

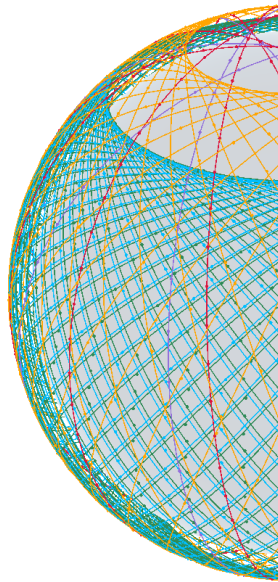
SatComTopology and SatGoNetEm

**A flexible platform for dynamic
satellite network experimentation**

Louis Barbier

October 14, 2026

Excellence with Passion



SUMMARY



01 Context and Problem

02 SatComTopology

03 SatGoNetEm

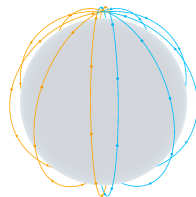
04 Conclusion

01

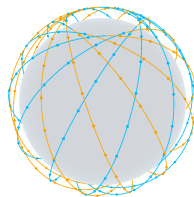
Context and Problem

01 / A network in permanent motion

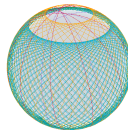
- ▶ Thousands of satellites at $\sim 27,000$ km/h
- ▶ Several shell patterns: Walker Star, Walker Delta, sun-synchronous
- ▶ Ground links change every few minutes (handovers)
- ▶ Optical inter-satellite links



Walker Star



Walker Delta



Starlink Gen1, five FCC shells

Consequence

The network is not a fixed graph: we need tools to model it.

01 / Limits of existing tools

An example: Hypatia (Kassing et al. 2020)

- Packet-level simulation on ns-3
- Constellation generation tailored to one experiment and to its modified network stack
- No runtime state to query or modify

More generally: C++ simulators (ns-3, OMNeT++)

- The constellation model lives inside the simulator and follows its event scheduler
- Changing the topology logic means modifying and rebuilding the simulator

Common limit

Heavily **coupled**: the constellation model is tied to one network stack. These tools do not favor small experiments.

Our position

Separate the **modeling** of the constellation from the **network stack**.

How can **SatComTopology** and **SatGoNetEm** support our NTN experimentation?

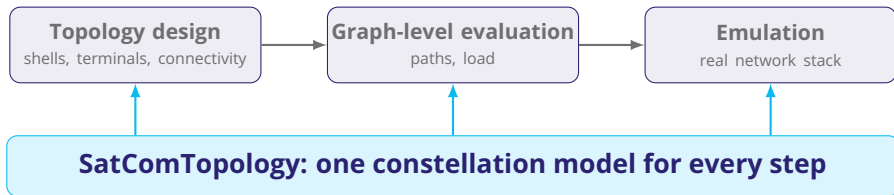
The routing case study

Hop-by-Hop routing, from the scenario to the emulation

02

SatComTopology

Modeling the dynamics of a constellation



- We observed that an NTN study goes through three independent steps
- A tight coupling between model and simulator makes it hard to go through them
- SatComTopology aims to support every one of these steps

Why Python?

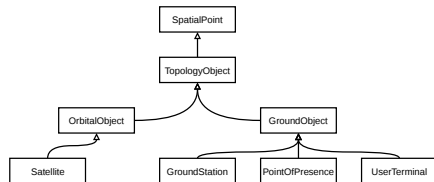
- Accessible: ideal for new contributors and students
- The language of the experiment code: NetworkX, notebooks
- Installed with `pip`: no simulator to rebuild

Domain-Driven Design (Evans 2003)

- Constellation knowledge encoded as entities
- Separation of concerns: no network stack in the model

