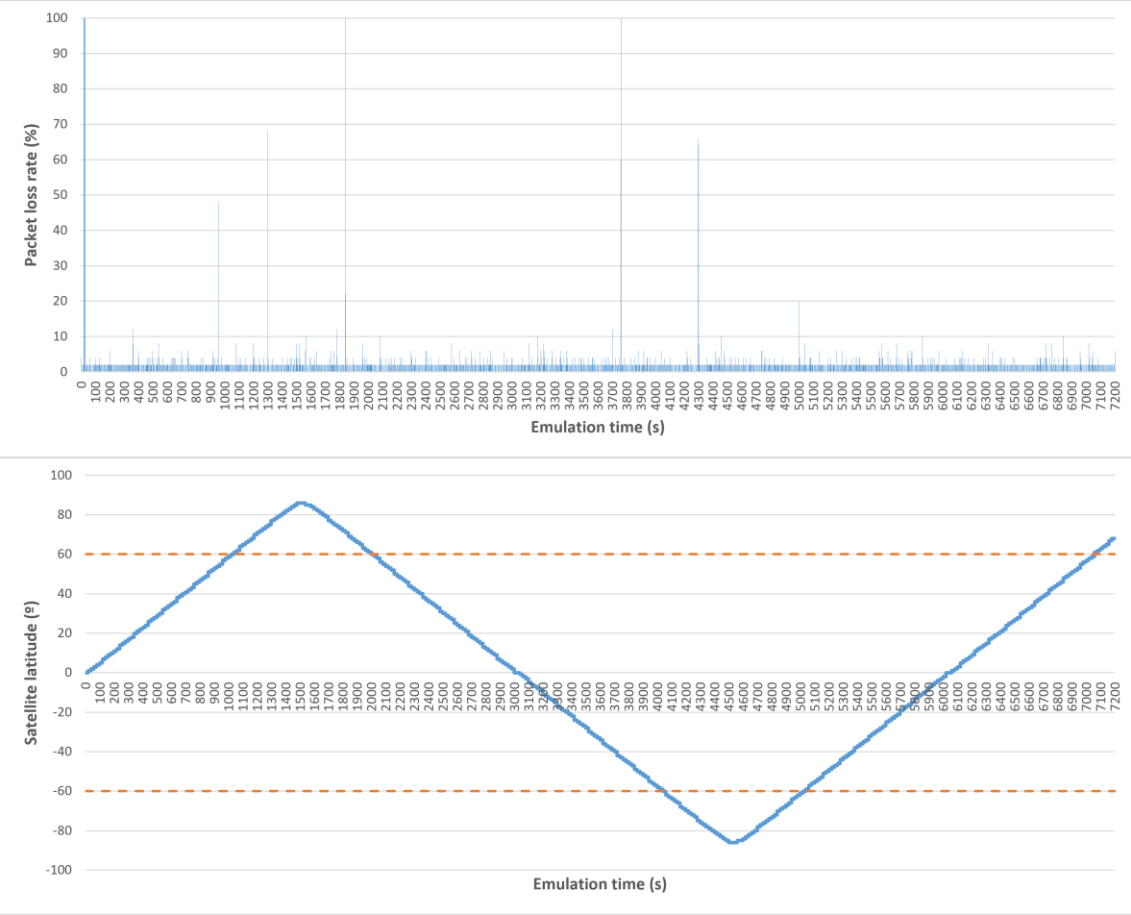

Fig. 4.9. Packet loss rate and satellite orbit latitudes. Δt = 20 s. Configuration 1.

4.2.1.2. Configuration 2

Table 4.2 shows a summary made by D-ITG of the metrics obtained during traffic generation.

Table 4.2. D-ITG summary. Δt = 20 s. Configuration 2.

Total time	7200 s
TRANSMISSION METRICS	
Total packets	22320000 pkt
Average packet rate	3100 pkt/s
RECEPTION METRICS	
Total packets	22266443
Minimum delay	4.162 ms
Maximum delay	133.881 ms
Average delay	46.024 ms
Average jitter	92 µs
Delay standard deviation	17.152 ms
Average packet rate	3092.5615 pkt/s
Packets dropped	53557 (0.24 %)

Figure 4.10 shows a comparison between the evolution of the packet delay and the number of hops at each time step. As in the previous configuration, it is still maintained that the higher the number of hops between the satellite tracked and the ground station, the higher the packet delay at the same instant time.

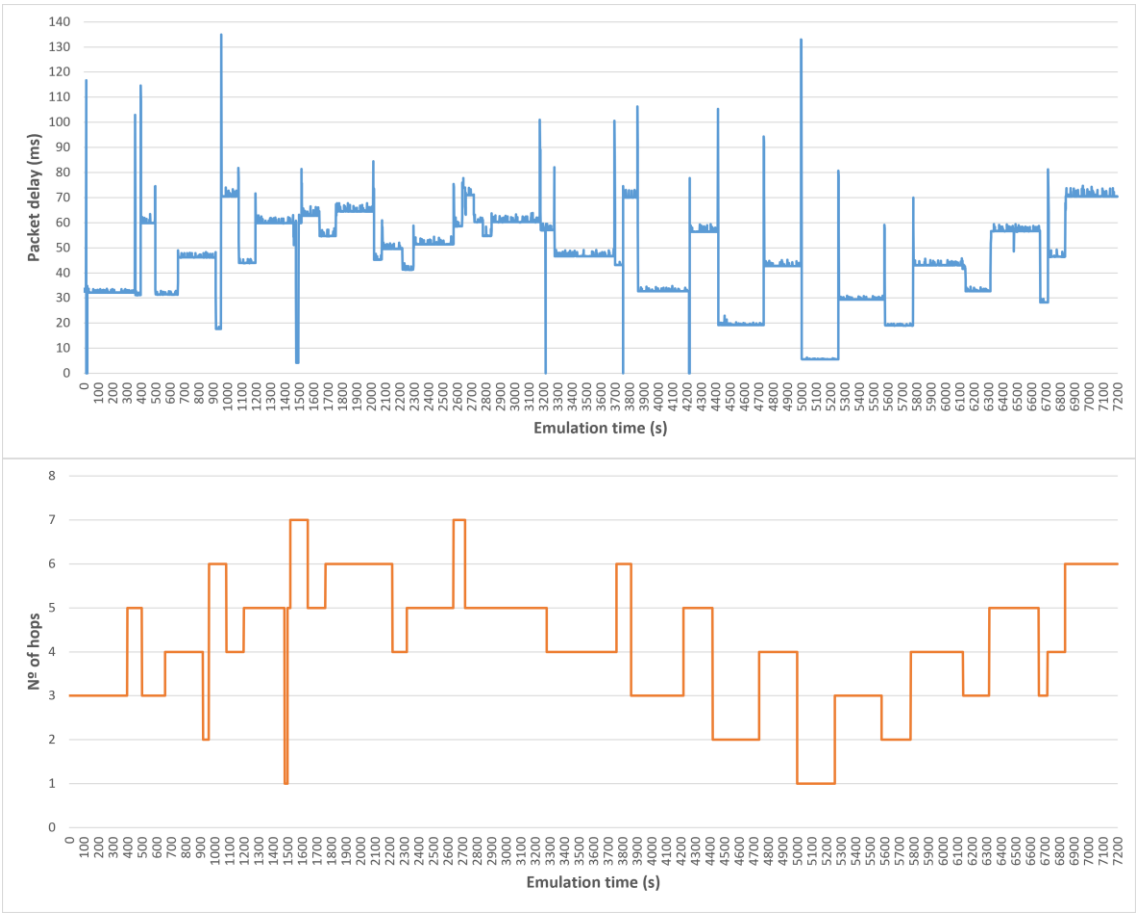

Fig. 4.10. Packet delay and number of hops. Δt = 20 s. Configuration 2.

