Advanced Constellation Emulation and Synthetic Datasets Generation for Non-Terrestrial Networks

Camilo Rojas, Juan A Fraire, Fabio Patrone, Mario Marchese

Advanced Constellation Emulation and Synthetic Datasets Generation for Non-Terrestrial Networks

Camilo Rojas*, Juan A. Fraire†‡, Fabio Patrone*, Mario Marchese*
*University of Genoa, Genoa, Italy
†Inria, INSA Lyon, CITI, UR3720, 69621 Villeurbanne, France
‡CONICET - Universidad Nacional de Córdoba, Córdoba, Argentina

***Abstract*—Mega satellite constellations, now realized entities, encompass thousands of nodes. However, efficient orchestration of multi-hop paths and distributed processing tasks in Non-Terrestrial Networks (NTN) remains a considerable challenge. The integration of NTN systems into 5G cellular networks necessitates innovative adaptations of Software-Defined Networking (SDN) and Multi-access Edge Computing (MEC) to suit the dynamic environments of NTN. In this context, we present *MeteorNet*, a state-of-the-art emulation tool conceived for satellite constellations. MeteorNet accurately replicates the behavior of NTNs by implementing space orbits, Earth rotation calculations, and Linux network interfaces across diverse network layers. Coupled with a continuous measurement system founded on *sFlow*, MeteorNet compiles critical switch variables in a centralized database, thus providing a distinctive methodology for creating realistic synthetic datasets. The pertinence of synthetic datasets is paramount in NTN, given the scarcity of operative systems and the inaccessibility of accurate data from the few existing systems due to proprietary constraints. These datasets are instrumental for formulating and training intelligent control algorithms and Machine Learning (ML) models for SDN and MEC advancements in NTN. To illustrate the efficacy of this approach, we explore a realistic networking case study with a *ring* topology, demonstrating how data models describe intricate routing and edge computing protocols for NTN.**

***Index Terms*—Satellite Constellation, Software-Defined Networks, Multi-access Edge Computing, Synthetic Datasets, Machine Learning.**

I. Introduction

The emergence of Low Earth Orbit (LEO) satellite constellations marks a significant breakthrough in space technology. The industry has achieved democratized access by utilizing Commercial Off-The-Shelf (COTS) components, implementing agile methodologies, and employing standardized launch infrastructures [1].

Private entities in the space sector have spearheaded the development of mega-constellations in Low Earth Orbit (LEO). A recent driving force in the industry has been the integration of aerial and space segments with terrestrial networks, giving rise to the concept of Non-Terrestrial Networks (NTN). This concept has been incorporated into 5G standards since Release 16 [2].

Nevertheless, for the seamless integration of NTNs into 5G networks, the network must comply with Key Performance Indicators (KPIs) defined by the 3GPP. These mainly concern capacity, latency, and service reliability. Software-defined networking (SDN) is considered an enabling technology to meet these KPIs in the context of space. However, adapting SDN to this emerging context poses substantial challenges.

While an efficient SDN layer is imperative, a high degree of automation is essential to harness the service potential of mega satellite constellations fully. Multi-access Edge Computing (MEC) is a promising solution to address contemporary computational challenges. MEC has the potential to facilitate the efficient allocation and distribution of computational and memory resources, encompassing both terrestrial and space-based assets throughout a satellite constellation. Such a framework could deliver real-time services to terrestrial users, including Machine Learning (ML) applications that empower autonomous land, air, and maritime vehicles globally. However, implementing MEC within satellite constellations is a non-trivial task that demands the development of intelligent algorithms to manage these resources effectively [3].

This paper examines the integration of SDN and MEC into satellite constellation typologies. We introduce *MeteorNet*: a self-made open-source framework designed for emulating MEC in non-terrestrial software-defined networks [4]. MeteorNet offers a dockerized environment based on Mininet [5] for deploying and evaluating flight and ground software among its diverse features. The emulation environment faithfully replicates communication conditions during operational scenarios, uses an SDN controller, and leverages native kernel and network management code. The constellation emulation framework includes an *sFlow*-based data collection and storage system that logs state variables and historical data sets into a centralized database. The obtained data sets can be used for the analysis of orbital scenarios and to train control and ML models to support the orchestration of future SDN and MEC solutions in NTN. MeteorNet can feed machine learning (ML) models with realistic datasets while comparing their performance to baseline SDN and MEC algorithms.

