MiniSatnet: An Open-Source Emulator for LEO Satellite Constellations

Jorge Mata-Díaz, Juanjo Alins, Israel Martin-Escalona, Enrica Zola, Oscar Esparza
Universitat Politècnica de Catalunya (UPC), Spain

***Abstract*—Reliable and reproducible experimentation is essential for developing network architectures, protocols, and applications for Low Earth Orbit (LEO) satellite constellations and Integrated Space–Terrestrial Networks. Since experimentation with operational constellations is costly and provides limited control, emulation platforms are required to evaluate real networking software under dynamic satellite conditions. This paper presents MiniSatNet, a container-based emulator that separates the generation of time-varying orbital and network states from their real-time execution. MiniSatNet reproduces changes in connectivity and link conditions while processing real traffic through Linux networking stacks, enabling the evaluation of forwarding mechanisms, transport protocols, applications, and measurement tools. MiniSatNet and its HypatiaSeam state-generation component are released as open-source software together with the experimental and result-processing tools used.**

***Index Terms*—LEO satellite constellations, network emulation, Software-in-the-Loop, dynamic topology, orbital propagation**

I. Introduction

Low Earth Orbit (LEO) satellite constellations have attracted considerable interest because they can provide lower propagation delays than traditional geostationary systems while extending broadband connectivity to areas that are difficult to serve through terrestrial infrastructure. Commercial systems and planned constellations such as Starlink, OneWeb, Kuiper, and Telesat also aim to integrate satellite connectivity with existing operator networks. This integration is expected to become an important component of future Integrated Space–Terrestrial Networks (ISTNs).

LEO satellite constellations consist of multiple satellites orbiting at altitudes ranging from 250 km to 2000 km, with corresponding orbital periods between 90 to 120 minutes. The satellites of the constellation follow their own orbits with predictable patterns. By knowing the orbit parameters of each satellite, it is possible to compute their positions, as well as the inter-satellite distances and the distances between them and the Ground Stations (GS), at any given time.

The benefits of LEO systems are accompanied by substantial networking challenges. For instance, in Walker Star constellations, at the point where two extreme orbits meet, the satellites in those orbits counter-rotates and therefore they form a seam. This seam is problematic as it prevents the establishment of inter-satellite links (ISLs) across it. This and many other problems cause that ground coverage, ground-to-satellite links (GSLs), and some inter ISLs change continuously. Although this evolution is largely predictable, it produces time-varying propagation delays, frequent handovers, and recurrent changes in the forwarding topology. Routing and transport protocols designed for comparatively stable terrestrial networks may experience route instability, transient disconnections, and performance degradation.

Experimenting directly with operational constellations is expensive and provides limited control over their internal configuration, so researchers rely on simulation and emulation platforms. Discrete-event simulators provide reproducible and scalable environments for analysing network behaviour, but usually execute modelled protocol implementations. Network emulators instead process real packets through operating-system network stacks, allowing to use unmodified routing daemons, transport protocols, applications, and measurement tools. However, a LEO network emulator must reproduce orbital mobility, dynamic connectivity, and time-varying link characteristics while keeping pace with wall-clock time.

Orbital and network-state generation need not be performed by the same component that processes network traffic. Satellite trajectories are predictable and can be used to precompute a sequence of network states describing satellite positions, visibility conditions, link parameters, handovers, and, when required, forwarding decisions. These states can subsequently be replayed by a real-time network emulator. This separation avoids placing orbital calculations and packet processing within the same real-time execution loop.

This paper presents MiniSatNet, a container-based network emulator for LEO constellations, together with HypatiaSeam, its companion orbital and network-state generation component. The two components implement the separation between the generation of time-varying constellation states and their real-time execution. HypatiaSeam is our enhanced version of the Hypatia framework [4], incorporating substantial extensions and improvements. HypatiaSeam preserves Hypatia's snapshot-based workflow while extending its constellation, visibility, ISL, GSL, seam, and contact models.

MiniSatNet replays the generated network states using Containernet [2], a fork of the Mininet network emulator [8] that supports Docker containers as hosts within emulated network topologies. Satellites and GSs are instantiated as containerised nodes running independent Linux network stacks and software components. A pre-instantiated *TopoSwitch* fabric reproduces changes in ISL and GSL connectivity through forwarding-state updates rather than through repeated creation and removal of virtual interfaces [1, 3, 5, 6, 9]. Linux traffic control independently applies the delay, bandwidth, jitter, and packet-loss conditions associated with each satellite link. This design allows real network applications, transport protocols, and

979-8-3195-0167-7/26/$31.00 ©2026 IEEE

measurement tools to be evaluated under time-varying LEO conditions.