Open source

- GPLv3 license, on GitHub, with tutorials



The entities of the domain model

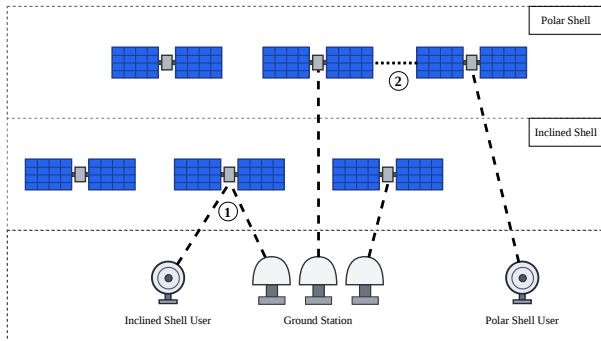
02 / Constellations and ground links

Topology design

Graph-level evaluation

Emulation

- Multi-shell constellations
- Walker Star, Walker Delta
- SSO constellations
- Ground domains, one **first-hop strategy** per domain



Two shells, two ground domains: each user is served by its own shell

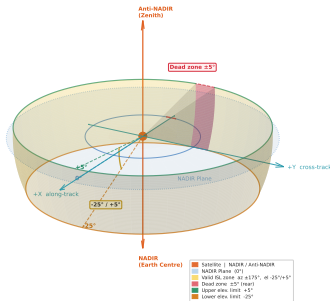
02 / Optical inter-satellite links

Topology design

Graph-level evaluation

Emulation

- **Static topologies:** built from addressing plans
- **Dynamic topologies:** links reassigned at runtime, between shells
- **Physical constraints checked at every tick:**
 - line of sight
 - range
 - slew rate
 - field of regard
 - contact duration



Line of sight of a CONDOR terminal (Carrizo et al. 2020)



Multi-shell constellation: a Walker shell relaying an SSO constellation

02 / One pattern to extend the library

```
# In a notebook: a new first-hop strategy
@register_ground_to_space_connectivity_strategy
class RandomFirstHopStrategy(
    GroundToSpaceConnectionStrategy):
    strategy_name = "random-first-hop"

    def compute_ground_object_connection_strategy(
        self, ground_object, satellites):
        ...

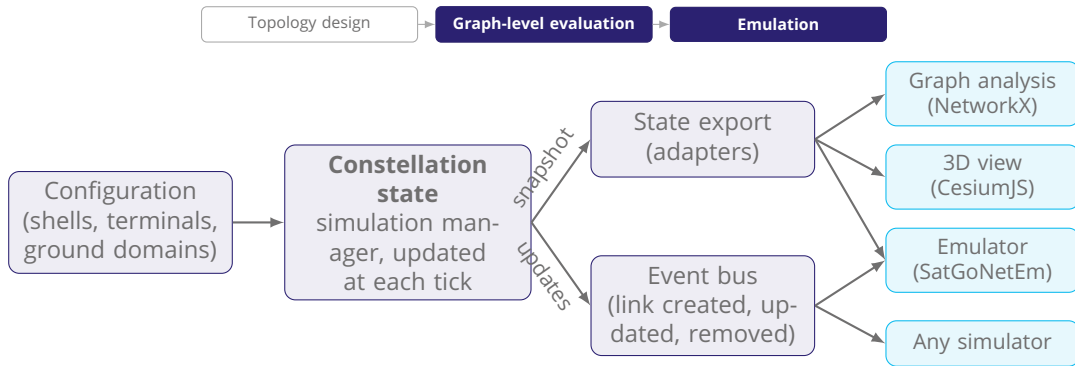
# A new assignment strategy for optical links
class RandomLinkAssignmentStrategy(
    InterSatelliteConnectivityStrategy):
    type = "random-link-assignment"

    def compute_strategy(self):
        ...

manager.inter_satellite_link_connection_strategy = \
    RandomLinkAssignmentStrategy(satellites, plans)
```

- ▶ Ground and optical links follow the same **Strategy** pattern
- ▶ A new behavior = a new class; the core of the library is not modified
- ▶ It can be written directly in a **notebook**
- ▶ Ground strategies are registered with a decorator and selected by name in the configuration

02 / One topology, many consumers



The state can be **exported** at any time, or **followed** through the event bus:
SatComTopology plugs into any simulation or emulation layer.

