SatGoNetEm: an Open Source Emulator of Satellite Constellations and Non-Terrestrial Networks

Juan ARIAS SUAREZ, Louis BARBIER, Marina DEHEZ-CLEMENTI, Oana HOTESCU
ISAE-SUPAERO, *Université de Toulouse*, Toulouse, FRANCE
{juan.arias-suarez, louis.barbier, marina.dehez-clementi, oana.hotescu}@isae-supaero.fr

***Abstract*—SatGoNetEm is an open-source, modular, container-based emulator for large-scale satellite constellations and non-terrestrial networks. It bridges the gap between discrete-event simulation and real-world experimentation by combining high-fidelity constellation dynamics with realistic Linux-based protocol stacks running inside Docker containers. The framework integrates a dedicated topology engine for orbit propagation and time-varying Inter-Satellite and Ground-to-Satellite links connectivity with GoNetEm, a network emulator written in Go, which provides scalable container and virtual-link orchestration. SatGoNetEm supports flexible addressing, automatic link-state management and multiple routing modes, including pre-computed routes and distributed protocols such as OSPF via FRRouting. The framework further supports hardware-in-the-loop setups by bridging container interfaces to Linux veth/bridge devices, enabling direct connection of real network equipment to the emulated constellation. An API-based control plane and web interface enable interactive scenario management and observation. Experiments with realistic LEO constellations show near-linear initialization time, efficient link updates with constant per-link latency, and low steady-state CPU overhead after network construction, making the platform suitable for scalable, protocol-accurate evaluation of next-generation satellite internet systems.**

***Index Terms*—LSN, network emulation, Docker containers**

I. Introduction

The rapid expansion of Low Earth Orbit (LEO) satellite constellations has fundamentally transformed the landscape of global communications and networking. Major commercial deployments such as Starlink [1], OneWeb [2] and Amazon's project Kuiper [3] are establishing thousands of satellites to provide global broadband coverage, creating complex dynamic networks that require advanced evaluation and testing methodologies [4], [5]. As these constellations become integral components of future internet infrastructure, researchers and engineers face increasing challenges in developing and testing protocols and applications for satellite-terrestrial networks.

Traditional network simulators, while valuable for initial protocol development, often fall short when evaluating real-world satellite network scenarios due to their limitations in modeling realistic protocol stacks, container-based applications and distributed system behaviors [6]. Existing satellite network emulators such as LeoEM [7], StarryNet [8], OpenSN [9] and simulators such as Hypatia [10], LEOCraft [11] have made significant contributions to the field, yet they face constraints in scalability, flexibility, and integration with modern network functions like firewalls and other standard Linux-based routing suites such as Free Range Routing (FRRouting) [12].

ISBN 978-3-903176-82-9 ©2026 IFIP

The need for reproducible, extensible and realistic testing environments has become critical as satellite networks transition from research prototypes to operational infrastructure supporting mission-critical applications.

Container-based network emulation has emerged as a promising approach for bridging the gap between simulation and real-world deployment [13]. By using containerization technologies such as Docker [14], researchers can execute unmodified protocol implementations, run distributed applications and achieve horizontal scalability. However, adapting these techniques to the unique challenges of satellite networks, including dynamic topologies, variable link conditions and complex routing requires specialized tools and methodologies.

To address these challenges, we present **SatGoNetEm** [15], a modular container-based network emulator specifically designed for satellite-ground network scenarios. SatGoNetEm is built on top of GoNetEm [16], a general-purpose Go-based network emulator that handles container lifecycle and virtual-link construction, but has no awareness of satellite dynamics or routing semantics. SatGoNetEm contributes the layers above it: SGP4-based orbit propagation, time-varying topology management, satellite-aware addressing and a full routing framework supporting OSPF, MPLS and pre-computed routes. Unlike previous approaches that focus primarily on simulation or basic emulation, SatGoNetEm provides a comprehensive framework that integrates realistic satellite constellation modeling, dynamic topology management, and container-based protocol execution. Our system separates concerns through a modular architecture that decouples topology management, addressing and routing schemes, dynamics and orchestration parameters (e.g., node lifecycle and link impairment scheduling), enabling researchers to focus on protocol development and evaluation rather than infrastructure complexity.

The main contributions of this work are:

- An **open-source, fully implemented modular and scalable emulation platform** named *SatGoNetEm* (Section IV): we design and implement a decoupled architecture that separates topology management, simulation control, time synchronization, and network configuration. Bulk operation of address assignment and batched routing updates reduce the computational overhead of each step.

- A **real-time dynamic satellite constellation module** called *SatComTopology* (Section IV-B): the platform natively embeds a tool that handles time-varying satellite constellation with automatic link state updates, interface management, and routing table synchronization.
- A **flexible routing framework**: users can choose between pre-computed file-based routes, static shortest-path computation, or fully dynamic routing protocols such as OSPF, label-switching mechanisms like MPLS [17] or custom protocols.
- **Container-oriented deployment**: we provide docker images built upon GoNetEm [16], and a gRPC-based [18] orchestration overlay that enables each network node to run in its own isolated container. This approach affords realistic protocol stack emulation and distributed application testing.

The rest of the paper is organized as follows. In Section II, we review existing satellite network simulators and emulators. In Section III, we formalize the system model used to design SatGoNetEm. Then, in Section IV, we introduce and describe SatGoNetEm. Finally, we evaluate SatGoNetEm through comprehensive experiments demonstrating its scalability, performance, and flexibility and accuracy in Section V.

II. Related Work

The evaluation of satellite networking protocols and architectures has been split between discrete-event simulation and virtual emulation. These two methodologies are judged based on metrics such as initialization performance and the accuracy of the network metrics they produce. Furthermore, each offers distinct advantages and limitations such as distributed routing or custom docker images. This section categorizes existing platforms for satellite network evaluation, summarized in Table I, and identifies the gaps that SatGoNetEm addresses.

A. Discrete-Event Simulation for Satellite Networks

Discrete-event simulators have been widely adopted for modeling satellite network behaviors at different levels, where events occur at a specific point in time and marks a change in the system state. Unlike real-time emulation, the simulation clock can advance independently of real-world time, allowing for the rapid execution of long-duration orbital scenarios. These simulators can be broadly classified into *flow-level* and *packet-level* simulators.