The particular MiniSatNet use case investigated in this paper evaluates the feasibility of operating a LEO satellite constellation as an end-to-end switched Ethernet network. This approach is inspired by the Layer-2 Carrier Ethernet services announced for the Telesat Lightspeed constellation, which are intended to extend terrestrial operator networks through space-based connectivity [10]. Rather than executing a distributed routing daemon [6], each emulated satellite node runs an Open vSwitch instance that forwards Ethernet frames between the ports connected to its satellite links. This node-level Open vSwitch is distinct from the TopoSwitch fabric, which reproduces the time-varying connectivity of the emulated constellation. For each snapshot, centrally precomputed end-to-end paths are translated into per-node forwarding rules and updated when the predicted topology changes.

The main contributions of this work are as follows:

- We introduce HypatiaSeam, a state-generation component derived from Hypatia that produces reproducible orbital and network states for Walker Star and Walker Delta constellations, including time-varying ISL and GSL connectivity and link characteristics.
- We also introduce MiniSatNet, a real-time Software-in-the-Loop execution platform based on Mininet, Containernet, Docker containers, TopoSwitch, Open vSwitch, and Linux traffic control. The platform instantiates satellites and GSs as nodes with independent Linux network stacks and executes unmodified network applications, measurement tools, and protocol implementations.
- We implement an end-to-end switched Ethernet forwarding scheme in which paths are precomputed from the predicted constellation topology and translated into per-node Open vSwitch rules. The scheme is evaluated in a dynamic LEO scenario using delay, bitrate, jitter, packet-loss, and snapshot-processing measurements.
- We release MiniSatNet and HypatiaSeam as open-source software through a single public GitLab repository [7]. The repository contains the orbital and network-state generation software, the emulation platform, experimental configurations, measurement and result-processing tools, and the artefacts associated with the results reported in this paper.

II. HypatiaSeam: Orbital and Network-State Generation

HypatiaSeam generates the orbital and network states subsequently replayed by MiniSatNet. Performing these calculations before the experiment removes orbital propagation, visibility analysis, contact selection, and centralised path computation from the real-time execution loop.

A. Extension of Hypatia

Hypatia consists of two main components. Its Python-based `satgenpy` framework generates satellite constellations, time-varying topology information, ground-station coverage, and forwarding states. These data are subsequently consumed by `ns3-sat-sim`, which performs packet-level simulation in ns-3. MiniSatNet retains and extends the state-generation workflow, but replaces the ns-3 execution stage with a real-stack network emulator. The original Hypatia implementation generates just Walker Delta constellations and uses a $+Grid$ ISL pattern[1]. Ground-to-satellite visibility is derived from the satellite and ground-station positions.

HypatiaSeam extends `satgenpy` to support a wider range of constellation and operational constraints. An explicit orbital epoch was added to anchor time-dependent position and Earth-rotation calculations. The original GSL distance approximation was replaced by a spherical geometric model, and the minimum communication elevation can be configured by the user. The right ascension of the ascending node (RAAN) range can span either 180° or 360°, allowing the generation of polar Walker Star and Walker Delta configurations. Satellite phasing and the angular separation between adjacent planes are also configurable. For polar constellations, HypatiaSeam explicitly represents the seam between counter-rotating terminal planes. Inter-plane links can be disabled above a configurable latitude to avoid geometrically impractical ISLs near the poles. The GSL model also limits the number of simultaneous satellite connections according to the available ground-station antennas. When more satellites are visible than interfaces are available, a contact-selection procedure chooses the active links from the feasible candidates.

B. Snapshot Generation

HypatiaSeam discretises the evolution of the constellation into snapshots separated by a configurable interval. Figure 1 summarises this generation process. For every snapshot, it computes satellite positions, active ISLs, visible GSLs, link characteristics, and, when centralised routing is selected, forwarding paths. The output retains the directory organisation and data structure used by Hypatia, facilitating interoperability with existing scenarios and analysis tools. The generated files describe the satellite topology, the satellites in range from each GS, the associated link parameters, and the forwarding state of the network. These files constitute the interface between HypatiaSeam and the MiniSatNet execution plane.