03

SatGoNetEm

The routing problem

We are interested in **decentralized** algorithms, such as **Hop-by-Hop** routing: each satellite decides the next hop on board.

Reference	Approach	Evaluation
Ekici et al. 2001	Manhattan distance	Simulation
Papapetrou et al. 2008	Deadlock-avoiding	Sim., ns-2
Iglesias-Sanuy et al. 2022	Location-based	Sim., ns-3
Stock et al. 2022	Hop-count heuristic	Sim., custom
Roth et al. 2025	Deep RL	Sim., custom
Zhang et al. 2021	Hierarchical OSPF	Sim., ns-2
Yan et al. 2024	OSPF extension	Emulation

A good example

Routing algorithms are often evaluated in simulation, but rarely **on a real network stack**.

- Simulation hides the protocol and the **cost of implementation**
- An open problem in the literature (Zhu et al. 2025)

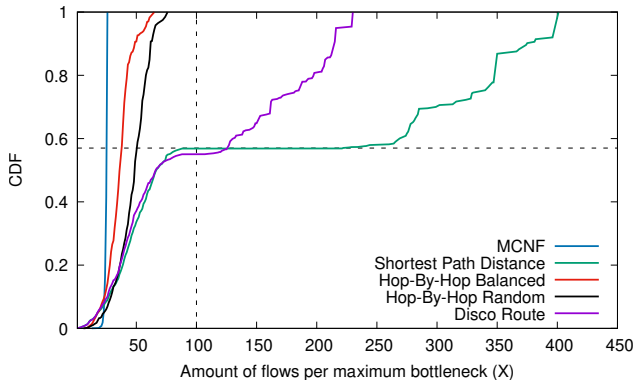
03 / Hop-by-Hop at graph level



- Each satellite chooses the next hop **locally**: closest to the destination on the grid
- On ties, the **least-loaded** link
- 1,600 satellites, 5,000 flows between the most populated cities

Load balancing

Close to the optimum (MCNF), far below the shortest path.

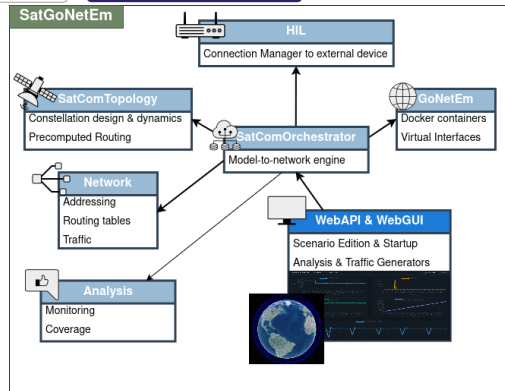


Bottleneck shared by each flow (CDF)

03 / A container-based emulator



- One Docker container per satellite, ground station and user terminal, each with a full TCP/IP stack (Suarez et al. 2026)
- **SatComTopology** computes the time-varying graph; the **orchestrator** (Python, gRPC) turns it into network configuration
- **GoNetEm** backend (Go): containers, virtual Ethernet links, delay and capacity enforced in the kernel
- Routing: pre-computed, static, OSPF (FRRouting), MPLS, and P4



Modular architecture of SatGoNetEm (Suarez et al. 2026)

P4 programs the data plane of a switch:
parser, match-action tables, deparser

(Bosshart et al. 2014).

- **Custom headers:** the protocol of the routing scheme is defined explicitly
- **Software switch:** BMv2 runs inside the satellite containers
- **Control plane:** a controller installs table entries with P4Runtime
- **Path to hardware:** the same program can target programmable switches
- **Standardized definition** of a protocol, in a framework still discussed in top conferences such as SIGCOMM

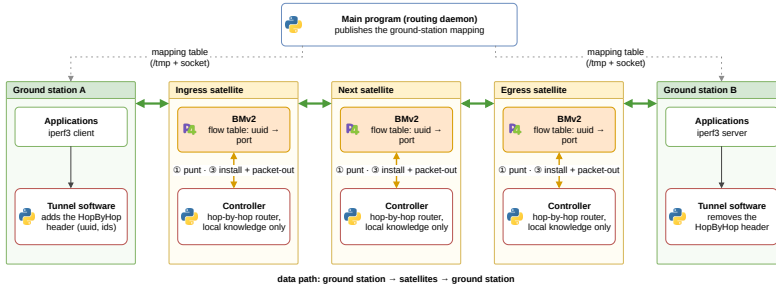
For routing research

The forwarding decision is no longer a simulator function: it becomes a real pipeline, with real headers and real table updates.