There are various potential ML management strategies in space missions, such as resource allocation and task offloading distribution [6]. Creating synthetic structured datasets derived from actual constellation scenarios offers several advantages for designing and deploying learning controllers. In the space sector, many tools and datasets are bound by proprietary constraints and closely guarded by satellite operators. A synthetic dataset generator like MeteorNet accelerates progress and facilitates advancements in this field.

The remainder of the paper is organized as follows. Sec-

tion II overviews fundamental concepts and related research. Section III explores the MeteorNet. Section IV presents a detailed case study. Finally, Section V summarizes the conclusions drawn from this study and suggests future research directions.

II. Background

A. Software Defined Networks

SDN is an innovative concept adopted by the 5G standard to facilitate the organization and deployment of flexible and scalable applications. SDN attempts to separate the network control and data planes to improve user experience, optimizing resources depending on the application-specific requirements. The original design of SDN protocols assumes a central controller that, on the one hand, gathers information from a direct connection with network switches (data plane) and, on the other hand, takes decisions involving bandwidth assignment and path prioritization from application needs. The aforementioned centralized approach of SDN controllers challenges the integration of this technology with the dynamic nature of NTN.

Efforts have been undertaken to customize SDN management specifically for satellite networks. Miao et al. [7] and Jiang et al. [8] have examined various research initiatives in this domain. They view SDN and Network Function Virtualization (NFV) as technologies employable to facilitate the integration of NTN-5G, proposing the deployment of multiple SDN controllers interconnected through tightly bound interfaces, particularly in the context of intricate and expansive networks. In this arrangement, each controller assumes responsibility for overseeing a distinct sub-network.

B. Multi-access Edge Computing

MEC presents a distributed computing paradigm to bring storage and processing capabilities close to end-users. The primary objective is to reduce computational latency and optimize network bandwidth utilization.

In their work, Wang et al. [9] characterize MEC as an innovative approach for enabling IoT applications in satellite networks. Their research proves that space-edge computing outperforms traditional satellite constellations regarding time and energy efficiency. Actual implementations of MEC involve deploying numerous servers across network nodes, each tasked with providing virtual functions tailored to particular services with low computation latency requirements. Zhang et al. [10] introduce MEC to enhance Quality of Service (QoS) and accelerate performance within NTNs. They propose task scheduling models aimed at collaborative computation on MEC servers. Pfandzelter et al. [11] delves into the distinctive attributes of LEO MEC and assesses the suitability of three constellations (SpaceX, Amazon, and Telesat) for implementing distributed computing.

The literature encompasses a variety of offloading strategies. For example, Sonmez et al. [12] propose a fuzzy orchestration approach for MEC offloading, which emulates the decision-making process of human administrators, resulting in an automated management system. Cassara et al. [13] conducts a comparative analysis of task-offloading strategies across different constellation scenarios, ultimately concluding that fuzzy strategies deliver superior outcomes when contrasted with round-robin and full-offloading solutions.

C. Emulation Tools for SDN and MEC in NTNs

Effective resource planning and real-time monitoring are pivotal in the context of satellite missions. Mission control teams must constantly adapt scheduling plans in response to mission modifications or natural events [14] Emulation and simulation tools play a crucial role during mission planning and are subsequently enriched with actual mission data to enhance logistical and operational planning. These tools are indispensable for the rigorous testing and deployment of software and algorithms in the context of space missions.

Various research teams and space agencies are currently employing several emulation tools during all the steps of mission plan development. The most popular tools for network simulation are NS3 [15] and Omnet++ [16]. Both frameworks follow a discrete-time simulation paradigm and support various fronted APIs in well-known programming languages like C++ and Python. Engineers and researchers must develop components interfacing with these tools to support constellations and satellite links.

Present discrete-time network simulation frameworks necessitate substantial adjustments to mission software and system architecture due to their abstract representation of computer networks. This can introduce coding errors in the transition of development and deployment phases. In addition, these tools do not emulate network interfaces and operating system kernels, creating a disconnect between simulation and computing systems.

In this article, we propose the creation of a novel emulation software addressing these limitations and providing a more faithful representation of actual system deployment. The final goal is to be able to generate realistic datasets to support ML approaches to MEC and SDN in future NTN systems.