1) Flow-Level Simulation: Flow-level satellite network simulators focus on high-level performance metrics such as end-to-end propagation delay and throughput analysis without capturing fine-grained packet forwarding behaviors. As shown in [19], StarPerf exemplifies this approach by providing trajectory simulation, topology generation and shortest-path calculation for LEO constellations. While simulators such as StarPerf offer insight into constellation architecture optimization, they cannot evaluate networking protocols or capture realistic packet-level interactions.

2) Packet-Level Simulation: Packet-level satellite network simulators are typically built on mature discrete-event simulation frameworks such as ns-3 [24] and OMNeT++ [25]. For instance, Hypatia [10] offers an extensive packet-level simulation environment based on ns-3, supporting detailed analysis of congestion control mechanisms and routing protocol behavior in satellite networks. Similarly, [26] expanded packet-level simulations to assess IP-based and Information-Centric Networking architectures in LEO satellite constellations using OMNeT++. Their study highlights the usefulness of such tools for examining networking approaches that diverge from traditional TCP/IP designs.

However, packet-level simulators also exhibit notable drawbacks. The complexity of discrete-event scheduling results in prolonged simulation runtimes, especially for large-scale constellations where routing protocols trigger substantial control traffic and frequent link-state updates. In addition, these tools typically cannot run full, unmodified protocol stacks or real-world applications, which constrains the fidelity and practical relevance of the obtained experimental results.

B. Virtual Network Emulation for Satellite Networks

Virtual network emulation addresses the limitations of simulation by running real protocol stacks and applications, providing more realistic experimental environments. Existing virtual network emulators employ different virtualization technologies, including *lightweight process virtualization (mininet)*, *container technology*, and *virtual machines*.

1) Lightweight process virtualization: Mininet [20] enables the creation of realistic virtual networks with real kernel stacks but lacks specific support for satellite network emulation. To address this, LeoEM, introduced in [7] as part of the SatCP work, extends Mininet to emulate LEO constellations, incorporating StarPerf's trajectory calculation and shortest-path algorithms. However, these lightweight process-based approaches face limitations, as they fall short when evaluating container-based applications or complex distributed system behaviors, as they lack the needed software isolation.

2) Virtual Machine-Based Emulation: Virtual machine (VM) technology offers complete isolation and kernel customization capabilities but at the cost of significant resource overhead. On one hand, LORSAT [21] provides VM-based emulation for LoRaWAN satellite scenarios but supports only bent-pipe architectures without routing capabilities. On the other hand, NEaaS [22] offers Network Emulation-as-a-Service through VM-container hybrid architecture, but the dominance of VMs prevents large-scale network emulation.

Celestial represents a micro-VM approach using Firecracker for LEO edge computing scenarios [23]. While providing better resource efficiency than traditional VMs, Celestial focuses primarily on edge computing rather than comprehensive network protocol evaluation and lacks support for distributed routing emulation.

3) Container-Based Emulation: Container-based emulation has gained traction for its balance between isolation and performance. StarryNet [8] demonstrates container-based

TABLE I
CLASSIFICATION OF EXISTING SIMULATION AND EMULATION TOOLS FOR SATELLITE NETWORKS

Category	Tool	Granularity	Technology	Networking Support	Open Source
Discrete-Event Simulation	StarPerf [19]	Flow-level	Python & Matlab	Route Calculation Only	Yes
	LEOCraft [11]	Flow-level	Python	Optimization & Routing	Yes
	Hypatia [10]	Packet-level	ns-3	Congestion Control & Routing	Yes
Virtual Network Emulation	Mininet [20]	Real Stacks	Process Namespaces	Generic (No Sat Support)	Yes
	LeoEM [7]	Real Stacks	Mininet Extension	Trajectory & Shortest Path	Yes
	LORSAT [21]	Real Stacks	Virtual Machines	Bent-pipe (No Routing)	No
	NEaaS [22]	Real Stacks	VM/Container Hybrid	Emulation as a Service	No
	Celestial [23]	Real Stacks	MicroVM (Firecracker)	Edge Computing (No Dist. Routing)	Yes
	StarryNet [8]	Real Stacks	Docker	Distributed (BIRD)	Yes
	OpenSN [9]	Real Stacks	Docker	Distributed (FRR)	Yes
	SatGoNetEm (Ours)	**Real Stacks**	**Docker**	**Distributed (OSPF/FRR) & Static**	Yes

satellite network emulation using Docker, with distributed routing implemented through BIRD routing software [27] on each container. Similarly, by default, OpenSN [9] implements FRRouting for distributed routing.

These container-based satellite network emulators face several challenges. StarryNet exhibits inefficiencies due to direct interaction with Docker's command-line interface (CLI), leading to unnecessary overhead during container and link management. The lack of optimization in task scheduling and resource utilization further limits scalability for large constellations. Additionally, most existing tools provide only limited extensibility, requiring significant modifications to support new routing protocols or custom network configurations.

To address these limitations, OpenSN introduces an architecture that separates user-defined configuration from container management via a centralized Key-Value Database. Therefore, by streamlining Docker CLI interactions and optimizing the creation of virtual links, OpenSN significantly reduces operational overhead.

Despite its architectural improvements, OpenSN also encounters performance constraints. The platform faces a scalability ceiling, in which a single machine cannot afford to emulate thousands of satellites running resource-hungry applications, needing a complex multi-machine system to overcome hardware limitations.

The heavy resource usage of hardware level virtualization leaves VMs unsuitable for high-density LEO constellations. Since each VM requires a full guest operating system instance, the memory and CPU overhead scales linearly with the number of nodes, creating a larger scalability ceiling than with docker. This limitation needs a shift towards OS-level virtualization (containerization), where nodes share the host kernel. This approach, adopted by SatGoNetEm, corresponds to the container-based approach. It preserves the ability to run unmodified user space routing binaries (such as FRR or BIRD) while reducing the overhead by orders of magnitude.

III. SYSTEM MODEL

This section formalizes the system model used by SatGoNetEm and summarized in Table II, with emphasis on the architectural view of the emulated network and the representation of nodes and links. The goal is to describe the abstract satellite communication system that the framework instantiates, independently of implementation details.

