Continuous Time Emulation for Software-Defined Non-Terrestrial Edge Computing Networks

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

Continuous Time Emulation for Software-Defined Non-Terrestrial Edge Computing Networks

Camilo Rojas*, Juan A. Fraire†‡, Fabio Patrone*, Alberto Gotta§, Mario Marchese*
*University of Genoa, Genoa, Italy
†Univ Lyon, Inria, INSA Lyon, CITI, F-69621 Villeurbanne, France
‡CONICET - Universidad Nacional de Córdoba, Córdoba, Argentina
§Institute of Information Science and Technologies (ISTI), CNR, Pisa

***Abstract*—As satellite constellations for communications rise in importance in both academic and industrial circles, we face the challenge of adapting Software-Defined Networks (SDN) and Multi-access Edge Computing (MEC), initially designed for static terrestrial networks, for the dynamic nature of Non-Terrestrial Networks (NTN). This paper presents MeteorNet, a continuous time emulation framework grounded in realistic orbital propagators, Mininet, and Docker virtualization, designed to develop and test space mission software. Our solution adapts link characteristics according to constellation orbits and Earth dynamics using Linux network interfaces. We provide the results of a tested networking scenario detailing how network usage measurements and output data analysis can be used to examine the behavior of routing protocols within the framework. This work underscores the necessity for appropriate network management and transport protocol adaptations for effective communication in emerging satellite constellations.**

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

I. Introduction

The emergence of Low Earth Orbit (LEO) satellite constellations is among the major developments within the space industry. Through the use of Commercial Off-The-Shelf (COTS) components, agile methodologies, and standardized launch facilities (including P-POD containers), the industry has opened to new participants with limited budgets, enabling them to conduct intricate space missions previously restricted to space agencies and large telecommunication companies.

The advent of private entities in the space industry has paved the way for establishing mega-constellations in LEO, encompassing hundreds or even thousands of interconnected satellites through Inter-Satellite Links (ISL). Their objective is to enhance global Internet access [1], supplementing terrestrial networks to reach areas where expanding cellular networks is impractical or uneconomical [2]. This integration of airborne and space segments with terrestrial networks leads to the inception of the Non-Terrestrial Network (NTN) concept, incorporated into 5G standards from Release 16 onwards [3].

However, to incorporate NTNs effectively into 5G networks, they must meet the Key Performance Indexes (KPIs) established by the 3GPP concerning capacity, delay, and service availability. Software-Defined Networks (SDN) are a critical enabling technology adopted to meet 5G KPIs in the space context. Consequently, numerous research works, such as those by Kodheli et al. [4] and Guidotti et al. [5], have been undertaken to integrate NTN into 5G. Despite its potential, SDN poses several challenges. Current frameworks were primarily constructed for stable internet topologies, not dynamic and ever-evolving satellite topologies. Therefore, successfully adapting SDN for this new context presents a considerable obstacle to overcome.

While an efficient SDN layer is crucial, a high degree of automation is required to maximize the service potential of mega satellite constellations. Multi-access Edge Computing (MEC) is a promising solution for addressing current computational challenges [**?**]. It potentially facilitates the efficient allocation and distribution of computational and memory resources, both terrestrial and space-based, across a satellite constellation. Such a framework could provide real-time services to terrestrial users, including machine learning applications to enable autonomous land, air, and maritime vehicles on a global scale. Nonetheless, implementing MEC in satellite constellations is non-trivial and necessitates the development of intelligent algorithms to manage these resources effectively [6]. Addressing these real-time scheduling problems involving SDN and MEC can become increasingly complex with larger constellations.

This paper explores the incorporation of SDN and MEC into current and upcoming mega-constellations. We introduce a unique open-source[1] multi-access edge computing emulation framework for non-terrestrial software-defined networks: MeteorNet. Among other features, MeteorNet provides a dockerized environment for deploying and assessing flight and ground code. This environment replicates the communication conditions during operations, controlled via SDN techniques and keeping native kernel and network management source code, allowing for exploring realistic MEC use cases.