D. Synthetic Dataset Collection

The data collected from NTN emulators can create realistic synthetic datasets representing network conditions, behavior, and traffic patterns. Among monitoring technologies, *sFlow* allows for granular data sampling, providing a more detailed and accurate representation of network traffic, essential for creating realistic synthetic datasets [17].

sFlow stands for *sampled flow*. It is designed to be a scalable and efficient way to measure network traffic, helping network administrators to manage and optimize network performance and bandwidth usage efficiently. sFlow operates by randomly sampling packets that traverse through a network device like a switch or a router and sending the sampled packets as sFlow datagrams to a central collector. The collector then comprehensively analyzes the sampled data to view network usage, traffic patterns, and performance. Because it uses sampling, sFlow can scale to handle high-volume traffic flows across complex, high-speed networks without consuming excessive

processing power or bandwidth. sFlow Agent resides on a network device and is responsible for sampling the packets and sending the sampled data to the sFlow collector. The sFlow Collector receives and analyzes the sampled data sent by sFlow agents. It can give network administrators detailed insights into network activity, traffic patterns, and potential issues.

As a result, sFlow is a perfect candidate to collect metrics in NTN emulators to create synthetic datasets.

III. METEORNET

Our Constellation Emulation tool called *MeteorNet*, is a continuous-time emulation platform for satellite constellation networks. MeteorNet can address the aforementioned challenges. It offers advantages over existing tools by incorporating network and operating system virtualization, including native source code from kernel and network systems. While discrete-time tools excel in repeatability and time acceleration, continuous-time frameworks provide unparalleled fidelity to real-world scenarios and facilitate scalable, multi-thread-driven development.

Built on open-source code and publicly accessible algorithms, it supports all phases of constellation mission planning, from research and development to operation. The main software structure is primarily written in Python but leverages Docker virtualization and kernel network libraries to maintain flexibility in user software development.

The proposed tool is constructed with a set of integral components: (*i*) An Orbit Propagation module, which is responsible for the management and simulation of satellite motion within its designated orbit; (*ii*) A Network Stack and Link Management system tasked with handling network protocols and communication links. (*iii*) An SDN Controller, functioning as the central hub for the supervision and coordination of network operations; (*iv*) Containerized Software, which harnesses Docker for operating system virtualization, thereby facilitating software isolation; (*v*) A Network Performance Monitoring and Storage System designed to gather network performance metrics defined by configurable *sFlow* commands [18].

a) SDN Aspect: We include an example of an SDN controller based in the Ryu library. Ryu implements the Spanning Tree Protocol (STP) as a default mechanism to manage network routing and prevent switch loops [19]. The SDN controller operates parallel with the emulation, permitting alterations to assess different routing algorithms or optimize service-specific KPIs. The framework leverages Mininet for network virtualization.

b) MEC Aspect: We prioritize the practical evaluation of MEC performance by ensuring that satellite and ground nodes operate within discrete environments from the perspective of system resources. Furthermore, it acknowledges and caters to the unique compilation and operational requirements of software deployed in satellite and ground architectures, emulating these based on mission stipulations. Utilizing Docker OS virtualization [20], the framework establishes isolated containers, enabling the software under scrutiny to function autonomously from the emulation code. This strategy enhances the authenticity of the emulations and minimizes the necessary alterations to implement mission software.

c) Workflow: The proposed framework requires, as input, the satellite Two-Line Element (TLE), the initial coordinates of ground nodes, and the overall emulation duration necessary for network instantiation. Once initiated, the framework executes the following operations: (*A*) it propagates the orbits of satellite nodes; (*B*) computes satellite-terrestrial contact tables; (*C*) routinely modifies network link parameters and status based on the data within the computed contact tables; (*D*) executes dockerized containers; and (*E*) concurrently measures and records output performance regarding network usage and task computation delay. In the succeeding subsections, we delve into the specifics of these operational steps.

A. Orbit Propagation

We utilize the Simplified General Perturbations Model 4 (SGP4) [21] to propagate the orbits of satellite nodes. SGP4, a standard defined by NORAD and NASA, uses a streamlined perturbation model to determine orbital state vectors based on initial conditions provided in a TLE object. The model yields each satellite's coordinates, $X_s(t) \leftarrow (x_s(t), y_s(t), z_s(t))$, at a given time t, which are expressed in the True Equator Mean Equinox (TEME) coordinate system. For ground nodes, we convert the initial latitude, longitude, and altitude values into TEME coordinates, $X_g(0) \leftarrow (x_g(0), y_g(0), z_g(0))$, and subsequently use Earth's rotational angular velocity and average radius to ascertain the node's coordinates, $X_g(t)$, at time t.