III. MiniSatNet Architecture and Real-Time Operation

MiniSatNet separates the generation of time-varying constellation states from their real-time execution. HypatiaSeam generates before the experiment the sequence of network states describing satellite positions, ISL and GSL connectivity, link characteristics, ground-station contacts, and, when centralised forwarding is selected, end-to-end paths. MiniSatNet subsequently replays these states while real packets are processed through isolated Linux networking contexts, allowing unmodified applications, transport protocols, measurement tools, and forwarding software to operate under the evolving LEO conditions. This separation removes orbital propagation, visibility

[1]In a $+Grid$ each satellite is connected to its two adjacent satellites in the same orbital plane and to satellites in neighbouring planes.

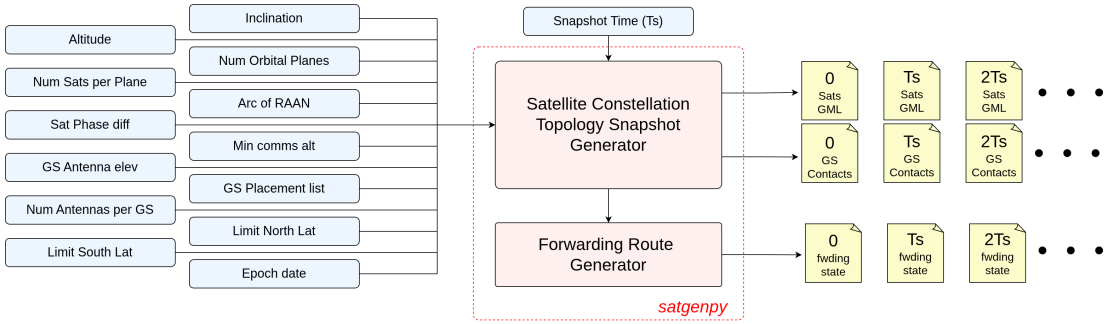


Fig. 1: HypatiaSeam topology and network-state generation process.

analysis, and path computation from the time-critical packet-processing loop.

A central element of this execution model is TopoSwitch, an emulator-internal connectivity abstraction that is distinct from the forwarding plane executed within the emulated satellite and ground-station nodes. Node-facing interfaces are created once during initialisation and remain attached to the TopoSwitch fabric throughout the experiment. Successive ISL and GSL topologies are reproduced by updating the forwarding state of this fabric rather than by repeatedly creating and removing virtual links. TopoSwitch therefore determines which node-facing interfaces are logically connected at each snapshot, whereas each emulated node independently determines how packets or frames are forwarded towards their destinations.

A. Architecture and Network Model

MiniSatNet is implemented using Mininet, Containernet, MiniEvents, Open vSwitch, Docker containers, and Linux traffic control. MiniSatNet comprises three top-level subsystems: the Controller, the Constellation Network, and the Performance Manager. The Controller creates the emulation environment, schedules topology transitions, and coordinates the experiment. The Constellation Network implements the satellite and terrestrial data plane, while the Performance Manager generates traffic and collects and processes measurements. Figure 2 illustrates the relationship between these components. Within the Controller, the *Machine Manager* orchestrates the experiment. It configures the containers, virtual interfaces, TopoSwitch fabric, addressing scheme, and auxiliary services, and coordinates connectivity, link-parameter, and forwarding-state updates. It also preloads the states required for subsequent snapshots.

Within the MiniSatNet node model, the execution context in which packet generators and applications run is represented as a *host*, following the terminology used in Mininet and Containernet. This host is implemented as an isolated Linux network namespace and is separated from the node-level forwarding function. Each emulated satellite or ground-station node therefore combines a Mininet host with a Docker container, providing separate Linux networking contexts for application execution and packet forwarding.

In the switched Ethernet use case considered in this paper, each forwarding node has a local Forwarding Manager that translates its assigned forwarding state into rules for the Open vSwitch instance running within the node container.

For the MiniSatNet use case considered in this paper, simple deterministic IP and MAC addressing schemes encode the node type, node index, interface, and associated virtual link. The complete allocation rules are provided in the MiniSatNet repository.

B. Dynamic Topology Control

A conventional implementation would create and remove virtual Ethernet pairs whenever an ISL or GSL changes. This introduces substantial overhead when many links are modified at each snapshot. MiniSatNet therefore creates the nodes and their node-facing interfaces only once during initialisation.