The remainder of the paper is structured as follows: Section II reviews basic concepts and related works, Section III delves into MeteorNet, Section IV presents a case study, and Section V outlines conclusions and future research directions.

II. Background

A. Basic Concepts

a) Software Defined Networking: SDN is a network management strategy segregating the data from the control plane. In SDN architectures, the control plane consists of one

[1]Repository URL will be provided in the camera-ready version of the paper.

or more controllers that possess the capacity to adapt routing based on service requirements [7]. This centralized control paradigm allows the controller to manage the network, where the governed nodes, referred to as SDN switches, are linked to the central processing unit, where all decision-making processes are executed and relayed to the switches.

Attempts have been made to tailor SDN management for satellite networks. Miao et al. [8] and Jiang et al. [9] have reviewed some of these research endeavours. They perceive SDN and Network Function Virtualization (NFV) as catalysts for integrating NTN-5G, suggesting the deployment of multiple SDN controllers, interconnected through bound interfaces, for complex and expansive networks. In this configuration, each controller would be accountable for managing a sub-network. This methodology underpins a two-tiered structure that could segregate segments with independent SDN controller placement.

b) Multi-access Edge Computing: MEC is a distributed computing model designed to bring storage and computing resources closer to the user. The goal is to reduce computational delays and conserve network bandwidth. Wang et al. [10] define MEC as a novel paradigm for accessing IoT applications within satellite networks. Their findings demonstrate that space-edge computing consumes less time and energy than conventional satellite constellations. MEC requires deploying multiple servers in network hosts that deliver virtual functions catering to specific services with low computation delay requirements.

Zhang et al. [11] introduce MEC to improve QoS and speed performance within NTNs, proposing task scheduling models to facilitate cooperative computation in MEC servers. Pfandzelter et al. [12] discuss the unique characteristics of LEO MEC and evaluate the appropriateness of three constellations (SpaceX, Amazon, and Telesat) for implementing distributed computing MEC. Numerous offloading strategies exist in current literature. For instance, Sonmez et al. [13] propose a fuzzy orchestration for MEC offloading that mimics the intuition of real-world administrators, resulting in an automated management system. Cassara et al. [14] compares task-offloading strategies across different constellation scenarios, concluding that fuzzy strategies yield superior outcomes compared to round-robin and full-offloading solutions.

B. Emulation Tools

Planning and monitoring resources are essential for satellite missions, requiring mission control to continuously update scheduling plans to respond to mission changes or natural phenomena [16]. Emulation and simulation tools are used throughout mission planning stages, and actual mission data is later incorporated into these models to improve logistic and operational planning. These are indispensable tools for testing and deploying software and algorithms in space missions.

Various research teams and space agencies are currently developing and maintaining several simulation and emulation tools, as summarized in Table I. OS3 [**?**], SNS3 [**?**], and SCNE [**?**] base their network emulation on well-known frameworks such as OmNet++ (C++) and NS3 (C++, Python), which are powerful discrete-time network simulation libraries that allow users to define various properties from the physical to the application layer. In contrast, SatEdgeSim [15] represents the only MEC framework for satellite networks to test offloading algorithms. It is built on the EdgeSim library (written in Java) and utilizes Java messaging to simulate networks. Regarding orbital propagation, OS3 and SCNE employ custom-made algorithms, while SCNE utilizes the proprietary software System Tool Kit (STK). A common feature of these tools is their use of a discrete-time simulation paradigm to emulate constellation networks.

Current discrete-time simulation frameworks, while helpful, require significant adaptation of mission software and system architecture due to their abstract representation of computer networks. This can lead to potential coding errors during deployment. These tools also lack the capability to emulate network interfaces and OS kernels, creating a gap between simulation and real computing systems. We propose developing a new emulation framework to bridge this gap and more accurately reflect actual system deployment.