Using the coordinates $X_s(t)$ and $X_g(t)$ at each time step t throughout the simulation, we can determine the elevation angle $\theta_{gs}(t)$ and distance $d_{gs}(t)$ between each ground node g and satellite s. Subsequently, we can estimate the line of sight $LoS_{gs}(t)$ using the elevation angle. We define a minimum elevation angle $\theta_C = 30^\circ$ to consider the satellite within the line of sight, making the link accessible. Links with angles less than θ_c will be deemed as having no connection.

The communication channel's availability, delay $T_{gs}(t)$, and capacity $C(t)$ is computed using Equations (1), (2), and (3).

$$LoS(t) = \begin{cases} true, & if \quad \theta_{gs}(t) \geq \theta_c \\ false, & if \quad \theta_{gs}(t) < \theta_c \end{cases} \tag{1}$$

$$T_{gs}(t) = d_{gs}(t)/c + \delta_0 \tag{2}$$

$$C(t) = \begin{cases} 200 \text{ Mb/s}, & 0 < d(t) \leq 500 \text{ km} \\ 80 \text{ Mb/s}, & 500 < d(t) \leq 1000 \text{ km} \\ 60 \text{ Mb/s}, & 1000 < d(t) \leq 2000 \text{ km} \\ 20 \text{ Mb/s}, & 2000 < d(t) \leq 3000 \text{ km} \\ 10 \text{ Mb/s}, & 3000 < d(t) \leq 4000 \text{ km} \\ \varnothing & d(t) > 4000 \text{ km} \end{cases} \tag{3}$$

where $c = 300,000 km/s$ is the speed of light in vacuum and δ_0 is an additional delay per link to account for switch computation time.

Table I
EXAMPLE OF A GROUND NODE'S (*gn*) CONTACT TABLE

time	st10	st9	st8
0	1672	3405	0
100	2188	2808	0
200	2769	2246	0
300	3376	1759	0
400	0	1431	0
500	0	1385	0
600	0	1644	3660
700	0	2096	3072
800	0	2639	2516
900	0	3225	2023
1000	0	0	1660

B. Contact Table Computation and Utilization

MeteorNet employs Mininet [5] to emulate a realistic virtual network, incorporating a genuine Linux network kernel, switch, and application code [22]. Mininet allows for the dynamic deployment and modification of custom virtual networks, thereby closely mirroring real-world communication networks by utilizing technologies and code from production environments. Ground stations can be configured for single or multiple connections with satellite nodes. In the *single-link* mode, the ground node establishes and maintains the connection with the nearest available satellite until it exits the line of sight (LoS). In the *multiple-link mode*, multi-path communications are created and preserved with all nodes within LoS. Link states for all communication nodes throughout the simulation are encapsulated in structures known as *Contact Tables*. Each node in the network has a corresponding contact table, which outlines the connections between that node and all destination nodes. Each row within a contact table corresponds to a time step, enabling or disabling links and adjusting their properties (delay and capacity) based on the predefined equations. Contact Tables, which can be pre-calculated and reused across various simulations, significantly enhance computational efficiency, especially in numerous node scenarios. By functioning as cache structures, they improve simulation performance and eliminate the need for repetitive computations. In this way, MeteorNet captures the dynamics of communication nodes in a satellite constellation and produces a network with continually evolving link parameters.

Table I shows an example of the dynamics and contact table, respectively, of a ground node ($gn11$) in a single-link mode that changes the established connection with three satellites ($st10$, $st9$, and $st8$) over time. We can observe that the table defines the contact state at each moment with all the other nodes. The value of each cell is zero when the link is not available, e.g., when there is no LoS or when $\theta_{gs} < \theta_C$; otherwise, the distance is logged in the cell when available. In this example, $gn11$ has available links with $st10$ and $st9$ at $t = 0s$, and it is connected to $st10$ since it is the closest. Then, it loses connection with $st10$ and connects to $st9$ at $t = 400s$, and finally loses connection with $st9$ and connects to $st8$ at $t = 1000s$.

C. Network Management and SDN Controller

In MeteorNet, each satellite is treated as a switch, establishing variable connections (none, one, or multiple) with ground nodes over time. Satellites are also interconnected through intra-plane and inter-plane ISLs. Multiple switch connections could potentially lead to network loop errors due to the creation of loops and multiple paths. To circumvent this, the framework incorporates a Ryu SDN controller and utilizes a straightforward Spanning Tree Protocol (STP) [23] to regulate switch routing and avoid loops. Ryu comes with several routing algorithms [19] and offers software components and an API for custom control application development. Nonetheless, alternate SDN controller libraries can be employed, provided they are compatible with the OpenFlow switch protocol. The Ryu controller monitors the network topology and dynamically configures the satellite nodes by capitalizing on the SDN paradigm.