03 / Hop-by-Hop in the emulator



Hop-by-hop (p4hb) — Architecture



Routing daemon: starts the controllers, shares the addressing plan and the ground-station mapping

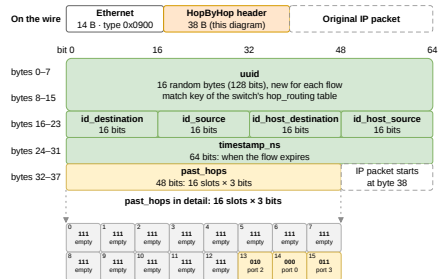
Satellite: BMv2 switch + Python controller (P4Runtime), reacts to ISL and ground-link events

Ground station: tunnel that encapsulates packets in the Hop-by-Hop header and manages the flows

03 / The Hop-by-Hop header



```
header hop_by_hop_t {  
    bit<128> uuid;           // flow id  
    bit<16> id_destination; // dst satellite  
    bit<16> id_source;       // src satellite  
    bit<16> id_host_destination;  
    bit<16> id_host_source;  
    bit<64> timestamp_ns;    // expiration  
    bit<48> past_hops;       // crossed ports  
}
```



03 / Data plane and control plane

```
table hop_routing {
    key = { hdr.hop_by_hop.uuid: exact; }
    actions = { hop_forward; flow_unknown;
                drop_packet; NoAction; }
    size = 65536;
    default_action = flow_unknown();
}

action hop_forward(egressSpec_t port) {
    // came back through its own egress port
    if (port == standard_metadata.ingress_port) {
        flow_looping(); // punt to controller
    } else {
        standard_metadata.egress_spec = port;
    }
}
```

1. First packet: table miss, punted to the controller
2. Controller: Hop-by-Hop decision (distance, then load)
3. Entry uuid → port installed with P4Runtime
4. Next packets forwarded by the switch alone

03 / Hop-by-Hop on a real network stack

Topology design

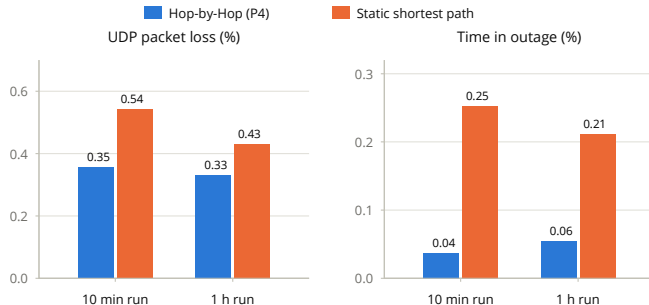
Graph-level evaluation

Emulation

- 308 satellites (14×22), ISLs at 100 Mbps
- 170 capitals $\leftrightarrow$ 27 EU capitals, one link per ground station
- 500 pairs, 300 Mbps (70 % UDP, 30 % TCP)
- Baseline: static IPv4 shortest paths (global view)

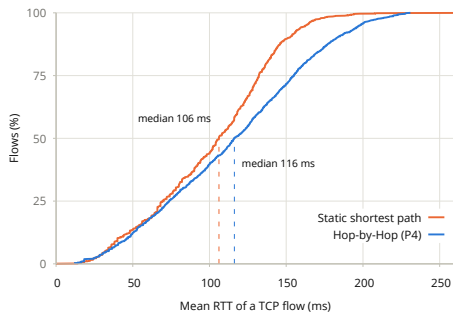
Closing the loop

The algorithm that was only evaluated in simulation now runs on a real data plane.