III. METEORNET

To address the aforementioned challenges, we present MeteorNet, a continuous-time emulation platform for satellite constellation networks. MeteorNet offers a significant advantage over existing tools by incorporating network and operating system virtualization, including kernel and network management native source code. While discrete-time tools boast replicability and time acceleration, continuous-time frameworks like MeteorNet offer unparalleled fidelity to real-world scenarios and facilitate scalable, multi-thread-driven development.

a) Overview: MeteorNet utilizes Docker for OS virtualization, ensuring flexibility in the choice of programming languages and compilation architectures for the software under test. Built on open-source code and publicly accessible algorithms, it supports all phases of constellation mission planning, from research and development to operation. MeteorNet enables the testing of offloading and routing strategies. The software itself is primarily written in Python. MeteorNet is characterized by the following key components: (*i*) an Orbit Propagation module, managing and simulating satellite motion within its orbit; (*ii*) a Network Stack and Link Management system, handling network protocols and communication links; (*iii*) an SDN Controller, centralizing and overseeing network operations; (*iv*) Containerized Software, utilizing Docker for OS virtualization to enable software instantiation and isolation; and (*v*) Network Performance Monitoring, collecting network performance metrics for further analysis and optimization.

b) SDN Aspect: In the proposed framework, we include Ryu as an example of an SDN controller and utilize the Spanning Tree Protocol (STP) as a default mechanism to manage network routing and prevent switch loops [**?**]. The SDN controller operates in parallel with the emulation, permitting alterations to assess different routing algorithms or optimize

Table I: Comparison among available satellite simulation and emulation tools

	OS3 [?]	**SNS3** [?]	**SCNE** [?]	**SatEdgeSim** [15]	**MeteorNet** (our contribution)
Language	C++	C++,Python	C++, Python	Java	Python
Libraries	OMNet++	NS3	STK, NS3	EdgeSim	SGP4, Mininet, Docker
Time	Discrete	Discrete	Discrete	Discrete	Continuous
Network Virtualization	No	No	No	No	Yes
OS Virtualization	No	No	No	No	Yes
Open Source	Yes	Yes	No	Yes	Yes

service-specific KPIs. The framework leverages Mininet for network virtualization.

c) MEC Aspect: In our proposed framework, we prioritize the effective 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 [17], 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.

d) Workflow: In our proposed framework, the requisite inputs are satellite Two-Line Element (TLE) files, 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

In our approach, we utilize the Simplified General Perturbations Model 4 (SGP4) 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 $C(t)$ can be estimated 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.

B. Contact Table Computation and Utilization

MeteorNet employs Mininet to emulate a realistic virtual network, incorporating a genuine Linux network kernel, switch, and application code [18]. 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

Table II: 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
1100	0	0	1524

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.

a) Example: Table II 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 of time 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, MeteorNet incorporates the Ryu SDN controller and utilizes a straightforward Spanning Tree Protocol (STP) [**?**] to regulate switch routing and avoid loops. Ryu comes with several routing algorithms [**?**] 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. In MeteorNet, the Ryu controller continuously monitors the current network topology and dynamically configures the satellite nodes by capitalizing on the SDN paradigm.

D. Containerized Applications

MeteorNet employs Docker to create isolated environments for satellites and ground nodes. Each Node class is equipped with 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 own network interface, CPU virtualization, and memory stack, managed by the OS kernel. Docker's use of 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 simulation from the second layer of the OSI model upwards. The physical layer remains the sole layer not directly simulated but is instead approximated using Eqs. (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 machine learning tasks, to name a few. The key requirement is that the code must be compatible with a Docker image and use network protocols that align with the Linux network.

E. Network Analytics

MeteorNet incorporates sFlow commands for real-time network usage measurements. As an industry-standard tool for network and resource monitoring, sFlow provides a flexible framework for periodic evaluations and computations. For instance, an sFlow query like *ipsource, ipdestination, link:inputifindex* can be configured on each switch to measure traffic between two IPs across every link interface. Upon completion of a simulation, this allows us to ping between nodes and gather the resulting measurements.