Figure 4.11 shows the delay histogram of received packets. The same intervals predominate. The average delay is equal to 46.024 ms. This figure shows that the transmission configuration parameters does not affect the packet delay.

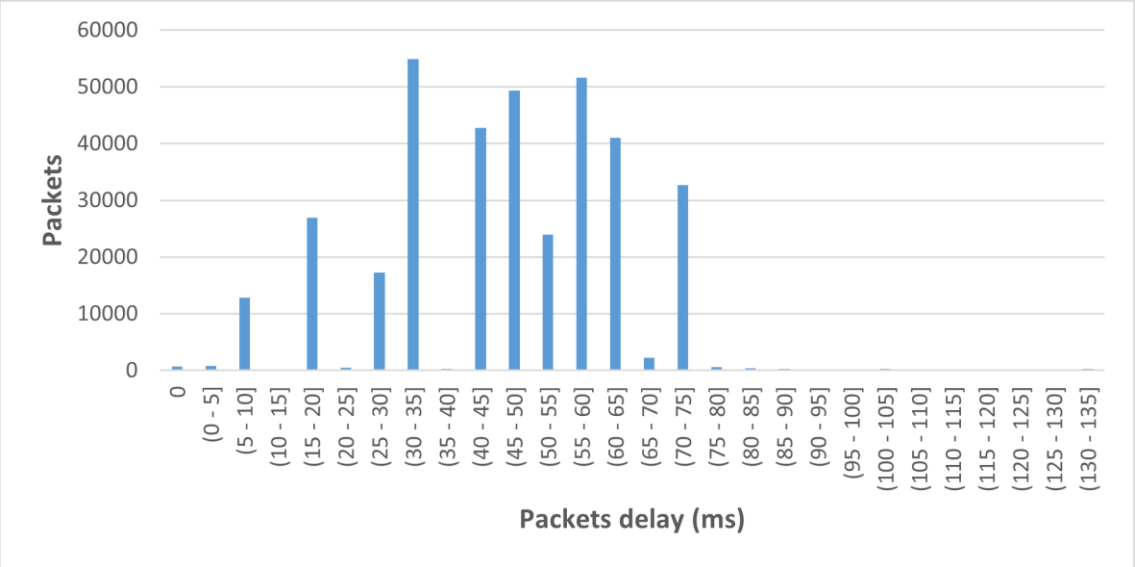

Fig. 4.11. Packet delay histogram. Δt = 20 s. Configuration 2.

Figure 4.12 shows the packet loss rate and compares it to the latitude at which the satellite is located at the same time instant. There are 4 zones or points where all the packets are lost. As in the previous test, the first critic point is due to the starting of the GSL events. The point located at 3220 seconds is due to the fact that it is one of the intervals where there are more hops to reach the base station. At 3760 seconds the inter-plane ISL disconnections occurs, since it is approaching the -60º latitude. And finally, there is a total loss of packets at 4200 seconds, a point that is in the zone below -60º latitude.

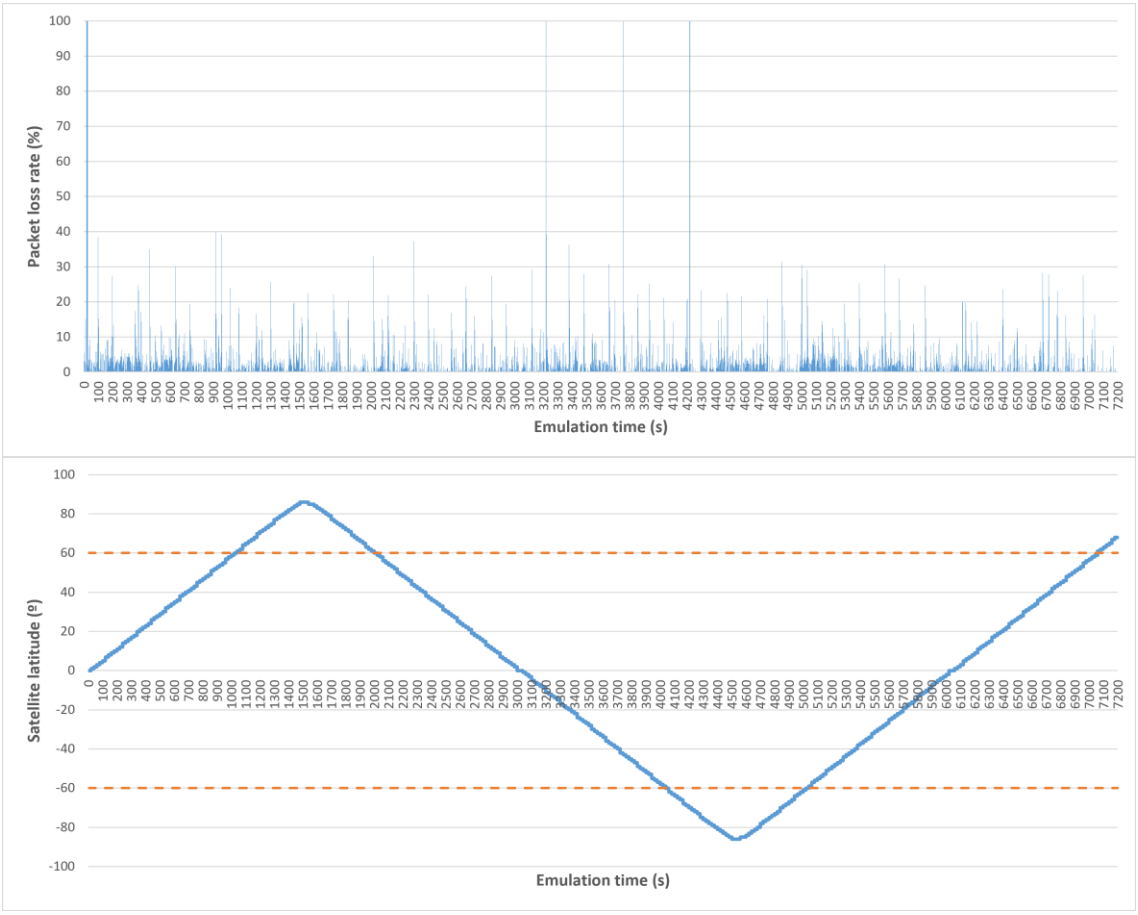

Fig. 4.12. Packet loss rate and satellite orbit latitudes. Δt = 20 s. Configuration 2.

4.2.2. Δt = 5 s

In this section, the time difference between snapshots is set to 5 seconds, therefore, 1440 consecutives snapshots have been emulated to complete the two hours.

Figure 4.13 shows the convergence time distribution of OSPFv3 Router LSAs for this time interval between snapshots. As can be observed, the most frequent time it takes for this type of LSAs to converge, and therefore the time it takes for the routers to learn about the changes that have occurred in the topology, is in the range of 0 to 1 second. The average convergence time is equal to 2.1842 second.