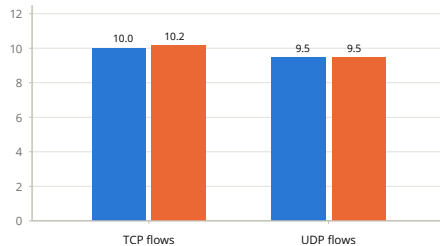
UDP flows: fewer losses and 4 to 7 times less time in outage than the static baseline

03 / Latency and throughput



Mean RTT of each TCP flow (1,384 flows per method)

Hop-by-Hop (P4) Static shortest path
Mean throughput per flow (Mbps)



Every flow is offered 10 Mbps

- **Latency:** +10 ms on the median RTT (minimum hop count, not the shortest distance)
- **Throughput:** identical, the P4 pipeline does not limit the flows

04

Conclusion

- **SatComTopology**: one standalone, fully configurable constellation model for every step of an NTN study
- **SatGoNetEm + P4**: Hop-by-Hop routing implemented on a real network stack and compared with static IPv4 shortest paths

Benefit

It forces realistic implementations: headers, flows, handovers, table updates.

Cost

Significant bootstrap work, but with an emulator and P4 pipelines, most novel solutions can now be tested.

Perspective: going deeper into decentralization. The whole decision moves into the data plane of each satellite, with only the knowledge of its neighbors: no controller, no global view.

-  Bosshart, Pat et al. (2014). "P4: Programming Protocol-Independent Packet Processors". In: *ACM SIGCOMM Computer Communication Review* 44.3, pp. 87–95.
-  Carrizo, Carlos et al. (2020). "Optical inter-satellite link terminals for next generation satellite constellations". In: *Proceedings of SPIE 11272, Free-Space Laser Communications XXXII*, p. 2.
-  Ekici, Eylem, Ian F. Akyildiz, and Michael D. Bender (2001). "A distributed routing algorithm for datagram traffic in LEO satellite networks". In: *TNET*.
-  Evans, Eric (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley.
-  Iglesias-Sanuy, Pablo et al. (2022). "An Efficient Location-Based Forwarding Strategy for Named Data Networking and LEO Satellite Communications". In: *Future Internet* 14.10.
-  Kassing, Simon et al. (2020). "Exploring the Internet from Space with Hypatia". In: *Proceedings of the ACM Internet Measurement Conference (IMC)*. ACM, pp. 214–229.
-  Papapetrou, Evangelos and Fotini-Niovi Pavlidou (2008). "Distributed Load-Aware Routing in LEO Satellite Networks". In: *IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference*.
-  Roth, Manuel M. H. et al. (2025). "Deep Reinforcement Learning for Adaptive Traffic Engineering in Satellite Constellation Networks". In: *2025 12th Advanced Satellite Multimedia Systems Conference and the 18th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, pp. 1–8.
-  Stock, Gregory, Juan A. Fraire, and Holger Hermanns (2022). "Distributed On-Demand Routing for LEO Mega-Constellations: A Starlink Case Study". In: *Advanced Satellite Multimedia Systems Conference / Signal Processing for Space Communications Workshop*.
-  Suarez, Juan et al. (May 2026). "SatGoNetEm: an Open Source Emulator of Satellite Constellations and Non-Terrestrial Networks".



Yan, Hongcheng et al. (2024). "Routing in future space-terrestrial integrated networks with SATNET-OSPF". In: *International Journal of Satellite Communications and Networking* 42.1, pp. 1–25.



Zhang, Xiang et al. (2021). "ASER: Scalable Distributed Routing Protocol for LEO Satellite Networks". In: *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pp. 65–72.



Zhu, Zhenyu et al. (2025). "Intelligent routing methods for low-Earth orbit satellite networks based on machine learning: A comprehensive survey". In: *Ad Hoc Networks* 178, p. 103995.

Thank you for your attention



SatComTopology on GitHub

louis.barbier@isae-superaero.fr

linkedin.com/in/louis-barbier-software-engineer

Excellence with Passion