The control messages within Ryu are processed in an out-of-band mode, meaning they are transmitted through a separate channel distinct from the one used for data messages. While data is transmitted through ground-satellite and inter-satellite links, control packets utilize the Docker internal network, assuming a direct connection between network switches and the central controller. While this assumption may not be entirely realistic for dynamic nodes within a satellite constellation, it serves as an initial approach for testing SDN controllers within satellite nodes avoiding extra complexities in initial scenarios.

D. Containerized Applications

MeteorNet employs Docker to create isolated environments for satellites and ground nodes. Each Node class has a host method that encapsulates a Docker process using a pre-built Linux image with the necessary test software. This arrangement allows each Docker image to be allocated its network interface, CPU virtualization, and memory stack, managed by the OS kernel. Docker isolation ensures consistency when transitioning the software to the production environment, reducing potential code alterations and debugging during deployment. In conjunction with Mininet, MeteorNet simulates network interfaces, routing-switching behavior, and computer resources, enabling upward simulation from the second layer of the OSI model. The physical layer remains the sole layer not directly simulated but is instead approximated by using Equations (1), (3), and (2). The test software can encapsulate any code or algorithm operating in satellites or ground nodes, such as flight software, platform control algorithms, mission control software, payload modules, and ML tasks, to name a few.

E. Network Analytics

The emulation framework incorporates sFlow commands for real-time network usage measurements. As an industry-standard network and resource monitoring tool, sFlow provides a flexible framework for periodic evaluations and computations.

For instance, a sFlow query like *ipsource, ipdestination, link:inputifindex* can be configured on each switch to measure

Table II
NETWORK MEASUREMENT DATA STRUCTURE

Id	Destination IP	Source IP	Interface	Value(bps)
######	10.0.0.2	10.0.0.11	st1-st2	1091

Table III
FRAMES DATA

Id	Source IP	Ethernet Protocol	Value (bps)
######	10.0.0.1	2048 (TCP)	1010

traffic between two IPs across every link interface. Upon completing a simulation, we can freely communicate between nodes and automatically gather the resulting measurements in a centralized database. Table II shows an example of the data structure collected from the sFlow command configured above. Another example is illustrated in Table III, depicting the data structure returned when employing the *ipdestination* and *ethernetprotocol* sFlow keys. In this particular instance, the Ethernet protocol ID 2048 corresponds to TCP.

The command provided in the example utilizes three sFlow keys: '*ipsource*', '*ipdestination*', and '*inputifindex*' to configure network switches for data measurement. By selecting these specific keys, the nodes will continuously monitor the utilization of links that correspond to the same source and destination IP addresses. Various statistics, such as averages, minimums, maximums, or percentiles, can be incorporated into the query to measure the desired metrics. The modifier *link:inputifindex* preceding the *inputifindex* key is a function that instructs the command to return the link name instead of the default, which is an identifier. An illustrative sFlow query can be composed using the following parameters:

```
{Command Atribute : Example}
command =
{name: pair,
 keys: ipsource,ipdestination,
 value: bytes, values: frames,
 filter: ipsource=10.0.0.1}
```

More complex examples of sflow commands for data acquisition would be:

```
c1 = {keys: link:inputifindex,direction,
link:outputifindex,vlansource
value: frames
filter: ipdestination=10.0.0.11}
c2 = {keys: outputdiscardreason,
link:inputifindex
value: rate:bytes}
```

Query c1 will generate a data structure similar to the firt structure in Table IV, displaying the number of frames per link interface, filtered by *ipdestination=10.0.0.11*. Meanwhile, query c2 will produce a data structure resembling the second structure, showing the rate (in bytes per second) of discarded packets on each link interface. For a comprehensive list of sFlow keys, functions, and filters, please refer to the standard documentation [18].

IV. CASE STUDY

A. Scenario

We show a case study with a constellation of ten satellites (from $st1$ to $st10$) in the same orbital plane (Orbital Inclination: 90°, Right Ascension of the Ascending Node (RAAN): 144°, eccentricity: 0) and two ground stations ($gs11$ and $gs12$), configured in single-link mode. We use a laptop computer with an Intel core i7 processor (4 cores, 8 threads) and 16 GB of RAM to run the emulation. We consider two test applications for the network analysis.