All nodes remain connected to a common switching fabric managed through the *TopoSwitch* abstraction. In the implementation evaluated here, Open vSwitch provides the packet-processing back end of this fabric. The logical connectivity represented by each snapshot is reproduced by updating its forwarding state, while Linux traffic control independently applies link delay, bandwidth, and packet loss.

MiniEvents schedules the snapshot transitions. During initialisation, the Machine Manager preloads the complete sequence of state changes required by the emulated scenario, thereby minimising file access during the time-critical update intervals. When a transition occurs, the Machine Manager updates the TopoSwitch connectivity and link parameters, publishes the node-specific forwarding states through shared Docker volumes, notifies the affected Forwarding Managers, and triggers any measurements scheduled for that snapshot. The associated events and measurement results are recorded by the logging subsystem for subsequent post-processing analysis.

Each Forwarding Manager monitors its update channel using the Linux `inotify` facility. Upon receiving a notification, it reads the state assigned to its node and updates the local forwarding plane without continuous polling. In the evaluated use case, this operation installs or removes Open vSwitch rules derived from precomputed end-to-end paths. Thus, the Machine Manager coordinates synchronously the sequence of operations associated with each global snapshot transition, while the Forwarding Managers apply the corresponding node-local forwarding decisions.

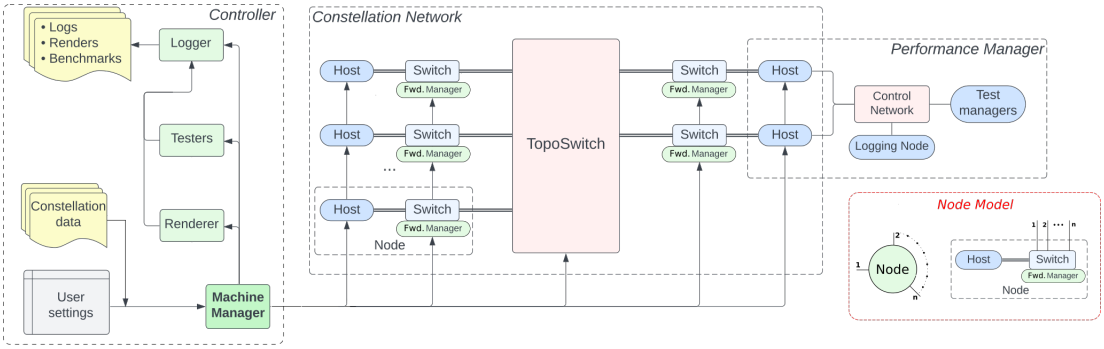


Fig. 2: MiniSatNet architecture.

C. Experimental Workflow

The emulator follows three phases: initialisation, dynamic execution, and result processing. During initialisation, the Machine Manager creates the containers, TopoSwitch fabric, interfaces, addresses, static ARP entries, Forwarding Managers, and auxiliary services. It then loads the initial state and registers snapshot and measurement events with Minievents. During execution, the Machine Manager coordinates snapshot transitions, the Forwarding Managers apply node-specific rules, and the Performance Manager controls traffic and measurements. At the end, traffic processes are stopped, results are collected, and the emulated resources are removed.

The Performance Manager can use a separate control network to isolate measurement coordination from emulated traffic. It uses D-ITG for traffic generation and delay, jitter, bitrate, and loss measurements, as well as `ping`, and `iperf`. Resulting data can be processed using available scripts.

IV. Performance Evaluation

In this section, we do not aim to assess if the performance metrics are good or bad in absolute terms; rather, our objective is to evaluate the MiniSatNet functionality. We pretend to evaluate two aspects: (1) demonstrate that the emulator transfers the effects of the time-varying constellation topology to end-to-end traffic, particularly during route changes associated with the constellation seam; and (2) evaluate if each network-state update is completed within the configured snapshot interval, as required to maintain wall-clock execution.

A. Experimental Scenario

The evaluation considers an end-to-end communication scenario between the GSs located in Panama and Puertollano (Spain). The stations communicate through an emulated classical Iridium LEO constellation with six planes, eleven satellites per plane, an inclination of $86.4°$, an altitude of 781 km, and an ISL latitude limit of $60°$. All these orbital and network states were generated by HypatiaSeam and subsequently replayed by MiniSatNet. The simulation lasts 120 minutes to make sure the constellation has completed at least one full orbital period, and that the simulation captures any situation that might happen. The constellation state was updated every 20 s. At each snapshot, MiniSatNet reproduced the active ISL and GSL connectivity, applied the corresponding link parameters, and installed the required forwarding state. D-ITG generated a continuous flow of 4 Mbps between the GSs, allowing one-way delay, bitrate, jitter, and packet loss to be observed as the topology and selected route evolved.