IV. Case Study

A. Scenario

We demonstrate a case study involving a constellation of ten satellites (from $st1$ to $st10$) in the same orbital plane and a single ground station ($gs11$), configured in single-link mode. To run the former simulation, we use a laptop computer with an Intel core i7 processor (4 cores, 8 threads) and 16 GB of RAM. We consider two test applications:

(i) A conventional ping application via ICMP packets from ground node $g11$ and a satellite $st6$. In such a case, we measure the ping times and compare them with the related theoretical values;

(ii) A task generator, i.e., an application on $g11$ that generates and offloads a task via a TCP connection to a MEC server on $st6$, which computes and returns the result. $g11$ produces - according to a Poisson distribution - around two tasks per minute, which take approximately $250ms$ of processing time on MEC. We measure the task computation time as the time elapsed from when $g11$ sends the task until it receives the result. Given that TCP involves acknowledgements, we anticipate discrepancies between the theoretical and actual measured task computation times. We also monitored the network usage across each interface link on the routing path between $g11$ and $st6$ throughout the 60-minute simulation.

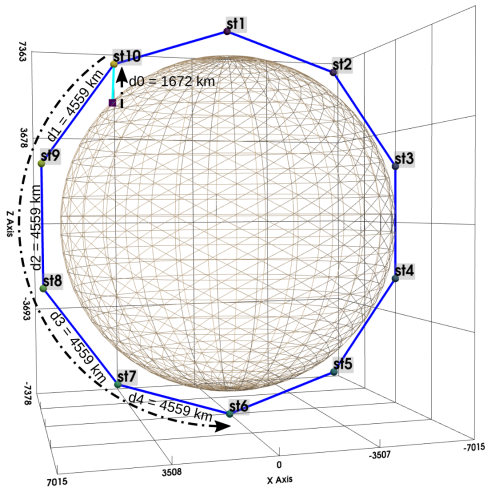

Figure 1: Network configuration and task routing path.

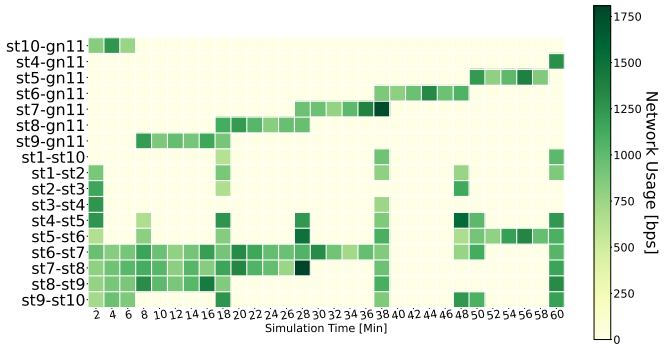

Figure 2: Per-interface network usage.

B. Analysis

At the simulation's commencement ($t = 0$ seconds), $gn11$ is connected to $st10$ and each satellite within the constellation is connected to its two nearest neighbors within the same orbital plane. Tasks transmitted from $gn11$ to $st6$ must traverse a path of $19,908$ kilometers through $g11$-$st10$-$st9$-$st8$-$st7$-$st6$, which represents the path selected by the STP, as depicted in Figure 1.

Utilizing Equation (2) and assuming a per-link packet switch computation time of $\delta_0 = 4ms$, we can compute the ping time as $\Sigma_{i=0}^{4} 2 \cdot \delta_i = 2 \cdot (19,908/C + 5 \cdot 4ms) \approx 172.72ms$. Based on the measured results presented in Figure 3, the ping time at $t = 0$ is approximately $180ms$, which aligns closely with the analytical estimation. At $t = 330s$, when the ground link transitions from $st10$ to $st9$, the $st10$-$st9$ link, which covers approximately 4559 kilometers, is eliminated, leading to a decrease in the measured propagation time of $\Delta\delta = 2 \cdot (4,559/C + 4ms) \approx 38.39ms$. This results in a total theoretical delay of $134.33ms$, which is consistent with the measured value. During the time frame between one ground link change and the subsequent one, measured ping times exhibit slight increases due to the growing distance between $g11$ and the associated satellite. However, the average values during contact closely align with the analytical estimations.