(i) A conventional ping application via ICMP packets from ground nodes $g11$ and $g12$ to satellite $st2$. In such a case, we measure the ping times and compare them with the related theoretical values;

(ii) Two task generators, i.e., an application located on $g11$ and $gn12$ generate and offload a task via TCP connections to a MEC server on $st2$. The generators compute and return the required outputs. Both ground stations produce dummy task requests equivalent to 200 devices according to a Poisson distribution - around two tasks per minute per device. The MEC server then queues the task and serves with parallel workers the task on demand that takes approximately $250ms$ of processing time on one core of the server. We measure the task computation time as the time elapsed from when the ground station sends the task until it receives the result. Given that TCP confirms packets and Ryu establishes valid SDN paths, we anticipate discrepancies between the theoretical and measured task computation times.

We also monitored the network usage across each interface link on the routing path between ground stations, satellite links, and inter-satellite links.

B. Analysis

a) Initial State: At the emulation's start time ($t = 0$ seconds), $gn12$ and $gn11$ are connected to the network through $st7$ and $st9$, respectively, and every satellite is connected to the two nearest satellite neighbors. Tasks transmitted from $gn11$ traverse the path $st7$-$st6$...-$st2$, five hops, 23,000 km end-to-end path length. Instead, tasks transmitted from $gs12$ traverse the path $st9$-$st8$...$-st2$, seven hops, 32,200 km end-to-end path length. Both paths are selected by the STP protocol coded in the Ryu SDN controller. Utilizing Equation (2) and assuming a per-link packet switch computation time of $\delta_0 = 4ms$, we can compute the minimum ping time for $gs11$ as $\Sigma_1^5 2 \cdot \delta_i = 2 \cdot (23000/C + 5 \cdot 4)ms \approx 172.72ms$. Based on the measured results in Figure 2, the ping time at $t = 0$ is approximately $193ms$.

Table IV
C1 AND C2 DATA STRUCTURES

Id	Link Input Interface	Direction	Link Output Interface	Virtual Lan Source	Value (Frames)
######	gn11-st7	ingress	st7-st5	0	30

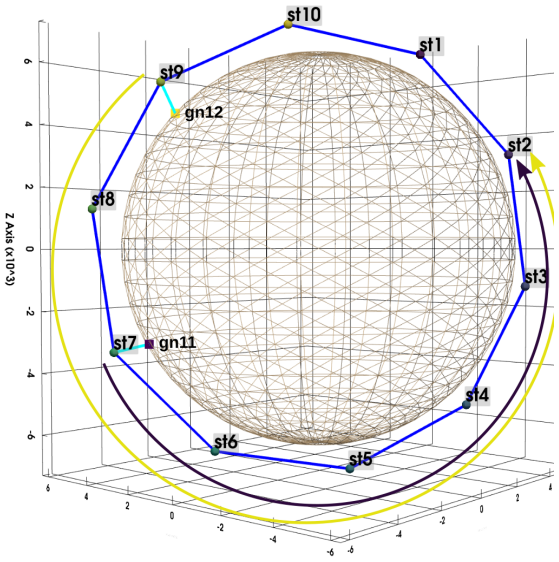

Figure 1. Network Case Scenario at start time ($t = 0$). The MEC server is placed on the far side of the ring constellation in $st2$.

b) Subsequent states: In the time interval when a ground station is connected to the same satellite, computation times experience minor increments due to the increasing distance between the ground station and the linked satellite. Nonetheless, the mean values recorded during contact periods closely correspond with the analytical predictions. We can calculate a gap with the delay computed analytically by analyzing task computation results. In the ideal case for $gs11$, the mean task computation time should be the sum of the theoretical task propagation delay and the task computation time, fixed to 250 ms, which is approximately 423 ms. Nonetheless, in Figure 2, we see an average computing time of about 1,000 seconds, a difference that increases with the number of hops in the network path. The difference can be associated with TCP reliability, which comes with the cost of waiting for the packet acknowledgment before the task processing can occur.

c) Interface utilization data: Further results are shown in Figure 3 regarding the links' network usage by sFlow measurements. The plot illustrates the data collected from around 30 minutes of emulation time. By analyzing such results, we can identify the path chosen by the STP protocol during each contact window and create synthetic data sets to, e.g., train interface utilization prediction algorithms.