A. Network Architecture

The satellite network is modeled as a multi-layer system comprising two segments. The **Space Segment** consists of a constellation organized into shells and orbital planes, including Low Earth Orbit (LEO), Medium Earth Orbit (MEO) and Geostationary Earth Orbit (GEO) satellites with Inter-Satellite Links (ISLs) and Ground-to-Satellite Links (GSLs). The **Ground Segment** includes user terminals and gateways that provide access to the terrestrial Internet or backbone. Formally, the system is represented as a directed graph where nodes (representing satellites, ground stations and user terminals), are connected by a time-dependent set of active links. The edges capture both ISLs and GSLs, modeling the dynamic nature of constellations where link availability and characteristics shift as satellites move along their orbits. SatGoNetEm realizes each node as a TCP/IP-enabled network entity and each edge as a bidirectional point-to-point link with specific capacity, delay and availability constraints. This model is mapped onto Linux network namespaces instantiated as Docker containers and connected via virtual links. Furthermore, the architecture supports Hardware-in-the-Loop configurations by bridging the emulated environment to physical infrastructure. This is implemented by creating a *veth* pair where one endpoint resides in the container and the other in the host namespace. The host-side endpoint is then attached to a Linux bridge connected to the Network Interface Card (NIC) as shown in Fig. 1.

B. Node model

Each node belongs to one of the two main classes, **Satellites** or **Ground Stations**, and is modeled with a common set of attributes plus type-specific parameters.

Each satellite node is defined by its orbital parameters, specified through configuration files and external libraries. These parameters include the orbital plane index, mean revolutions per day, inclination, and other Two Line Elements (TLE)-derived quantities such as right ascension of ascending node, argument of perigee, and mean anomaly for SGP4-based propagation [28]. Geographically, each satellite follows a moving position on Earth characterized by latitude, longitude and altitude. From a network perspective, each satellite

TABLE II
SUMMARY OF SATGONETEM SYSTEM MODEL AND ARCHITECTURE

Model	Entity	Physical Parameters	Network Representation	Dynamic Behavior
Node	Satellites	Orbital Planes, TLE, SGP4	IP Router (ISL/GSL I/F)	Moving Position $\vec{x}(t)$
	Ground Stations	Fixed Latitude/Longitude	IP Endpoint/Gateway	Static Position
Link	ISL (Sat-Sat)	$d(t) = \|\vec{x}_u - \vec{x}_v\|/c_{eff}$	High Capacity (Fixed)	4 Neighbors (Time-Varying)
	GSL (Sat-Gnd)	Elevation Angle $> \theta$	Bottleneck Capacity	Intermittent (Handovers)
Topology	Graph $G(V, E)$	Node set V (Static)	Edge set $E(t)$ (Dynamic)	Updates at discrete step Δt

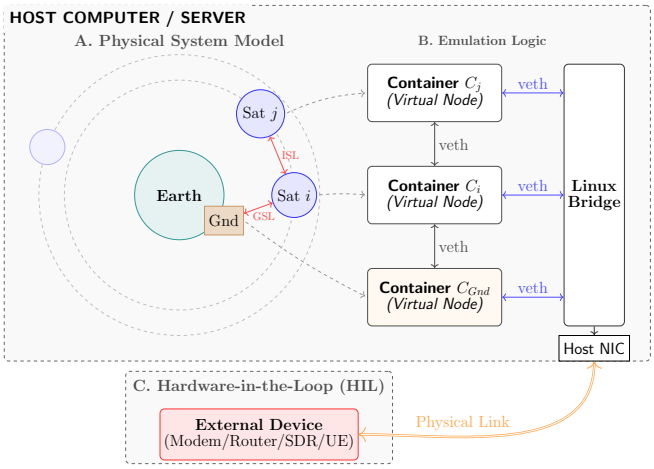

Fig. 1. The satellite system model and its translation to the emulation logic, as well as integration with Hardware-in-the-Loop (HIL).

maintains a set of network interfaces, each assigned specific IP addresses and logical roles, such as ISL or GSL interface. This is done alongside a local routing table that supports static routes derived from the constellation topology. While the system model treats these as generic IP routers with time-varying connectivity, the emulation realizes each satellite as a dedicated Docker container.

Ground station nodes function as terrestrial endpoints or gateways, characterized by a fixed geographic location (latitude, longitude and altitude). Their networking attributes consist of a set of interfaces connected to satellites or terrestrial entities, using IP addressing and routing logic consistent with the project configuration. Like satellites, ground stations are mapped to Docker containers with full TCP/IP stacks, facilitating granular network measurements such as latency, throughput and congestion control behavior.

C. Link model

The network defines links between different nodes through two primary categories: **ISLs**, which connect satellites within the constellation, and **GSLs**, which connect satellites to terrestrial stations. Each active link is characterized by two parameters. Firstly, a propagation delay $d(t)$, which is calculated as the distance between the two node positions $||x_u(t) - x_v(t)||$ divided by the effective propagation speed c_{eff}. Secondly, the capacity is defined by directional values in bits per second, representing the channel capacity between the different nodes according to the user configuration.

For ISLs, the propagation delay is derived from relative satellite positions, while capacity is fixed via configuration. In this model, each satellite typically establishes concurrent connections to four neighbors.

Meanwhile for GSLs, connectivity is governed by an elevation angle where a link is only active if the satellite's elevation angle over the horizon exceeds a threshold defined by the emulator user (*e.g.*, $10\,\text{deg}$). While delay and capacity are derived similarly to ISLs, these links serve as the primary source of bottlenecks in experiments, as their capacity is much lower than that of ISLs. Furthermore, their intermittent availability, driven by both geometric constraints and a user-defined limit on the maximum number of simultaneous links, triggers handover events and dynamic path changes. The emulator user can specifically configure the maximum number of simultaneous links for each terminal or gateway.

The emulator also includes a method to calculate realistic GSL capacities.

D. Time-Varying Topology

Combining the node and link models, SatGoNetEm views the constellation network as a time-varying graph which consists of:

- A node set *V*, static during a simulation, meaning that there is no creation or deletion of satellites or ground stations during a simulation.
- An edge set *E(t)* which evolves over discrete times steps $t_0, t_1, ..., t_T$ with step size Δt configured in the project. At each time step t_k:
  1) Satellite positions $\vec{x_s}(t_k)$ are recomputed from the orbital model.
  2) Visibility, elevation and distance checks determine which links satisfy the constraints.
  3) For each link, attributes are updated.