Analyzing task computation results we can calculate a gap with ICMP results. In the ideal case, mean computation time of tasks should be ICMP delay sum with base task computation time (250 ms), nonetheless, in Figure 3 we see an overhead of approximately 100 ms, that increases per hop in the network path. The difference can be associated to TCP reliability, that comes with a cost for waiting the acknoledgement of packets before process tasks.

Further results are shown in Figure 2 in terms of the links' network usage by sFlow measurements, which illustrates the data collected from a 60-minute simulation. By analyzing such results, we can identify the path chosen by the STP protocol during each contact window. Until minute 6, the chosen path is $gn11 \rightarrow st10 \rightarrow st9 \rightarrow st8 \rightarrow st7 \rightarrow st6$. Between minutes 6 and 8, we observe a period of network configuration, during which various interfaces operate in flood mode (a state where they broadcast packets they receive to every other network segment). Due to a change in the ground-satellite contact from $g11$-$st10$ to $g11$-$st9$, the path changes to $gn11 \rightarrow st9 \rightarrow st8 \rightarrow st7 \rightarrow st6$.

From minutes 8 to 16, the network usage remains stable until the ground-satellite link switches again, this time from $g11$-$st9$ to $g11$-$st8$. Similar patterns are observed with other link changes until minute 38 when $g11$ is directly connected to $st6$. In this case, the time is minimal since no ISL is involved in the communication. However, from minute 48 onwards, the time begins to increase again as direct connectivity is lost and a new path incorporating an ISL link ($st5$-$st6$) is selected.

The STP protocol leverages link capacity as a cost metric when constructing the network tree. As such, it does not optimize for delay along network paths, but rather optimizes for 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 paths with long delays being chosen.

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. To optimize for the delay, modifications should be made to the routing protocol used.

V. Conclusions

In response to the growing scale and complexity of satellite missions, we've developed a continuous-time constellation framework, leveraging network and operating system virtualization. This flexible simulation framework allows for direct implementation and real-time testing of SDN and MEC protocols. Through a case scenario utilizing STP and TCP, we found these technologies need adaptations for effective use in constellation environments. STP should prioritize delay for certain use cases rather than solely optimizing channel capacity. TCP's overuse of acknowledgment requests increases computing delay when used as a MEC communication protocol. Our findings underscore the need for research and mod-

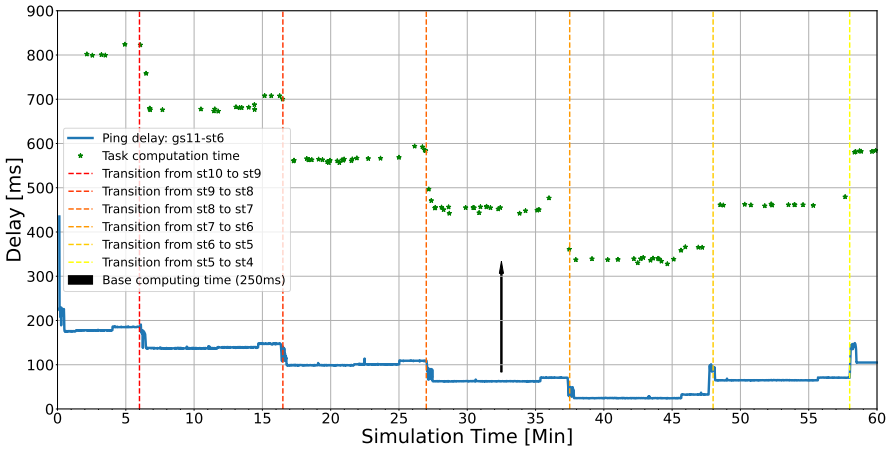

Figure 3: Task propagation and total task computation times of the test applications.

ification of networking protocols for satellite constellations. This will ensure these networks can deliver global, reliable, and efficient communication. Future work includes developing dynamic routing techniques and optimizing task computing delays within the SDN and MEC frameworks.