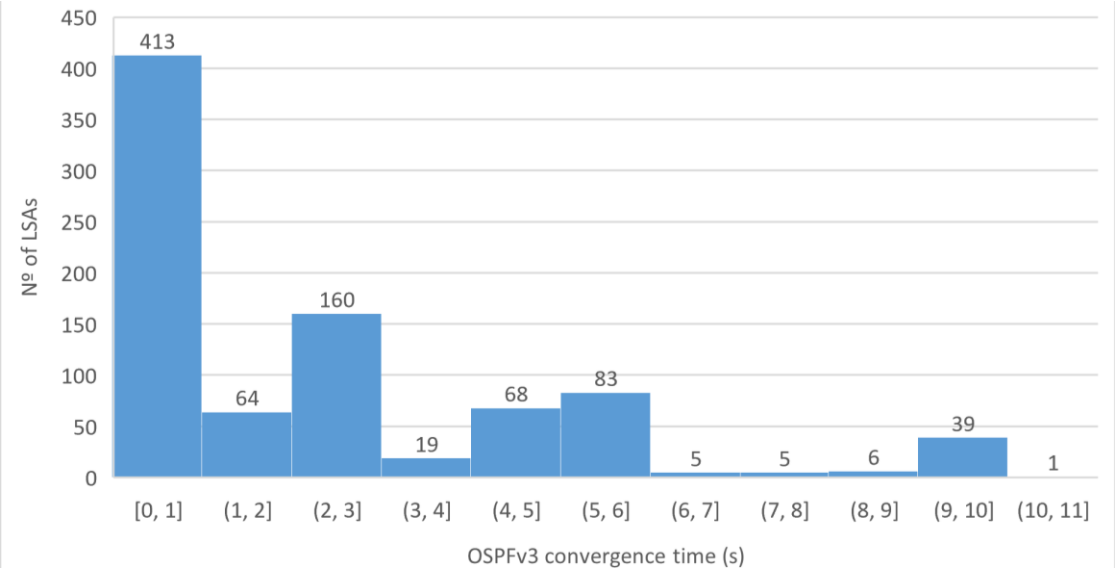

Fig. 4.13. OSPFv3 convergence time histogram. Δt = 5 s.

From `satelliteEvents.json` file and `groundStationsEvents.json` file, it has been calculated that an average of 0.25 events occur between snapshots.

Figure 4.14 shows the number of hops it takes R1 to reach the nearest, in terms of OSPFv3 cost, ground station in each snapshot. The quantity of hops to reach the base station is fairly even. Specifically, the most common is to be at a distance of 3, 4 and 5 hops.

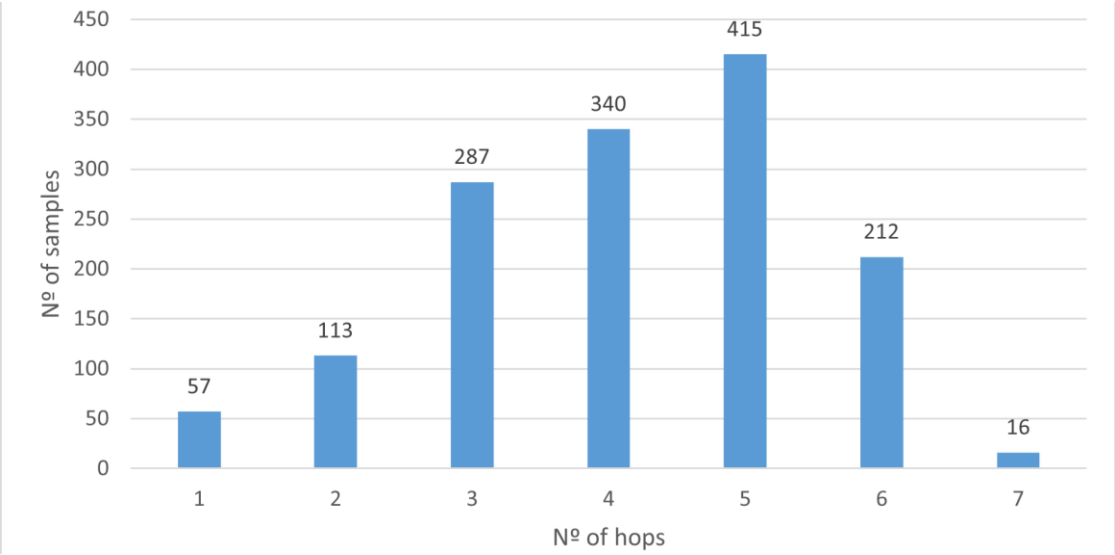

Fig. 4.14. Number of hops histogram. Δt = 5 s.

Figure 4.15 shows the satellite ground track during the two hours of emulation. As can be observed, the snapshots for Δt = 5 s and Δt = 20 s have been captured in the same time window and, therefore, the satellite travels exactly

the same trajectory, starting in northern Brazil and ending in the northern Canada. Most points are observed, especially at latitudes above 60º and below -60º due to the smaller spacing between snapshots.

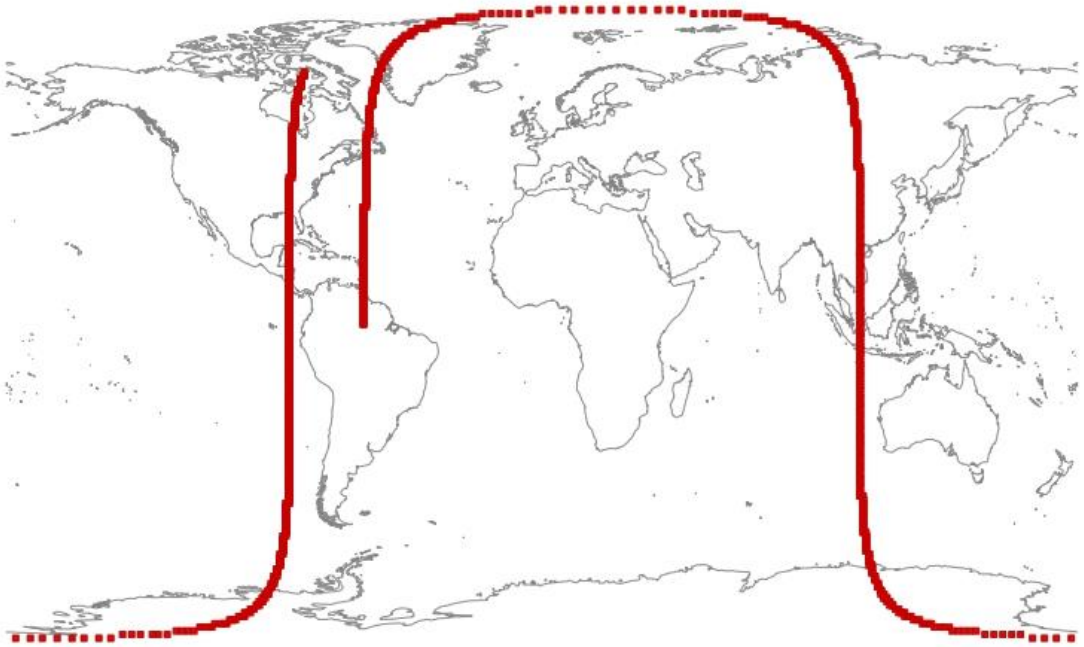

Fig. 4.15. Satellite ground track. Δt = 5 s.

4.2.2.1. Configuration 1

Table 4.3 shows a summary made by D-ITG of the metrics obtained during traffic generation.