Thus, the system model supports **dynamic connectivity**, as links may appear or disappear as satellites move, **dynamic routing**, as shortest path routes between ground stations may change over time, and **dynamic performance**, as end-to-end delay and capacity between two endpoints become time-dependent.

IV. SATGONETEM FRAMEWORK

This section presents the design of SatGoNetEm (Fig. 2), a modular, container-based emulation framework designed to decouple the dynamic complexities of constellation modeling from the functional aspects of network emulation.

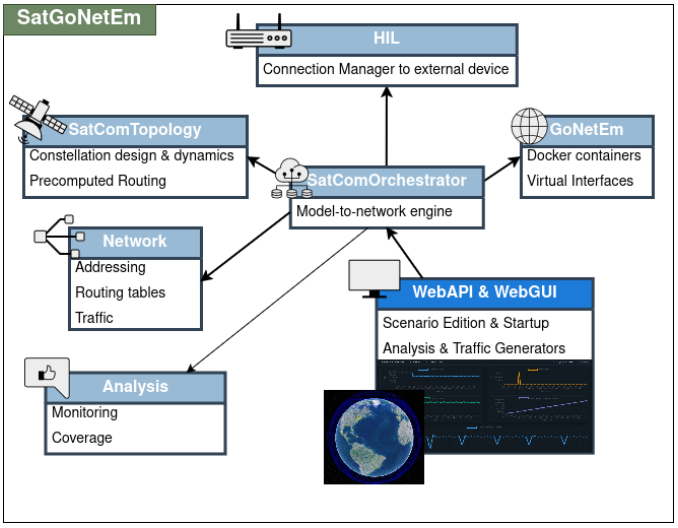

Fig. 2. Modular architecture of SatGoNetEm

A. The Orchestrator (SatComOrchestrator)

SatComOrchestrator is the heart of the framework. Written in Python, it serves as the model-to-network engine that manages the entire emulation lifecycle. Python was chosen for the orchestrator because of its scientific ecosystem (NetworkX, astropy), which directly supports the topology and orbital computations tasks, and its `grpcio` library provides gRPC support. Since all performance-critical operations are delegated to the Go-based GonetEm backend, the interpreted overhead of Python has no measurable impact on emulation fidelity. Its primary function is to transform the time-varying graph model of the constellation, as generated by the ***SatComTopology*** module (see Section IV-B), into active network configurations within the emulation environment. Firstly, it coordinates all major components, including handling the initial setup, time step synchronization and cleanup of the network environment. Then, regarding container orchestration, it utilizes gRPC-based communication for orchestration with the ***GoNetEm*** emulation backend (see Section IV-C). GoNetEm was chosen over direct use of the Docker SDK or higher-level orchestration frameworks for two primary reasons: First, its server architecture exposes a gRPC interface, which allows the orchestrator to control the emulation remotely and programmatically. Second, GoNetEm performs low-level network operations directly via netlink sockets, such as virtual Ethernet pair creation and traffic-control configuration. By contrast, the Docker SDK alone provides no native support for link-impairment creation, and general-purpose orchestrators such as Kubernetes introduce significant overhead. Finally, the orchestrator is designed for scalable implementation, supporting efficient operations for tasks like link creation or update, or address assignment and routing table updates, which is crucial for emulating large-scale constellations.

It is important to distinguish the contributions of SatGoNetEm from those of GoNetEm [16]. GoNetEm manages container lifecycle, virtual-link construction and kernel-level link impairments. SatGoNetEm contributes the layers above it: SGP4-based orbit propagation, satellite-aware addressing, routing semantics (OSPF, MPLS, pre-computed routes), and the orchestrator that closes the loop between constellation dynamics and the live emulation environment.

B. The Dynamics module (SatComTopology)

The ***SatComTopology*** module is responsible for transforming constellation configuration files into a set of active emulation entities representing orbital elements, ground objects, and communication links. It defines a generic entity model that is independent of the network state and any specific implementation. The core functionality of this module is the computation of the constellation state by applying orbital mechanics, updating entity positions over time, and determining connectivity across the constellation.

To compute satellite positions, $\vec{x}_s(t)$, high-fidelity orbit propagation models are used, including the Simplified General Perturbations-4 (SGP4) [28] model based on TLE data. At each discrete time step t_k, the module recomputes the state of the network, which is formally represented as a time-varying graph. To determine the active links at each time step t_k, the module performs geometric checks, including satellite-to-ground station elevation constraints, as well as additional parameters depending on the selected connection strategy. A link is established only if the internode distance and relative orientation satisfy user-defined thresholds for GSLs.

The module also supports the simultaneous modeling of multiple orbital shells and distinct constellations. Users can instantiate hybrid networks combining multiple LEO, MEO and GEO satellites within a single emulation scenario.

Finally, *SatComTopology* can also be used as an independent library. It can be directly imported and used within Python code, or it can generate intermediate *NetworkX* graph representations through a set of adapter submodules.

C. Network Emulation Backend (GoNetEm)

GoNetEm [16] is a network emulator written in Go that uses Docker containers and OpenVSwitch [29] for network construction. It operates as a client-server architecture, which facilitates its use by the SatGoNetEm framework.

The core binary responsible for managing the emulation environment is *gonetem-server*. It directly interfaces with the host system's Linux kernel to perform low-level network tasks, such as creating and launching Docker containers as network nodes and setting up virtual links.

On the other hand, *gonetem-console* acts as the command-line client, which SatGoNetEm replaces with its own Python-based orchestrator and gRPC interface for automated control.

1) Containerization and Node Orchestration: All network elements, including satellites, ground stations and user terminals, are instantiated as dedicated Docker containers. This approach ensures complete isolation and allows these nodes to run a full TCP/IP stack, enabling realistic testing of network functions. GoNetEm supports different Docker-based images to emulate various node roles, and SatGoNetEm takes full advantage of this to create custom images.

On the node side, each element in the topology (satellites, ground stations and user terminals) is instantiated in parallel. Upon loading a topology, GoNetEm walks through node definitions and submits concurrent tasks to create concrete node instances. Once instances are created, a pool of concurrent workers is used to start all containers simultaneously, significantly reducing the setup time for large constellations.