ACKNOWLEDGEMENT

This research has received support from the European Union's Horizon 2020 R&D program under the Marie Skłodowska-Curie grant agreement No 101008233 (MISSION project), the French National Research Agency (ANR) under the project ANR-22-CE25-0014-01, and the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART").

REFERENCES

[1] I. Del Portillo, B. G. Cameron, and E. F. Crawley, "A technical comparison of three low earth orbit satellite constellation systems to provide global broadband," *Acta astronautica*, vol. 159, pp. 123–135, 2019.

[2] I. del Portillo, S. Eiskowitz, E. F. Crawley, and B. G. Cameron, "Connecting the other half: Exploring options for the 50% of the population unconnected to the internet," *Telecommunications Policy*, vol. 45, no. 3, p. 102092, 2021.

[3] *3GPP, "Solutions for NR to support non-terrestrial networks (NTN)", TR 38.821 v0.6.0, 2019.*

[4] O. Kodheli, A. Guidotti, and A. Vanelli-Coralli, "Integration of satellites in 5g through LEO constellations," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. IEEE, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/8255103/

[5] A. Guidotti, A. Vanelli-Coralli, M. Conti, S. Andrenacci, S. Chatzinotas, N. Maturo, B. Evans, A. Awoseyila, A. Ugolini, T. Foggi, L. Gaudio, N. Alagha, and S. Cioni, "Architectures and key technical challenges for 5g systems incorporating satellites," vol. 68, no. 3, pp. 2624–2639. [Online]. Available: https://ieeexplore.ieee.org/document/8626457/

[6] 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.

[7] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (SDN): a survey," vol. 9, no. 18, pp. 5803–5833, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.1737. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1737

[8] 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*, Q. Yu, Ed. Springer Singapore, pp. 16–25.

[9] W. Jiang, "Software defined satellite networks: A survey," p. S2352864823000299. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2352864823000299

[10] Y. Wang, J. Yang, X. Guo, and Z. Qu, "Satellite edge computing for the internet of things in aerospace," *Sensors*, vol. 19, no. 20, p. 4375, 2019.

[11] Z. Zhang, W. Zhang, and F.-H. Tseng, "Satellite mobile edge computing: Improving QoS of high-speed satellite-terrestrial networks using edge computing techniques," vol. 33, no. 1, pp. 70–76. [Online]. Available: https://ieeexplore.ieee.org/document/8610431/

[12] T. Pfandzelter, J. Hasenburg, and D. Bermbach, "Towards a computing platform for the LEO edge," in *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*. ACM, pp. 43–48. [Online]. Available: https://dl.acm.org/doi/10.1145/3434770.3459736

[13] C. Sonmez, A. Ozgovde, and C. Ersoy, "Fuzzy workload orchestration for edge computing," vol. 16, no. 2, pp. 769–782. [Online]. Available: https://ieeexplore.ieee.org/document/8651335/

[14] P. Cassara, A. Gotta, M. Marchese, and F. Patrone, "Orbital edge offloading on mega-LEO satellite constellations for equal access to computing," vol. 60, no. 4, pp. 32–36. [Online]. Available: https://ieeexplore.ieee.org/document/9755271/

[15] J. Wei, S. Cao, S. Pan, J. Han, L. Yan, and L. Zhang, "SatEdgeSim: A toolkit for modeling and simulation of performance evaluation in satellite edge computing environments," in *ICCSN 2020 12th International Conference on Communication Software and Networks*. [Online]. Available: https://ieeexplore.ieee.org/document/9139057

[16] D. Paikowsky, "What is new space? the changing ecosystem of global space activity," vol. 5, no. 2, pp. 84–88. [Online]. Available: https://www.liebertpub.com/doi/10.1089/space.2016.0027

[17] S. Singh and N. Singh, "Containers & docker: Emerging roles & future of cloud technology," in *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, pp. 804–807.

[18] L. Yan and N. McKeown, "Learning networking by reproducing research results," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 2, pp. 19–26, 2017.