The scenario was selected because the end-to-end path approaches the constellation seam during the experiment. In a polar constellation, the seam separates the terminal orbital planes, whose satellites travel in opposite directions. When cross-seam ISLs are unavailable, paths must be redirected through other orbital planes, potentially changing their length, hop count, and propagation delay. Figure 3 shows three constellation states around one such transition. At $t = 1780$ s, $t = 1820$ s, and $t = 1860$ s, satellite movement changes the selected path between Panama and Puertollano. The sequence illustrates how the route is redirected as the constellation evolves relative to the GSs and the seam.

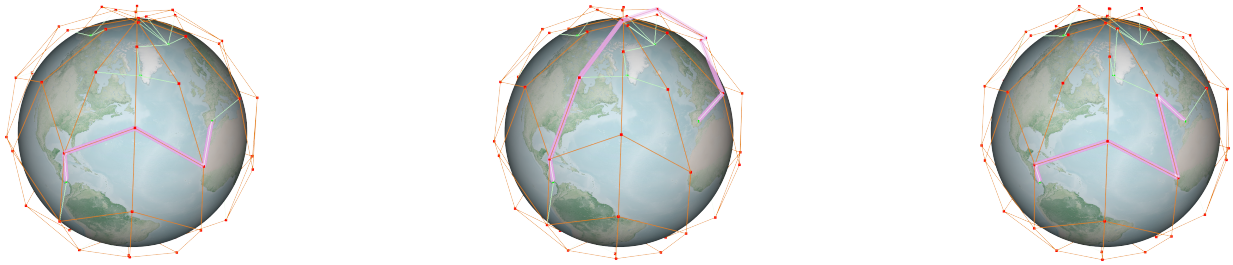
B. End-to-End Network Performance

Figure 4 presents the D-ITG measurements obtained for the Panama–Puertollano flow, including one-way delay (in miliseconds), bitrate (in Kbps), jitter (in ms), and packet loss (number of lost packets). Again, we want to emphasize that we do not pretend to evaluate if these performance metrics are good or bad, but to show that MinisatNet is able to capture the dynamics of the constelation.

We can see that the delay trace exhibits two dominant operating levels associated with alternative end-to-end routes having different propagation distances and hop counts. Transitions between these levels are consistent with the route evolution shown in Figure 3. As the satellites move relative to the GSs and the seam, the previously selected path becomes unavailable or is replaced by a shorter valid path, and MiniSatNet updates the forwarding state accordingly.

The delay variation therefore follows from the network states generated by HypatiaSeam and the corresponding connectivity and forwarding updates applied by MiniSatNet. The agreement between the visualised paths and measured delay supports the conclusion that the emulator transfers the effects of orbital movement and path changes to real packet traffic.

The bitrate, jitter, and packet-loss measurements complement the delay trace by revealing transient effects during topology transitions. Changes associated with GSL handovers, ISL availability, and forwarding-state updates can therefore be



(a) Route at $t = 1780$ s. (b) Route at $t = 1820$ s. (c) Route at $t = 1860$ s.

Fig. 3: Evolution of the Panama–Puertollano route near the constellation seam.

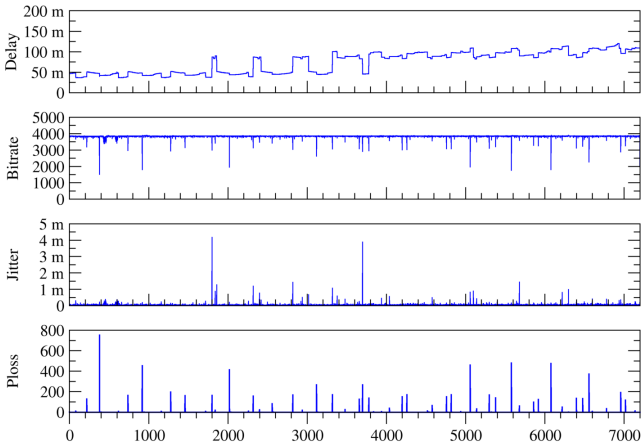


Fig. 4: D-ITG measurements for the Panama–Puertollano flow. Delay and jitter are expressed in ms, bitrate in kbps, and packet loss (Ploss) as the number of lost packets.

distinguished from the stable behaviour between transitions. All measurements were obtained from packets processed by the Linux networking stacks rather than from analytical or discrete-event protocol models.