Table 4.3. D-ITG summary. Δt = 5 s. Configuration 1.

Total time	7200 s
TRANSMISSION METRICS	
Total packets	357552 pkt
Average packet rate	49.66 pkt/s
RECEPTION METRICS	
Total packets	356996
Minimum delay	4.182 ms
Maximum delay	134.51 ms
Average delay	45.853 ms
Average jitter	325 µs
Delay standard deviation	17.083 ms
Average packet rate	49.5828 pkt/s
Packets dropped	556 (0.16 %)

Figure 4.16 shows a comparison between the evolution of the packet delay and the number of hops at each time step, throughout the entire emulation. The similarity between the shapes of the two graphs is also maintained in this scenario, showing that the greater the number of hops, the greater the packet delay. The peaks in the two graphs that last only 5 seconds are due to the fact that the time between snapshots is very short, and this can cause errors when choosing the paths to create the snapshots (gml files).

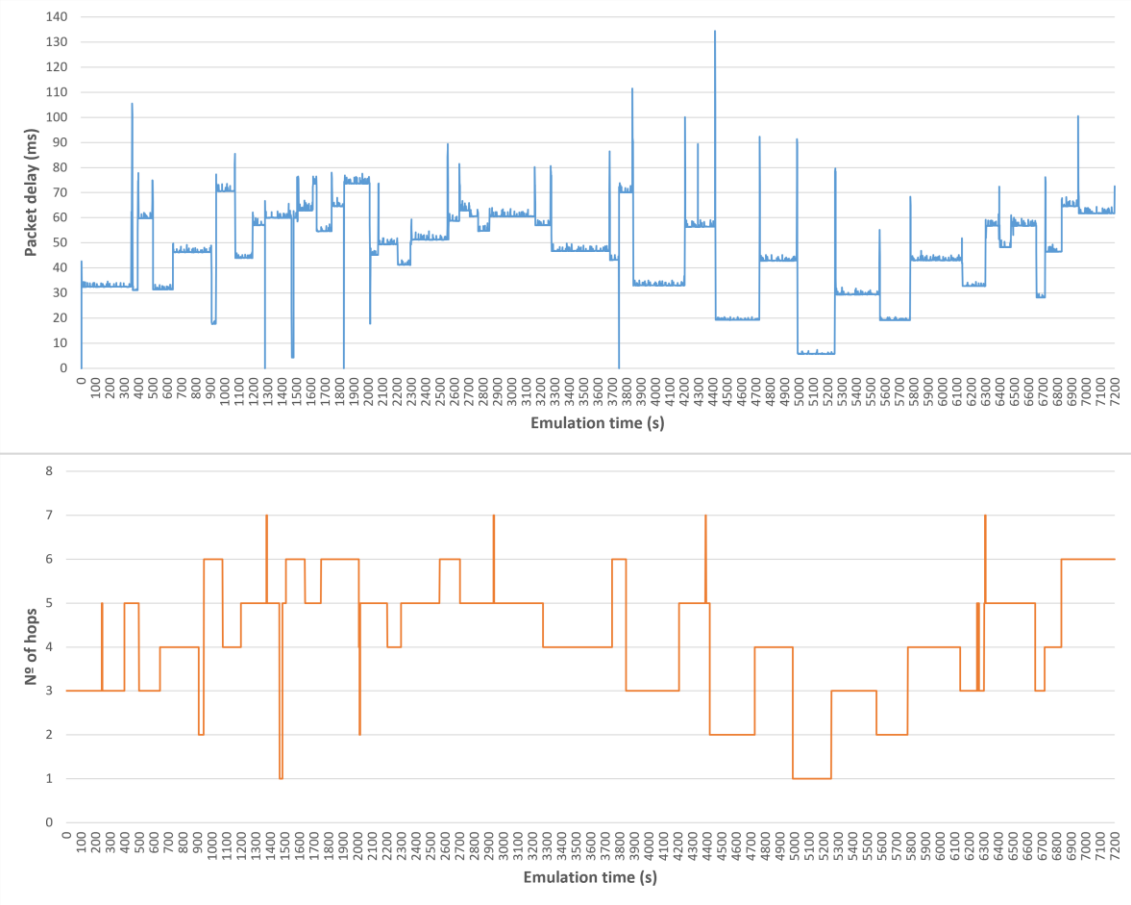

Fig. 4.16. Packet delay and number of hops. Δt = 5 s. Configuration 1.

Figure 4.17 shows the delay histogram of received packets. By having the same trajectory in the two different time intervals analyzed, the intervals where the packet delay is most concentrated remain the same. The most predominant are the (30, 35], the (45, 50] and (60, 65] ms intervals. The average delay is equal to 45.853 ms, which is almost the same as the previous scenarios.

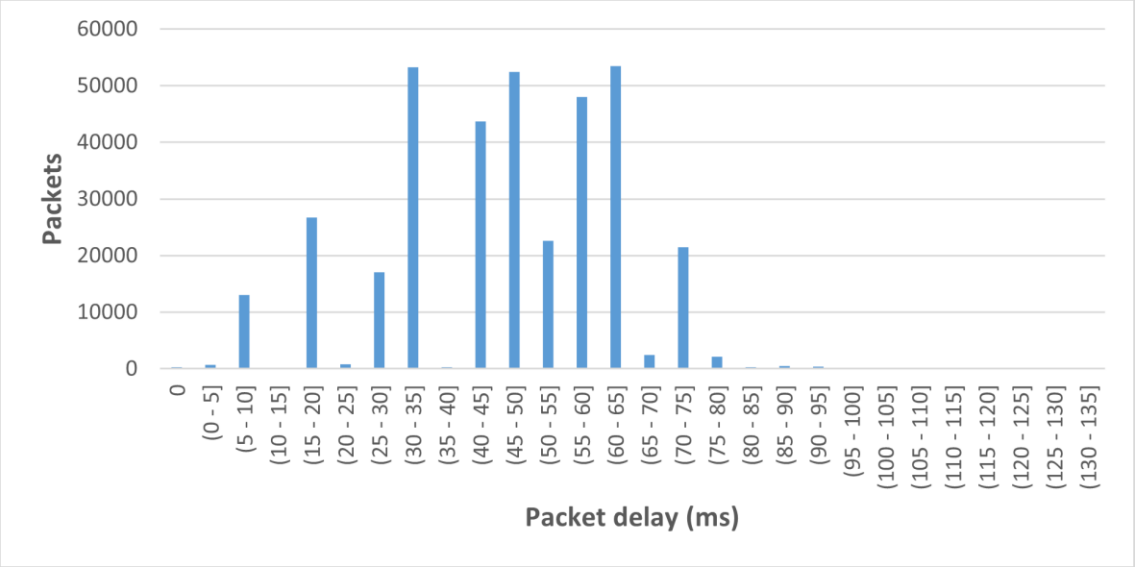

Fig. 4.17. Packet delay histogram. Δt = 5 s. Configuration 1.

Figure 4.18 shows the packet loss rate and compares it to the latitude at which the satellite is located at the same time instant, to see if there is any relationship. There are three points where 100% loss occurs and one point with more of a 90% of losses. The first takes place at 5 seconds, these losses are due to the initialization of GSL events due to dropping and recovering the link in each ground station. The loss located at 3750 seconds is due to the drop of the inter-plane ISL, as the satellite is approaching to the -60º latitude. Finally, the points located at 1285 and 1835 seconds are above of the 60º latitude.