2) Virtual Link Construction: At the link level, GoNetEm models point-to-point connections as *virtual Ethernet (veth)* pairs that span separate network namespaces. For each logical link between two nodes, using low-level *netlink* operations and explicit namespace switching, it then creates a virtual Ethernet pair whose two endpoints reside in the respective nodes namespaces.

Beyond virtual topologies, SatGoNetEm supports hybrid setups where one endpoint of a veth pair is placed in a host namespace or Linux bridge instead of a container. This design makes it possible to connect external interfaces of physical devices to the emulated ISL/GSL fabric, effectively embedding real hardware into the constellation for hardware-in-the-loop testing and protocol validation under realistic, time-varying link conditions.

3) Layered Link Impairments: Once the interfaces are established, link characteristics parameters are enforced through a layered queueing discipline composition applied at the kernel level via *rtnetlink* sockets. Parameters such as delay, jitter and loss are applied via a *netem* [30] discipline, while a Token Bucket Filter (TBF) [31] discipline is used to accurately model rate limiting, by stacking it on top of the netem hierarchy. This layered approach enables independent manipulation of temporal properties via netem and throughput constraints via TBF.

4) Satellite Node Container Stack: The emulated satellite nodes utilize a minimal-base Docker image to ensure a low-overhead network stack supporting both IP routing and advanced Software-Defined Networking (SDN). Routing Functionality (Layer 3) is provided via the FRRouting suite for dynamic protocols like OSPF, while a Programmable Forwarding Plane is enabled via BMv2 [32] and the P4Runtime Interface (PI) [33], allowing for P4-based packet processing logic, as well as MPLS-label switching. For experiment monitoring, the image includes testing tools such as *iperf3* [34] for performance metrics, *hping3* [35] for attack simulation and *tcpdump* [36] for packet-level analysis.

D. Network Module

The ***Network*** module focuses on defining and applying the logical network configuration to the containers managed by GoNetEm. It manages the flexible addressing schemes (IPv4 and IPv6) and the logical roles of network interfaces (ISL, GSL) on each satellite and ground station container. It also offers routing support, in particular: **Static/Precomputed** routing configuration derived from the constellation topology, which can be applied through standard routing tables or MPLS-label switching via source routing, and **Dynamic or Distributed** routing, enabling the execution of distributed routing protocols like OSPF directly within the container.

Finally, to manage traffic and Quality of Service (QoS), the system converts the physical link latencies and capacities, calculated by the ***SatComTopology*** (IV-B) module, into network control configurations to accurately shape the traffic flow between nodes.

E. WebAPI and WebGUI

The control and observation layer of SatGoNetEm is implemented through a structure consisting of a ***FastAPI-based API*** and an interactive ***Web Graphical User Interface (WebGUI)***.

The core control plane is exposed via a FastAPI application, which serves as the backend for all front-end interactions and external scripting. This API ensures that the entire emulation stack is remotely controllable and observable.

The API exposes endpoints corresponding directly to the core services within the ***SatComOrchestrator*** (IV-A), including project management (operations for creating new emulation scenarios, loading configuration, starting/stopping the network...), topology state (retrieval of the current network state, including dynamic satellite positions, active links and node addressing) and dynamic updates (interface to push new configuration parameters, enabling runtime modification of link conditions and node parameters).

F. Additional Capabilities

SatGoNetEm also connects between pure emulation and real-world testing through its ***Hardware-in-the-Loop (HIL)*** capability. External hardware, such as 5G modems, routers, IoT devices or any other user equipment can be physically connected to the emulator via standard network interfaces like Ethernet or Wi-Fi. In this setup, the connected device perceives the emulated satellite link as a real network connection. The emulator applies the dynamic delay, jitter, packet loss and capacity characteristics of the satellite path to the link in real-time. The HIL interface is configured using veth pairs, in which one endpoint resides inside the emulated container, while the other is placed in the host network namespace. This host-side endpoint is then attached to a Linux bridge connected to a physical network interface card, through which external hardware interfaces directly with the emulated constellation fabric.

V. Performance Evaluation

This section assesses the emulation efficiency, scalability, and network fidelity of the proposed emulator. The experimental environment uses a single machine equipped with an Intel Core Ultra 7 155H CPU and 32 GB of memory, running Ubuntu 22.04 LTS with Linux kernel 6.8.0, Python 3.10.12, and Docker Engine 29.3.0.

The following metrics are evaluated to assess efficiency and scalability: *Network construction* (Section V-A), *CPU and memory usage* (Section V-B), and *link state updates* (Section V-C). Network fidelity is assessed through *RTT-based network fidelity* (Section V-D), *Ground fiber comparison*

(Section V-E), considering large-scale constellations given in Table III, for fidelity evaluation.

TABLE III
LEO CONSTELLATION TOPOLOGIES USED IN EXPERIMENT

Constellation	Planes × Sats	Total Sats	Altitude (km)
Iridium [37]	6 × 11	66	780
IRIS^2 (LEO Shell) [38]	12 × 22	264	750
OneWeb [2]	18 × 40	720	1200
Kuiper [3]	34 × 34	1156	630
Starlink (1st Shell) [1]	72 × 22	1584	480

A. Network Construction

For each topology of LEO constellations given in Table III, the measurement covers the total time from starting the emulation script to having all containers and virtual links ready.

Fig. 3 shows the construction time in seconds, categorized by the various stages of the startup process. The total initialization time is dominated by the GoNetEm Initialization, which includes both parallel container deployment (Node Initialization) and virtual interface creation (Link Initialization), while orchestrator initialization, which also includes SatComTopology startup, and addressing remain relatively negligible, staying below 10 seconds even for the largest configurations. The combined initialization time scales linearly with the number of satellites. Specifically, for the Iridium (66 nodes) constellation, the total global initialization is achieved in around 3 seconds. In contrast, for the Starlink Shell 1 constellation (1584 nodes), the total time reaches approximately 95 seconds. These results confirm that the Docker container initialization is the primary bottleneck, accounting for nearly 60% of the total startup time in large-scale scenarios.