C. Snapshot-Update Performance

Maintaining real-time operation requires each snapshot to be processed before the next network state becomes active. In this scenario, the processing budget was 20 s per snapshot. The measured update time includes the operations required to modify logical connectivity, install forwarding information, apply link parameters, and coordinate the associated event handlers. Figure 5 shows the Snapshot update time (in miliseconds) of the 360 successive snapshots. The Figure shows that all these update times are far below the configured interval of 20 seconds, allowing MiniSatNet to maintain wall-clock execution throughout the experiment. The observed variation reflects the number and type of operations required by each state. Snapshots containing several GSL handovers, ISL changes, or forwarding modifications require more processing than those involving only a small number of link-parameter updates. The Figure also includes the histogram of the snapshot update time, showing that in most cases the processing time is around half a second, and that no value is above 1.5 seconds. From these results, we can state that, for this simulation and this particular constellation, we could emulate the behavior with a snapshot time of 3 seconds, as the update time per snapshot would be equal or less (there would be fewer changes per snapshot), and we could better capture the constellation's dynamics.

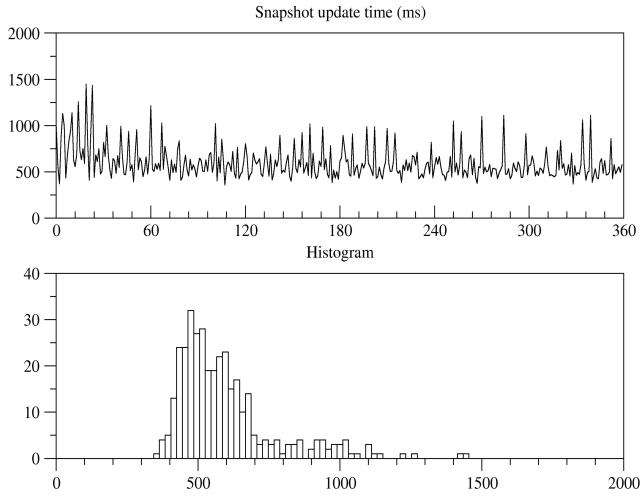


Fig. 5: Snapshot update time for the Panama–Puertollano experiment. The horizontal axis represents the 20 s processing budget imposed by the snapshot interval.

These results show that the pre-instantiated TopoSwitch architecture can reproduce this dynamic Iridium scenario without repeatedly creating or removing node-facing interfaces. By applying each transition through forwarding-state and link-parameter updates, MiniSatNet maintains the required temporal progression while processing continuous traffic.

D. Discussion

The snapshot interval determines both the temporal resolution of the experiment and the processing budget available for each update. A short interval captures topology changes more precisely but increases the frequency of state processing and may require differential or compressed snapshot representations. Conversely, a long interval reduces the processing load

but may omit short-lived states or combine several topology changes into a single update.

Because MiniSatNet represents topology transitions through changes in the TopoSwitch forwarding state, the snapshot interval can be selected according to the required temporal resolution and available processing capacity. The results presented here validate the operation and timing of the proposed architecture for the evaluated scenario; broader scalability claims require experiments with larger constellations, shorter snapshot intervals, and higher traffic loads.

V. Conclusions

This paper presented MiniSatNet, an open-source, Containernet-based emulation platform for dynamic LEO satellite constellations and Integrated Space–Terrestrial Networks. To the best of our knowledge, MiniSatNet is the first open-source LEO constellation emulator to combine Containernet-based node virtualisation with precomputed orbital and network-state generation and real-time replay of dynamic ISL and GSL connectivity. MiniSatNet separates this state generation from real-time network execution. HypatiaSeam, a substantially extended derivative of Hypatia's state-generation framework, produces reproducible snapshots describing satellite positions, ISL and GSL connectivity, link characteristics, ground-station contacts, and, when required, forwarding states. These snapshots are replayed through an environment in which satellite and terrestrial nodes run independent Linux networking stacks, applications, forwarding software, and measurement tools.