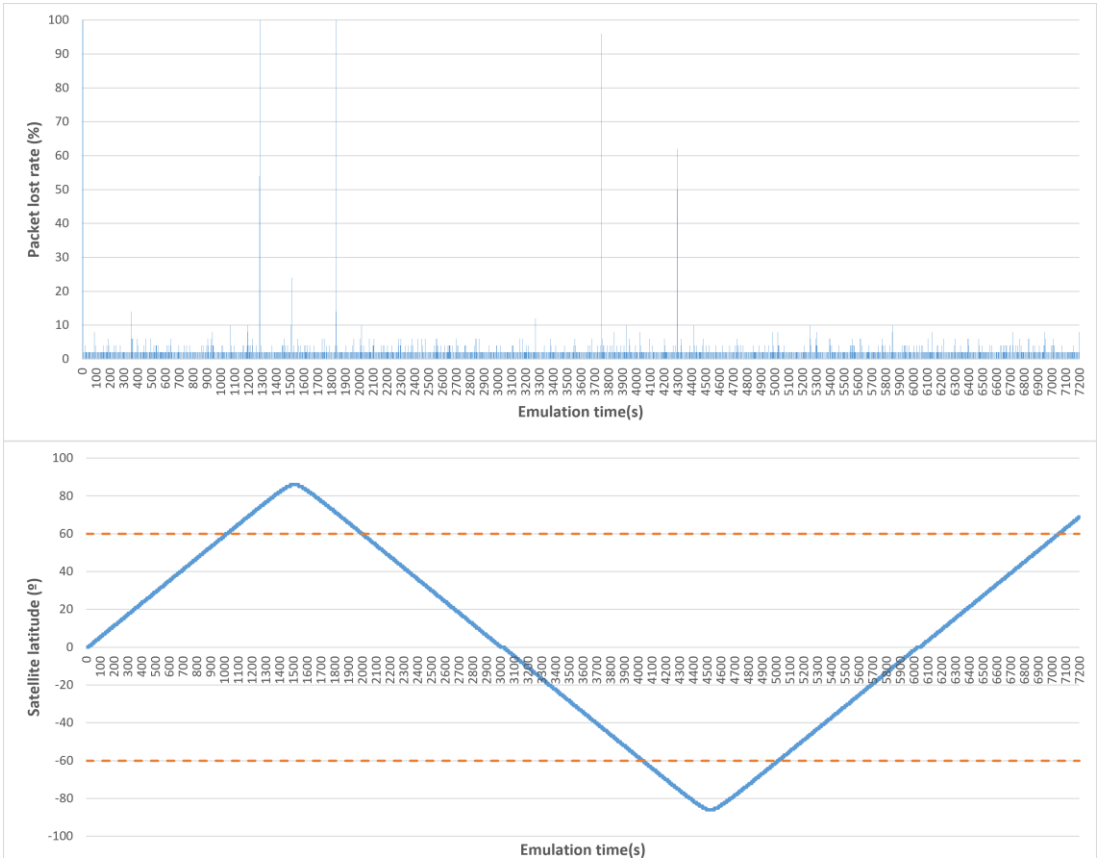

Fig. 4.18. Packet loss rate and satellite orbit latitudes. Δt = 5 s. Configuration 1.

4.2.2.2. Configuration 2

Table 4.4 shows a summary made by D-ITG of the metrics obtained during traffic generation.

Table 4.4. D-ITG summary. Δt = 5 s. Configuration 2.

Total time	7200 s
TRANSMISSION METRICS	
Total packets	22320000 pkt
Average packet rate	3100 pkt/s
RECEPTION METRICS	
Total packets	21298463
Minimum delay	4.177 ms
Maximum delay	132.427 ms
Average delay	46.387 ms
Average jitter	107 µs
Delay standard deviation	20.13 ms
Average packet rate	2958.12 pkt/s
Packets dropped	1021537 (4.58 %)

Figure 4.19 shows a comparison between the evolution of the packet delay and the number of hops at each time step. As in the previous scenarios, the two shapes of the graphs are equal.

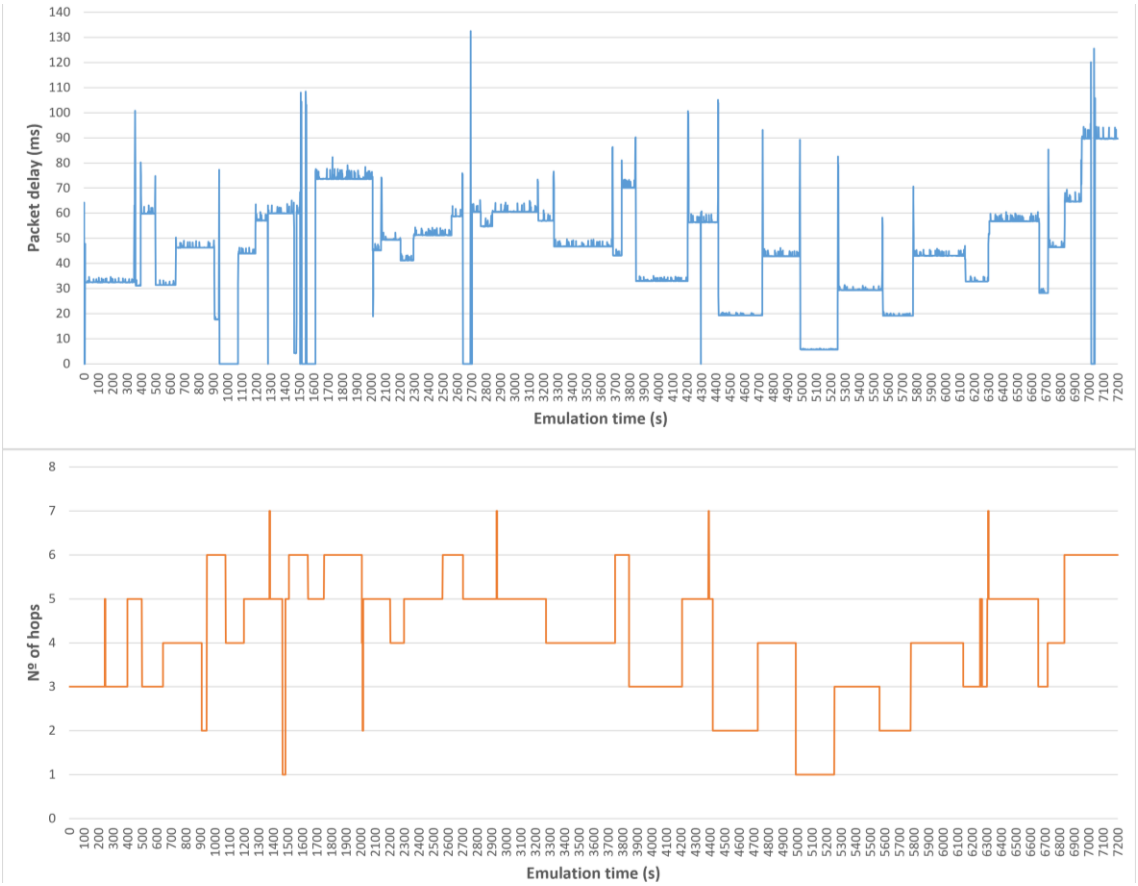

Fig. 4.19. Packet delay and number of hops. Δt = 5 s. Configuration 2.