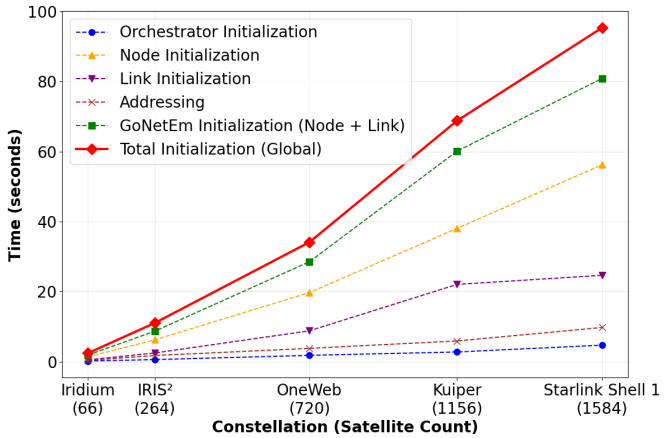

Fig. 3. Initialization performance of SatGoNetEm across five different LEO constellations

B. CPU / Mem Usage

To evaluate the computational overhead of the emulation platform, we monitor CPU and memory utilization across the entire lifecycle of the constellation from initial orchestration to final state. We analyze two scenarios: a small IRIS^2-like constellation (22 × 12 satellites) and a large Starlink-like constellation (72 × 22 satellites).

As shown in Fig. 4, the IRIS^2 constellation exhibits a very short initialization window. The orchestrator and GoNetEm phases occur within the first 15 seconds, characterized by a sharp CPU spike to approximately 80% and a very small increase in memory usage. Due to the relatively small number of nodes, the addressing and routing phase occurs in a second, allowing the system to reach its final state in under 15 seconds total. After that, during the dynamic state (Emulation Start period), in which satellite positions keep changing and links update their state, CPU usage remains almost non-existent, only characterized by very subtle spikes when an update occurs.

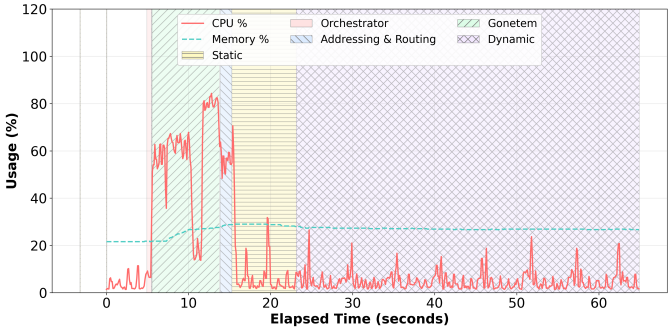

Fig. 4. System resource utilization over time for IRIS^2 constellation.

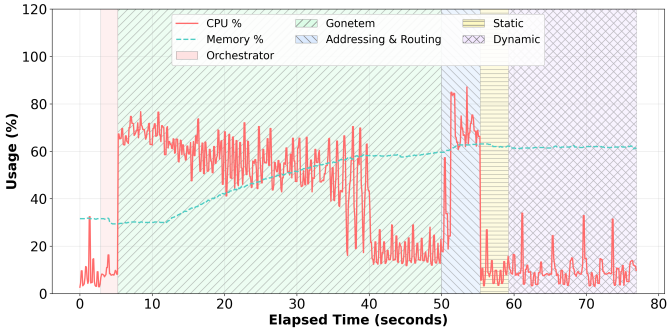

Fig. 5. System resource utilization over time for Starlink constellation.

In contrast, the Starlink-like constellation demonstrates the scalability of the emulator as shown in Fig. 5. During the GoNetEm initialization phase, CPU usage remains high during the first stage (representing Docker container startup), followed by a period of lower CPU activity (during link creation). Notably, memory usage grows linearly during the node startup phase, rising from around 30% to 60% as the 1,584 nodes are loaded. Following GoNetEm startup, the addressing and routing phase shows a second CPU surge peaking at around 85% as the network topology and routing tables are established. After the small period of steadyness, during the dynamic state, update times can be clearly seen due to the sudden spikes in CPU usage, meaning that an update has been triggered.

C. Link State Update

Link creation and update is an important aspect of the emulation in order to properly emulate the time-varying topology.

In fact, the propagation delay and throughput of both GSLs and ISLs changes greatly throughout the emulation as time moves and nodes change positions, getting closer or further away from each other. Therefore, it is necessary to benchmark the efficiency of our emulator in updating and creating hundreds or even thousands of links simultaneously.

Link update focuses mostly on updating the propagation delay of links, done using traffic control settings, while link creation focuses on ground station handover. When a satellite flies over the connectivity area of a ground station, a new link has to be added. Moreover, when the satellites flies outside this area, the link has to be destroyed.

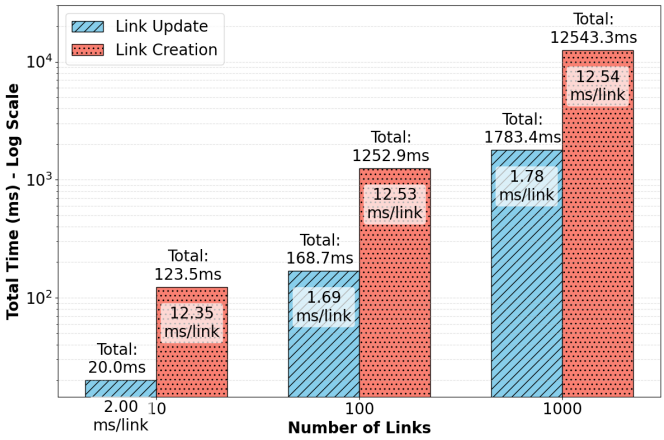

Fig. 6. Scalability of SatGoNetEm link state changes: total execution time and per-link time in milliseconds (ms) for link update versus link creation for 10, 100, and 1000 links

Fig. 6 shows the time it takes for SatGoNetEm to create new links and update existing ones, emulating ground station handover. It can be clearly seen that for any number of links, the time of execution per link remains constant (*i.e.,* around 12.5 ms for link creation and 2 ms for link update), while the total time grows linearly with the number of links to process.