A key architectural contribution is *TopoSwitch*, the internal connectivity abstraction used to reproduce topology changes without repeatedly creating or removing node-facing virtual interfaces. In the implementation evaluated in this paper, Open vSwitch provides the packet-processing back end of the TopoSwitch fabric. Logical connectivity is modified through forwarding-state updates, while delay, bandwidth, and packet-loss conditions are applied independently through Linux traffic control. This separation reduces the work required at each snapshot and allows continuous packet traffic to be processed while the emulated constellation evolves.

The Panama–Puertollano experiment over an emulated Iridium constellation showed that MiniSatNet transfers the effects of orbital motion, handovers, and seam-related route changes to real end-to-end traffic. The measured delay variations were consistent with the corresponding path transitions, and all snapshot updates were completed within the configured 20 s interval. The evaluated use case follows the Telesat Lightspeed paradigm by treating the satellite constellation as a single end-to-end switched Ethernet network with centrally computed forwarding. These results validate the functional and temporal operation of the proposed architecture for the considered scenario. A more complete evaluation should also precompute alternative failover paths and install the corresponding Open vSwitch rules in advance, allowing rapid recovery from unexpected link or node failures.

Future work will address larger constellations, shorter snapshot intervals, and higher traffic loads. Planned improvements include more efficient snapshot serialisation, optimisation of forwarding and link updates, and support for additional measurement tools. Further experiments will also investigate alternative TopoSwitch back ends, distributed routing protocols such as those provided by FRRouting [6], additional orbital models, and the scalability limits of the platform.

Acknowledgments

This research has been funded by the Spanish Ministry of Science and Education under the projects TCO-SeamSat (PID2021-124350OB-I00), STELLAR (PID2025-173046OB-I00), and VERIFICO (PID2024-159480OB-I00), by MCIN/AEI/ 10.13039/501100011033/FEDER, UE.

References

[1] Bruce Barbour, Alexander Kedrowitsch, John S. Downs, Deven Mhadgut, Suryansh Aryan, Jonathan Black, and Samantha Parry Kenyon. Spacenet: A resource-efficient simulation testbed for leo mega-constellation networks. *IEEE Access*, 14:28097–28118, 2026.

[2] Containernet Project. Containernet: A mininet fork with support for container-based hosts. GitHub repository, 2026. Accessed: 19 June 2026. [Online]. Available: https://github.com/containernet/containernet.

[3] Andrea Detti, Shahram Dadras, and Giuseppe Tropea. Netsatbench: A distributed leo constellation emulator with an srv6 case study. *ArXiv*, abs/2604.27854, 2026.

[4] S. Kassing, D. Bhattacherjee, A. B. Águas, J. E. Saethre, and A. Singla. Exploring the "internet from space" with hypatia. In *Proceedings of the ACM Internet Measurement Conference*, pages 214–229, 2020.

[5] T. Lu, W. Zhang, M. Deng, T. Huang, and D. Jiang. Opensn: An open source library for emulating leo satellite networks. In *APNET '24: Proceedings of the Asia-Pacific Workshop on Networking 2024*, pages 26–32, 2024.

[6] J. Malgosa-Sanahuja, J. Alíns, J. Ozón S. Machado, and J. Mata-Díaz. Ospf in low earth orbit: Stability, overhead, and routing dynamics. *Computer Networks*, 287, 2026.

[7] J. Mata-Díaz, J. Alins, I. Martin-Escalona, Zola E., and O. Esparza. Minisatnet: An open-source emulator for leo satellite constellations. https://arpasat.upc.edu/seamsat/MiniSatNetHypatiaSeam/-/tree/main, 2026.

[8] Mininet Project. Mininet: An emulator for rapid prototyping of software-defined networks. GitHub repository, 2026. Accessed: 19 June 2026. [Online]. Available: https://github.com/mininet/mininet.

[9] Z. Tang, J. Zhao, H. Li, T. Guo, Q. Wang, H. Chen, and B. Zhao. Mini-Savi: Realistic satellite network simulation platform based on open-source tools. In *Proceedings of the Fourth International Conference on Frontiers of Computers and Communication Engineering*, pages 27–30, Xiamen, China, 2023. IEEE.

[10] Telesat. Telesat Lightspeed LEO Network. Telesat, 2026. Accessed: 19 June 2026. [Online]. Available: https://www.telesat.com/network-emulator/.