Figure 4.20 shows the delay histogram of received packets. Since they have the same spacing between snapshots and follow the same satellite ground path, the histogram distribution for this configuration is practically the same as for configuration 1. The (30, 35] ms interval is still the predominant one, but the second one now is the (55, 60] ms interval. The average delay is equal to 46.387 ms, which is almost the same.

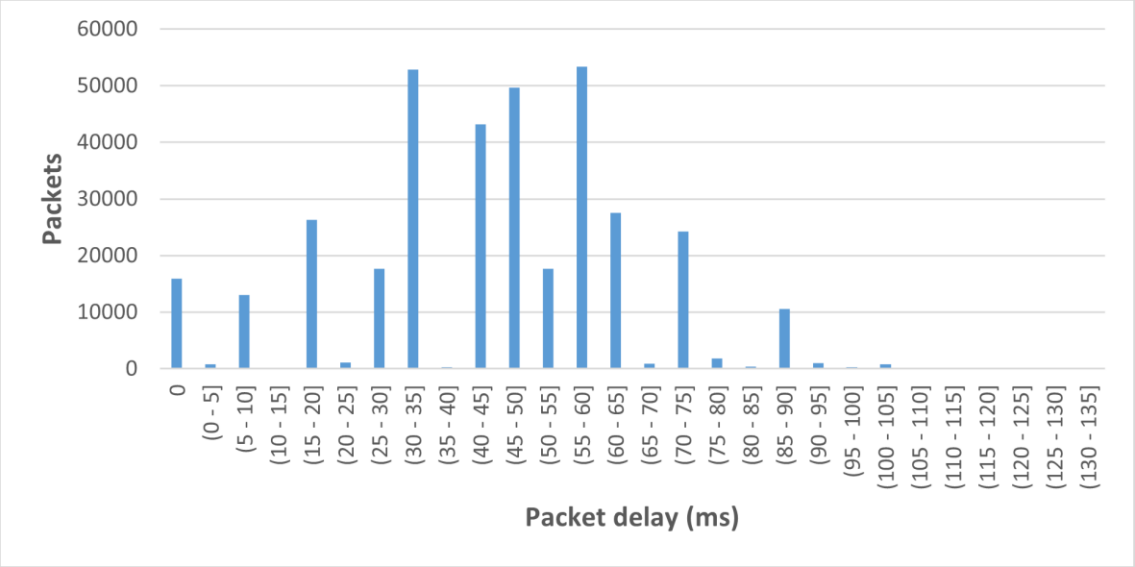

Fig. 4.20. Packet delay histogram. Δt = 5 s. Configuration 2.

Figure 4.21 shows the packet loss rate and compares it to the latitude at which the satellite is located at the same time instant. In this case, there are 6 zones or points where all the packets are lost. The first, as in the previous scenario, is due to the starting of the GSL events. The areas from 945 to 1075 seconds and from 7015 to 7040 seconds, are the points where the 60º of latitude are just crossed. The peak at 1280 seconds and the zone between 1515 and 1545 seconds are above 60º latitude where the inter-plane ISL is down. The peak at 4300 seconds is a similar case, but in this case, it is below -60º. Finally, the zone from 2640 to 2700 seconds is around the equator, which is one of the points where there is a greater number of hops between the satellite and the nearest base station. It is clear that at such a high rate the system becomes saturated at certain times.

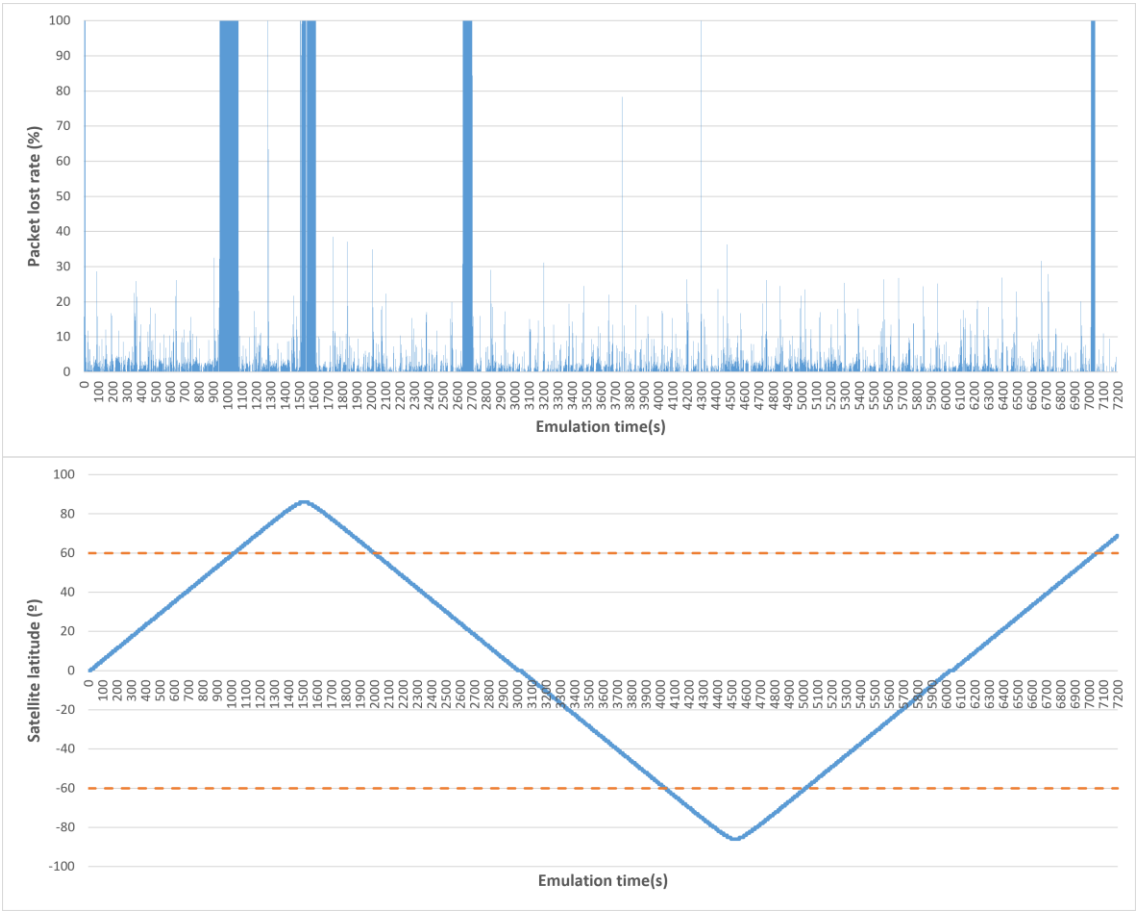

Fig. 4.21. Packet loss rate and satellite orbit latitudes. Δt = 5 s. Configuration 2.

Chapter 5. Conclusions and Future Lines

5.1. Conclusions

In this project we have detailed the design decisions of an emulation platform for testing IP routing protocols in LEO satellite networks using LXC as the virtualization technology to emulate the routing capabilities of satellites and the ISL and GSL links. Also, we have developed a set of python tools that allow us to launch the emulation and realize captures to analyze the routing protocols. In particular, OSPFv3 that is the studied routing protocol.

Once the emulation platform was implemented, we analyzed the general performance of OSPFv3 protocol. Results of convergence time, hop count, delay, jitter delay and loss rate. The study has been done in a full-lap Iridium constellation taking topology snapshots in steps of 20 and 5 seconds and two data rates, one a low data rate typical of VoIP applications, and the other a high data rate in order to stress the system.