D. RTT-Based Network Fidelity

Beyond construction and update overhead, we evaluate the fidelity of the emulated network by comparing measured round-trip time (RTT) against a theoretical baseline derived from the graph. We run Dijkstra's algorithm to obtain the minimum-delay path between a pair of ground stations to obtain the theoretical end-to-end RTT. In parallel, we measure RTT between the corresponding emulated nodes using ICMP echo requests (ping) between the containers.

Fig. 7 shows the RTT for a pair of ground stations considering the constellations given in Table III. It can be noticed that the measured ICMP latencies closely track the theoretical baseline derived from Dijkstra algorithm. The figure captures the dynamic nature of the network in which the RTT continuously fluctuates, reflecting the changing propagation delays caused by the physical movement of LEO satellites relative to the ground stations and to the handover of ground stations to a different satellite. A handover can be identified by the sudden peak in RTT, showing the ARP discovery

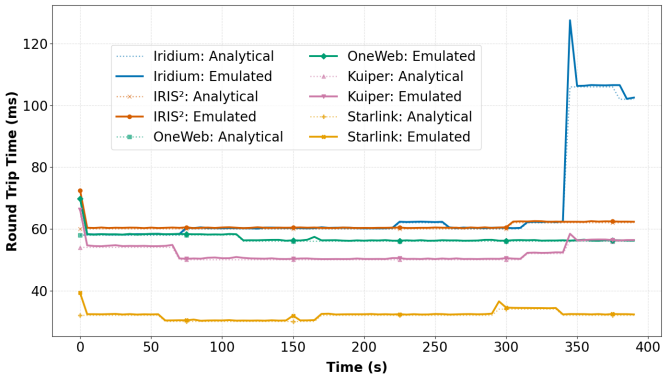

Fig. 7. Comparison of Analytical (Dijkstra) vs. Emulated Round Trip Time (RTT) for multiple LEO constellations between a pair of ground stations (Paris - San Juan).

when a new link is created between a ground station and a satellite. Furthermore, the small difference between emulated and the theoretical baseline is a result of the emulator's design. While Dijkstra represents a purely mathematical calculation of propagation delay, the emulated measurement includes real-world networking overheads.

E. Comparative Analysis: LEO Constellation vs. Terrestrial Latency

To evaluate the potential latency benefits of satellite Networks, we conducted a comparative analysis between our emulated LEO constellations and a terrestrial fiber baseline as highlighted in Fig. 8. The analysis covered over 1,100 global city-pair routes for 47 randomly generated ground stations, providing a comprehensive dataset for benchmarking performance. Furthermore, in this scenario, it is considered that ground-stations can connect to every satellite visible with an elevation angle higher than $10\,\mathrm{deg}$.

Fig. 8 illustrates the RTT comparison for a sample of 10 global routes. The RTTs were compared against a theoretical terrestrial fiber baseline (Ground), calculated using the great circle distance between ground stations adjusted by a refractive index of 1.5 (speed of light in fiber $\approx$ 200,000 km/s).

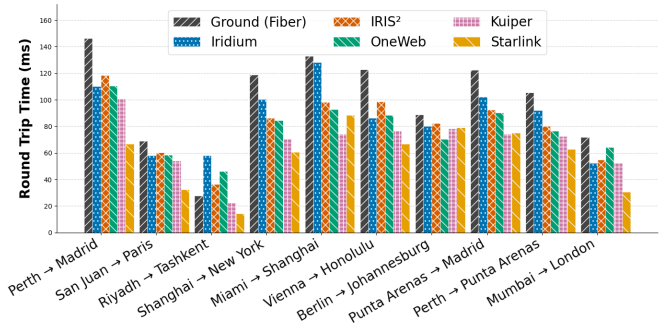

Fig. 8. RTT comparison for 10 randomly selected global city pairs.

The full dataset reveals distinct performance characteristics across network architectures. IRIS² outperformed terrestrial fiber on 71.38% of tested routes, while Kuiper and Starlink

led the field at 84.93% and 82.51% respectively. OneWeb improved on 56.02% of cases due to its higher orbital altitude, and Iridium showed the most balanced profile at 46.54%. Terrestrial fiber retained an advantage on shorter intra-continental routes, where satellite links incurred a significant propagation delay penalty.

VI. Conclusion

In this work, we presented **SatGoNetEm**, an open-source, modular and container-based emulation platform specifically designed for large-scale satellite constellations. By decoupling topology management from network execution, our framework achieves efficient linear scaling, initializing over 1,500 satellites in approximately 95 seconds. The platform's ability to handle dynamic link updates with a consistent per-link latency ensures realistic modeling of time-varying satellite-ground connectivity and handover events. Furthermore, experimental results across five major LEO constellations show high fidelity in capturing temporal dynamics, with measured RTTs closely tracking the theoretical Dijkstra baseline.

Moving forward, we aim to expand SatGoNetEm's capabilities by enhancing the security implementations, advancing routing protocols and leveraging the Hardware-in-the-Loop capability for 5G integration. By providing a realistic testing environment that supports unmodified protocol stacks and modern containerized network nodes, SatGoNetEm enables researchers to validate the next generation of satellite internet infrastructure.

Acknowledgement

The work in this paper is supported by the French Governmental Agency DGA/AID via the eQoSpace (Enhancing Quality of Service from Space in heterogeneous non-terrestrial networks) project. The authors also wish to thank GoNetEm development team from ENAC, Michaël ROYER and Emmanuel LOCHIN, for the insightful discussions and their continuous support.

References

[1] S. Clark, *SpaceX launches more Starlink satellites, beta testing well underway*, Sep. 2020. [Online]. Available: https://spaceflightnow.com/2020/09/03/spacex-launches-more-starlink-satellites-beta-testing-well-underway/.

[2] Eutelsat Group, *OneWeb LEO Constellation: High-speed, low-latency LEO satellite*, https://www.eutelsat.com/satellite-network/oneweb-leo-constellation, Accessed: 2026-02-18, 2026.

[3] Amazon, *Everything you need to know about Amazon Leo, Amazon's satellite broadband network*, https://www.aboutamazon.com/news/innovation-at-amazon/what-is-amazon-project-kuiper, Accessed: 2026-02-18, Feb. 2026.

[4] 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," in *69th International Astronautical Congress (IAC)*, Bremen, Germany, 2019.