d) Multi-Path Data: The only inter-satellite link not used at the beginning of the emulation is $st9$-$st10$. This means that the STP protocol blocks this hop to avoid a cycle within the network. From a delay optimization perspective, it would be better to block $st2$-$st3$, which leaves 5 hops for $gn11$ and 3 hops for $gn12$. The STP protocol leverages link capacity as a cost metric when constructing the network tree. As such, it does not optimize the delay along network paths but rather optimizes the path capacity and prevents loops by blocking interfaces that would cause multiple paths. When two interfaces of the same capacity exist, STP randomly selects which path to maintain and which to block. This can occasionally result in choosing paths with long delays. Given that delay is a critical KPI for NTN-5G and beyond, our findings indicate that the STP routing protocol is not ideally suited for satellite constellation networks. Modifications should be made to the routing protocol to optimize the delay. These hop selection choices are properly recorded using sFlow commands and can be leveraged as synthetic datasets.

V. CONCLUSIONS

Given the increased complexity of space missions, there is a growing interest in designing and developing emulation tools to assist in mission planning for satellite operations.

To address this challenge, we have constructed MeteorNet, a flexible constellation emulation framework based on a continuous-time paradigm. This framework accurately emulates the complete protocol stack in Non-Terrestrial Network (NTN) scenarios. MeteorNet incorporates a central database and configurable sFlow queries, allowing for the definition of automatic measurements to be stored during various scenarios. As the user can customize the emulation tool to gather relevant network data, it can be valuable for generating synthetic datasets and testing Machine Learning (ML)-based solutions.

By utilizing a specific case study scenario and selecting STP and TCP as network and communication protocols, we demonstrate that these standards would require significant adaptations to effectively apply in an NTN constellation. STP, for instance, should prioritize latency for specific use cases instead of focusing solely on optimizing channel capacity. In contrast, TCP's excessive use of acknowledgment requests leads to increased computational latency when employed as a Multi-Access Edge Computing (MEC) communication protocol. Whichever the case, MeteorNet can generate representative datasets to assess the behavior of NTN constellations.

Future work involves developing and testing dynamic ML-based routing and task distribution techniques to optimize task computing delays within the considered Software-Defined Networking (SDN) and MEC satellite scenario.

ACKNOWLEDGEMENT

This work was supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART").

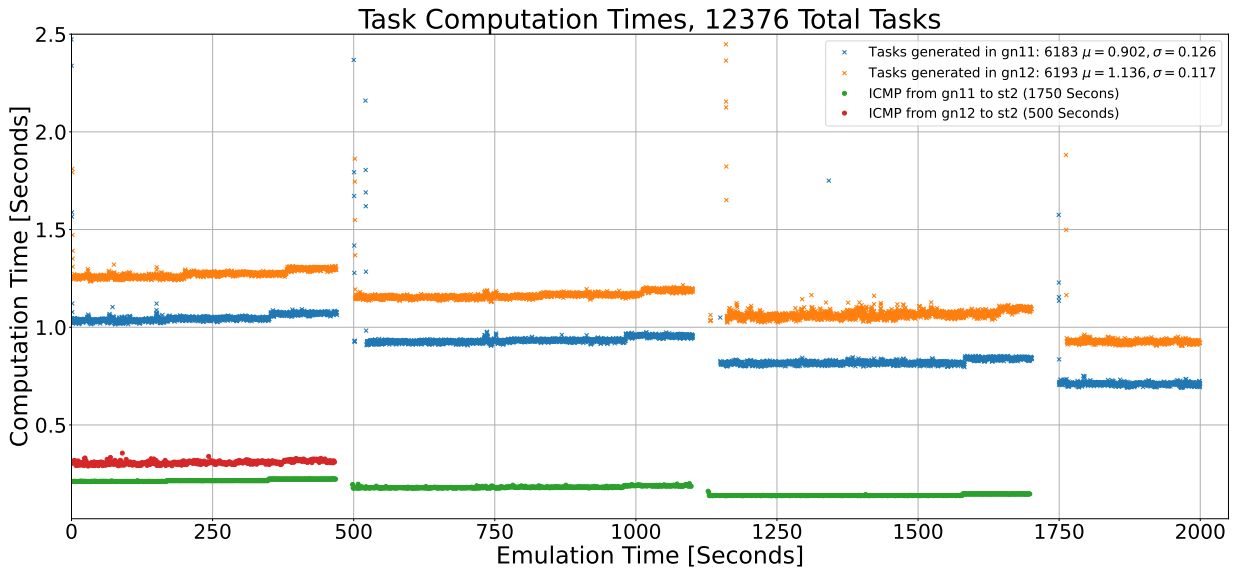

Figure 2. Total task computation times of the test applications.