The convergence time of the OSPFv3 protocol, which is the time it takes for all satellite to learn about topology changes, is about 2~3 seconds on average. In terms of average packet delay, the values oscillate between 15 ms and 75 ms, centered at 45~46 ms. As for the delay variation, i.e., jitter, it varies by a maximum of 325 µs. The most common number of hops is 3, 4 or 5 hops. This last statement is not valid for all cases, since it depends on the time window in which the topology captures are taken.

Focusing on packet losses, the most important points to take into account, regardless of the initial events of the ground stations which is a matter of programming, are:

- Exceed ± 60º latitudes.
- Moment in which the inter-plane ISL goes down.
- Zone near the equator, where the satellites are farther away from ground stations in number of hops, but at mention above, the number of hops distribution depends on the time window when the captures were taken.
- Switching from being connected to one ground station to another one.

Although there are several critical points to monitor, losses in general have been very low, always below 5% losses on average in the four scenarios studied.

Although final loss rate is acceptable, it is true that in short periods of time the loss rate is high, which means that we can lose communication for periods of 5-

10 seconds, approximately. The highest loss rates occur in high or low (depending of the hemisphere) latitudes.

5.2. Future lines

One of the possibilities it is to study other routing protocols as IS-IS or RIP and compare them. Independently of the benefits in the selection of one of them as the result of the study all these routing protocols find the optimal route in terms of its own cost functions. As satellite networks links are all equal, all will find the shortest hop count route.

It is not always necessary to find the optimal route. For example, if we define a maximum delay of T time units, any route that satisfies this restriction will be valid, and then, other non-optimal algorithms should be evaluated.

Segment Routing over IPv6 (SRv6) appears as an elegant and modern solution that we consider that can be applied in these networks. SRv6 is a source routing solution, that is, the source packet who includes the route to destinations. There is a data plane, where an ingress router sets the complete source-destination route, and a control plane where one entity, usually called Controller, instructs the router which route apply to a particular flow of packets.

The idea is that the controller, which has the topology snapshots and the airplane route can compute a valid route (in terms of accomplish defined restrictions) applying routing algorithms, not necessarily optimal. Obviously, we must also define a protocol which allows to the Controller install routes in the satellites.

BIBLIOGRAPHY

[1] FUTUR UPC. *Seamless integration of LEO Satellite, Aeronautical and Terrestrial Networks* [online]. September, 2022. [Accessed: May, 2023]. Available at: https://futur.upc.edu/34233458

[2] EUROCONTROL. ISA project – Satellite perspectives for CNS/ATM. December, 1997. EEC Note No. 29/97, EEC Task D14

[3] Han, L.; Retana, A.; Westphal, C.; Li, R. Large Scale LEO satellite Networks for the Future Internet: Challenges and Solutions to Addressing and Routing. *Computer Networks and Communications* [online], 2022, vol.1, no. 1, p. 31-63. [Accessed: May, 2023]. Available at: https://ojs.wiserpub.com/index.php/CNC/article/view/2105

[4] Xiaogang, Q.; Jiulong, M.; Dan, W.; Lifang, L.; Shaolin, H. A survey of routing techniques for satellite networks. *Journal of Communications and Information Networks*. 2016, vol. 1, no. 4, p. 66-85. [Accessed: May, 2023]. Available at: https://doi.org/10.11959/j.issn.2096-1081.2016.058

[5] Hussein, M.; Hanani, A. Routing in IP/LEO satellite communication systems: past, present and future. *International Journal of Electronics and Communication Engineering*, 2016, vol. 3, p. 745. [Accessed: May, 2023]. Available at: http://hdl.handle.net/20.500.11889/4353

[6] Chen, Q.; Guo, J.; Yang, L.; Liu, X.; Chen, X. Topology virtualization and dynamics shielding method for LEO satellite networks. *IEE Communication Letters*, 2020, vol. 24, no. 2, p. 433-437. [Accessed: May, 2023]. Available at: https://doi.org/10.1109/LCOMM.2019.2958132

[7] Wood, L.; Clerget, A.; Andrikopoulos, I.; Pavlou, G.; Dabbous, W. IP routing issues in satellite constellation networks. *International Journal of Satellite Communications*. 2001, vol. 19, no. 1, p. 69-92. [Accessed: May, 2023]. Available at: https://doi.org/10.1002/sat.655

[8] Duan, C.; Feng, J.; Chang, H.; Song, B.; Xu, Z. A Novel Handover Control Strategy Combined with Multi-hop Routing in LEO Satellite Networks. *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW).* 2018, p. 845-851. [Accessed: May, 2023]. Available at: https://doi.org/10.1109/IPDPSW.2018.00132

[9] Fossa, C. E.; Raines, R. A.; Gunsch, G. H.; Temple, M. A. An overview of the Iridium® low Earth orbit (LEO) satellite system. *Proceeding of the IEEE 1998 National Aerospace and Electronics Conference. NAECON 1998. Celebrating 50 Years (Cat. No.98CH36185)*. 1998, p. 152-159. [Accessed: May, 2023]. Available at: https://doi.org/10.1109/NAECON.1998.710110

[10] Lemme, P. W.; Glenister, S. M.; Miller, A. W. Iridium® aeronautical satellite communications. *IEEE Aerospace and Electronic Systems Magazine*. 1999, vol. 14, no. 11, p. 11-16. [Accessed: May, 2023]. Available at: https://doi.org/10.1109/62.809197

[11] Champlin, C. Iridium® satellite: A large system application of design for testability. *Proceedings of IEEE International Test Conference – (ITC)*. 1993, p. 392-398. [Accessed: May, 2023]. Available at: https://doi.org/10.1109/TEST.1993.470673

[12] Anderson, B.J.; Angappan, R.; Barik, A.; Vines, S. K.; Stanley, S.; Bernasconi, P. N.; Korth, H.; Barnes, R. J. Iridium® communications satellite constellation data for study of Earth's magnetic field. *Geochemistry, Geophysics, Geosystems*, 2021, vol. 22. [Accessed: May, 2023]. Available at: https://doi.org/10.1029/2020GC009515

[13] Bo, W.; Lixiang, L.; Shuaijun, L.; Shan, W.; Hailong, H. Performance of analysis of OSPF in integrated satellite and terrestrial network. *2022 International Symposium on Networks, Computers and Communications (ISNCC)*, 2022, p. 1-4. [Accessed: June, 2023]. Available at: https://doi.org/10.1109/ISNCC55209.2022.9851783

[14] Pan, T.; Huang, T.; Li, X.; Chen, Y.; Xue, W.; Liu, Y. OPSPF: Orbit Prediction Shortest Path First Routing for Resilient LEO Satellite Networks. *ICC 2019 – 2019 IEEE International Conference on Communications*, 2019, p. 1-6. [Accessed: June, 2023]. Available at: https://doi.org/10.1109/ICC.2019.8761611

[15] Fu, M.; Guo, B.; Yang, H.; Pang, C.; Huang, S. Routing Optimization Based on OSPF in Multi-Layer Satellite Network. *Proceedings of CECNet 2021*, 2021. [Accessed: June, 2023]. Available at: https://doi.org/10.3233/FAIA210452

[16] He, Y.; Chen, H.; Ma, C. A Cross-Domain Aggregation Routing based on Lightweight OSPF Protocol for the GEO Satellite-Ground Integrated Network. *2022 2nd International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI)*, 2022, p. 459-463. [Accessed: June, 2023]. Available at: https://doi.org/10.1109/CEI57409.2022.9948464