[5] M. Handley, "Delay is Not an Option: Low Latency Routing in Space," *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, pp. 85–91, 2018.

[6] N. Cheng et al., "A Comprehensive Simulation Platform for Space-Air-Ground Integrated Network," *IEEE Wireless Communications*, vol. 27, no. 1, pp. 178–185, 2020.

[7] X. Cao and X. Zhang, "SatCP: Link-Layer Informed TCP Adaptation for Highly Dynamic LEO Satellite Networks," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, IEEE, 2023, pp. 1–10.

[8] Z. Lai et al., "StarryNet: Empowering researchers to evaluate futuristic integrated space and terrestrial networks," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI '23)*, Boston, MA: USENIX Association, Apr. 2023, pp. 1309–1324.

[9] W. Lu, Z. Wang, H. Zhang, S. Zhang, and H. Luo, "OpenSN: An Open Source Library for Emulating LEO Satellite Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 8, pp. 1574–1590, Aug. 2025.

[10] S. Kassing, D. Bhattacherjee, A. B. Aguas, J. E. Saethre, and A. Singla, "Exploring the Internet from Space with Hypatia," in *Proceedings of the ACM Internet Measurement Conference (IMC)*, ACM, 2020, pp. 214–229.

[11] S. Basak, A. Pal, and D. Bhattacherjee, "LEOCraft: towards designing performant LEO networks," in *Proceedings of the 2025 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '25, Boston, MA, USA: USENIX Association, 2025.

[12] FRRouting Project, *FRRouting (FRR) Protocol Suite*, https://frrouting.org/, Version 10.5.1, Accessed: January 2026, 2026.

[13] E. Casalicchio and S. Iannucci, "The State-of-the-Art in Container Technologies: Application, Orchestration and Security," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 17, 2020.

[14] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[15] J. Arias Suarez, *SatGoNetEm: an Open Source Emulator of Satellite Constellations and Non-Terrestrial Networks*, https://github.com/SatComProjects/satgonetem.

[16] M. Roy, *GoNetEm: Network emulator written in Go and based on Docker/Open vSwitch*, https://github.com/mroy31/gonetem, 2021.

[17] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," RFC 3031, Jan. 2001. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3031.txt.

[18] The gRPC Authors, *gRPC: A High Performance, Open Source Universal RPC Framework*, Accessed: 2026-02-05, Google, 2024. [Online]. Available: https://grpc.io/.

[19] Z. Lai, H. Li, and J. Li, "StarPerf: Characterizing Network Performance for Emerging Mega-Constellations," in *IEEE 28th International Conference on Network Protocols (ICNP)*, IEEE, 2020, pp. 1–11.

[20] B. Lantz, B. Heller, and N. McKeown, *A Network in a Laptop: Rapid Prototyping for Software-Defined Networks*, Mininet, http://mininet.org/, 2011.

[21] M. Afhamisis, S. Barillaro, and M. R. Palattella, "A Testbed for LoRaWAN Satellite Backhaul: Design Principles and Validation," in *2022 IEEE International Conference on Communications Workshops (ICC Workshops)*, IEEE, 2022, pp. 1171–1176.

[22] J. Lai, J. Tian, K. Zhang, Z. Yang, and D. Jiang, "Network Emulation as a Service (NEaaS): Towards a Cloud-based Network Emulation Platform," *Mobile Networks and Applications*, vol. 26, pp. 766–780, 2021.

[23] T. Pfandzelter and D. Bermbach, "Celestial: Virtual Software System Testbeds for the LEO Edge," in *Proceedings of the 23rd International Middleware Conference*, ACM, 2022, pp. 69–81.

[24] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Güneş, and J. Gross, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34. DOI: 10.1007/978-3-642-12331-3_2. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_2.

[25] OMNeT++ Community, *OMNeT++ Discrete Event Simulator*, https://omnetpp.org/, 2024.

[26] F. Yan, H. Luo, S. Zhang, Z. Wang, and P. Lian, "A Comparative Study of IP-based and ICN-based Link-State Routing Protocols in LEO Satellite Networks," *Peer-to-Peer Networking and Applications*, vol. 16, no. 6, pp. 3032–3046, 2023.

[27] BIRD Team, *The BIRD Internet Routing Daemon*, https://bird.network.cz/, Accessed: 2026-02-02, 2024.

[28] D. Vallado, P. Crawford, R. Hujsak, and T. Kelso, "Revisiting spacetrack report #3," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, AIAA, 2006, p. 6753.

[29] B. Pfaff et al., "The design and implementation of Open vSwitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, Oakland, CA: USENIX Association, 2015, pp. 117–130.

[30] S. Hemminger, "Network emulation with NetEm," in *Proceedings of the 6th Linux Conference Australia (LCA)*, Canberra, Australia, Apr. 2005.

[31] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th. Prentice Hall, 2010, pp. 396–398.

[32] The P4.org Architecture Working Group, *BMv2: The reference P4 software switch*, Accessed: 2026-02-05, 2024. [Online]. Available: https://github.com/p4lang/behavioral-model.

[33] The P4.org API Working Group, "P4Runtime specification," Open Networking Foundation, Specification, version 1.3.0, 2020. [Online]. Available: https://p4.org/specs/.

[34] ESnet, *iPerf3: The Ultimate Speed Test Tool*, Accessed: 2026-02-05, Lawrence Berkeley National Laboratory, 2024. [Online]. Available: https://software.es.net/iperf/.

[35] S. Sanfilippo, *Hping3*, Accessed: 2026-02-05, 2006. [Online]. Available: http://www.hping.org/.

[36] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture," in *Proceedings of the USENIX Winter 1993 Conference*, San Diego, CA: USENIX Association, 1993, pp. 259–269.

[37] S. R. Pratt, R. A. Raines, C. E. Fossa, and M. A. Temple, "An Operational and Performance Overview of the Iridium Low Earth Orbit Satellite System," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1029–1045, 2021. DOI: 10.1109/COMST.2021.3060506.

[38] European Commission, *IRIS²: Secure Connectivity – Infrastructure for Resilience, Interconnectivity and Security by Satellite*, https://defence-industry-space.ec.europa.eu/eu-space/iris2-secure-connectivity_en, Accessed: 2026-02-18, 2026.