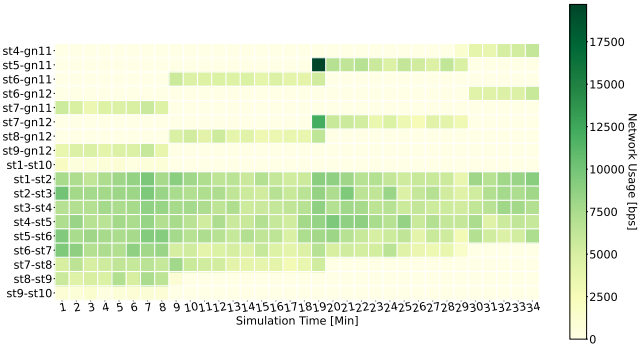

Figure 3. Per-interface network usage.

References

[1] S. Lee, A. Hutputanasin, A. Toorian, W. Lan, R. Munakata, J. Carnahan, D. Pignatelli, and A. Mehrparvar, "Cubesat design specification rev. 13."

[2] 3GPP, "Solutions for NR to support non-terrestrial networks (NTN)," *Technical Report 38.821*, pp. 1–28, 2019.

[3] Z. Zheng, J. Guo, and E. Gill, "Swarm satellite mission scheduling & planning using hybrid dynamic mutation genetic algorithm," *Acta Astronautica*, vol. 137, pp. 243–253, 2017.

[4] "Meteornet." [Online]. Available: https://gitlab.com/camilo.rojas/satellite_constellation.git

[5] "Mininet." [Online]. Available: http://mininet.org/

[6] F. Fourati and M.-S. Alouini, "Artificial intelligence for satellite communication: A review," *Intelligent and Converged Networks*, vol. 2, no. 3, pp. 213–243, 2021.

[7] Y. Miao, Z. Cheng, W. Li, H. Ma, X. Liu, and Z. Cui, "Software defined integrated satellite-terrestrial network: A survey," in *Space Information Networks*. Springer, 2017, pp. 16–25.

[8] W. Jiang, "Software defined satellite networks: A survey," *Digital Communications and Networks*, 2023.

[9] Y. Wang, J. Yang, X. Guo, and Z. Qu, "Satellite Edge Computing for the Internet of Things in Aerospace," *MDPI Sensors*, vol. 19, no. 20, 2019.

[10] Z. Zhang, W. Zhang, and F.-H. Tseng, "Satellite Mobile Edge Computing: Improving QoS of High-Speed Satellite-Terrestrial Networks Using Edge Computing Techniques," *IEEE Network*, vol. 33, no. 1, pp. 70–76, 2019.

[11] T. Pfandzelter, J. Hasenburg, and D. Bermbach, "Towards a computing platform for the LEO edge," in *International Workshop on Edge Systems, Analytics and Networking*, 2021-04-26, pp. 43–48.

[12] C. Sonmez, A. Ozgovde, and C. Ersoy, "Fuzzy Workload Orchestration for Edge Computing," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 769–782, 2019.

[13] P. Cassara, A. Gotta, M. Marchese, and F. Patrone, "Orbital Edge Offloading on Mega-LEO Satellite Constellations for Equal Access to Computing," *IEEE Communications Magazine*, vol. 60, no. 4, pp. 32–36, 2022.

[14] D. Paikowsky, "What is New Space? The Changing Ecosystem of Global Space Activity," *New Space*, vol. 5, no. 2, pp. 84–88, 2017.

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

[16] A. Varga, "Omnet++," in *Modeling and tools for network simulation*. Springer, 2010, pp. 35–59.

[17] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments," *Computer Networks*, vol. 62, pp. 122–136, 2014.

[18] "sFlow." [Online]. Available: https://sflow.org/

[19] R. P. Team *et al.*, *RYU SDN Framework-English Edition*. RYU project team, 2014.

[20] S. Singh and N. Singh, "Containers & docker: Emerging roles & future of cloud technology," in *International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, 2016, pp. 804–807.

[21] D. Vallado, P. Crawford, R. Hujsak, and T. Kelso, "Revisiting spacetrack report #3," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. American Institute of Aeronautics and Astronautics, 2006.

[22] L. Yan and N. McKeown, "Learning Networking by Reproducing Research Results," *ACM Computer Communication Review*, vol. 47, no. 2, 2017.

[23] IEEE, "Standard for local and metropolitan area networks: Media access control (MAC) bridges," *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, pp. 1–281, 2004.