[17] Cao, S.; Zhang, T. Congestion Control Based on OSPF in LEO Satellite Constellation. *2019 IEEE 19th International Conference on Communication Technology (ICCT)*, 2019, p. 1111-1115. [Accessed: June, 2023]. Available at: https://doi.org/10.1109/ICCT46805.2019.8947269

[18] Yang, H.; Guo, B.; Xue, X.; Deng, X.; Zhao, Y.; Cui, X.; Pang, C.; Ren, H.; Huang, S. Interruption Tolerance Strategy for LEO Constellation with Optical Inter-satellite Link. IEEE Transactions on Network and Service Management, 2023. [Accessed: June, 2023]. Available at: https://doi.org/10.1109/TNSM.2023.3274638

[19] Canonical Ltd. *Container and virtualization tools* [online]. [Accessed: May, 2023]. Available at: https://linuxcontainers.org/

[20] What's a Linux Container? *Red Hat* [online]. May 11, 2022. [Accessed: May, 2023]. Available at: https://www.redhat.com/en/topics/containers/whats-a-linux-container

[21] Alpine Linux. *Alpine Linux* [online]. [Accessed: May, 2023]. Available at: https://www.alpinelinux.org/

[22] Kassing, S.; Bhattacherjee, D.; Águas, A. B.; Saethre, J. E.; Singla, A. Exploring the "Internet from space" with Hypatia. *IMC '20: Proceedings of the ACM Internet Measurement Conference*, 2020, p. 214-229. [Accessed: May, 2023]. Available at: https://doi.org/10.1145/3419394.3423635

[23] Kassing, S.; Bhattacherjee, D.; Águas, A. B.; Saethre, J. E.; Singla, A. *Hypatia source code*. Available at: https://github.com/snkas/hypatia

[24] Canonical Ltd. Pylxd documentation. *Read the Docs* [online]. [Accessed: May, 2023]. Available at: https://pylxd.readthedocs.io/en/latest/

[25] Canonical Ltd. Pylxd source code. Available at: https://github.com/lxc/pylxd

[26] Grönholm, A. Advanced Python Scheduler. *Read the Docs* [online]. [Accessed: May, 2023]. Available at: https://apscheduler.readthedocs.io/en/3.x/

[27] Grönholm, A. APScheduler source code. Available at: https://github.com/agronholm/apscheduler/tree/3.x

[28] Hagberg, A. A.; Schult, D. A.; Swart, P. J. Exploring Network Structure, Dynamics, and Function using NetworkX. *Proceedings of the 7th Python in Science conference*. 2008, p. 11-15. [Accessed: May, 2023]. Available at: https://conference.scipy.org/proceedings/SciPy2008/paper_2/

[29] Hagberg, A. A.; Schult, D. A.; Swart, P. J. *NetworkX source code*. Available at: https://github.com/networkx/networkx

[30] Newt, K. *PyShark documentation*. [Accessed: May, 2023]. Available at: http://kiminewt.github.io/pyshark/

[31] Newt, K. PyShark source code. Available at: https://github.com/KimiNewt/pyshark/

[32] Tilmans, O.; Jadin, M. *IPMininet*. [Accessed: February, 2023]. Available at: https://github.com/cnp3/ipmininet

[33] Mininet Project Contributors. *Mininet*. [Accessed: February, 2023]. Available at: http://mininet.org/

[34] Coltun, R.; Ferguson, D.; Moy, J.; Lindem, A. RFC 5340 - "OSPF for IPv6". *RFC editor*, 2008. [Accessed: June, 2023]. Available at: https://doi.org/10.17487/RFC5340

[35] Machado, S.; Agustí, A; León, O.; Raspall, F. *Encaminament IP interior – Protocol OSPF*. Notes of the subject Arquitectura i Protocols d'Internet, taught at Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels of the UPC. [Accessed: June, 2023]

[36] Agustí, A; León, O.; Raspall, F. *Introducció al protocol IPv6*. Notes of the subject Arquitectura i Protocols d'Internet, taught at Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels of the UPC. [Accessed: June, 2023]

[37] Botta, A.; De Donato, W.; Dainotti, A.; Avallone, S.; Pescapé, A. *D-ITG 2.8.1 Manual*, 2013. Accessed: July, 2023]. Available at: https://traffic.comics.unina.it/software/ITG/manual/index.html

LIST OF ACRONYMS AND ABBREVATIONS

CDAR-L	Cross-Domain Aggregation Routing based on Lightweight OSPF
CSPF	Constrained Shortest Path First
DHCP	Dynamic Host Configuration Protocol
D-ITG	Distributed Internet Traffic Generator
FRR	FRRouting Protocol Suite
GEO	Geostationary Earth Orbit
GSL	Ground to Satellite Link
IGP	Interior Gateway Protocol
ISL	Inter-Satellite Link
ITSN	Integrated Terrestrial-Satellite Network
L4S	Low Latency Low Loss Scalable Throughput Internet Service
LEO	Low Earth Orbit
LS	Link State
LSA	Link State Advertisement
LXC	Linux Container
LXD	Linux Container Daemon
NAT	Network Address Translation
OOWLP	Optimized OSPF with Link Plan
OPSPF	Orbit Prediction Shortest Path First
OSPF-PUR	OSPF-based Predictive Update Routing
OSPFv3	Open Shortest Path First version 3
PSTN	Public Switched Telephone Network
RIB	Routing Information Base
SRv6	Segment Routing over IPv6
UDP	User Datagram Protocol
ULA	Unique Local Address

LIST OF FIGURES

LIST OF TABLES

APPENDICES

TFG TITLE: ANALYSIS OF OSPFv3 IN LEO SATELLITE NETWORKS

DEGREE: Double bachelor's degree in Aerospace Systems Engineering and Telecommunications Systems

AUTHOR: Daniel Román Martín

DIRECTOR: Sergio Machado Sánchez

CODIRECTOR: Jorge Mata Díaz

DATE: July 24th, 2023

APPENDIX A. EXAMPLE OF EMULATION RUNNING

The steps required to start the emulation, as well as to start the tests, are described in the below. All steps are described from the `/seamsat` directory.

1. The first step is to place the `satellite_network_i.gml` and `satellite_in_range_i.txt` files in the folders named `gmlSatellites` and `txtSatInRange`, respectively.

2. Then you would have to set the time between snapshots in the `convertToJSON.py` and `createEvents.py` files, in the variable called `intervalTime`, at the beginning of the code.

3. You would have to do the same as in the second step, but in the `topology.py` file, in the variable `prevTime` that is located inside the `__addEvents()` function.

4. In the `cli.py` file, at the end in the `numSnapshots` variable, you set the number of snapshots to complete the path, i.e., if you want to cover two hours with a snapshot interval of 20 seconds, the value to set would be 360.

5. The code is executed with the command `sudo python cli.py`. Once the terminal is loaded, the only thing left to do is to execute the `start` command.

To start the pcapng captures, wait until all the routers have been created and started. When the message ‘Routers started’ appears on the screen, manually run the `capture.sh` script. In this script the only thing you have to modify is the name you want to save the capture.

The code for counting the number of hops is found at the end of the `__addEvents()` function. The only thing to modify is the time between snapshots.

Finally, to start the D-ITG you would have to pen two terminals and attach on to the generator and the other to the sink. Then, you would have to put the commands to capture and execute them before the first events occurs.