An open source Multi-Agent Deep Reinforcement Learning Routing Simulator for satellite networks

Federico Lozano-Cuadra, Mathias D. Thorsager, Israel Leyva-Mayorga, and Beatriz Soret *

This paper introduces an open source simulator for packet routing in Low Earth Orbit Satellite Constellations (LSatCs). The simulator, implemented in *Python*, supports traditional Dijkstra's based routing as well as more advanced learning solutions based on Q-Routing and Multi-Agent Deep Reinforcement Learning (MA-DRL) from our previous work. It uses an event-based approach with the *SimPy* module to accurately simulate packet creation, routing and queuing, providing real-time tracking of queues and latency. The simulator is highly configurable, allowing adjustments in routing policies, traffic, ground and space segment topologies, communication parameters, and learning hyperparameters. Key features include the ability to visualize system motion and track packet paths while considering the inherent uncertainties of such a dynamic system. Results highlight significant improvements in end-to-end (E2E) latency using Reinforcement Learning (RL)-based routing policies compared to traditional methods. The source code, the documentation and a *Jupyter notebook* with post-processing results and analysis are available on *GitHub*.

1 Introduction

Efficient routing in LSatCs is critical for global connectivity in 6G networks. This requires addressing multiple challenges, including the partial knowledge of the network at the satellites and their continuous movement, and the time-varying sources of uncertainty in the system, such as traffic, communication links, or communication buffers [1]. Traditional routing algorithms are inadequate to address these problems: They either lack adaptability to network changes or congest the network with feedback messages. To overcome these challenges, new algorithms must be developed, some of them RL-based, which need to be accompanied by a robust framework. *Python* is the best environment for developing RL-based algorithms due to its extensive libraries for machine learning, such as *Keras-TensorFlow*, *PyTorch*, *NumPy*, and *Pandas*. This paper introduces an open source MA-DRL Routing Simulator for satellite networks built in *Python*, where these designed algorithms can be implemented and tested. The simulator supports various routing algorithms, including some Dijkstra's [2] shortest path-based and those from our recent works: (1) The Q-Routing with Q-tables for distributed routing decisions [3], and (2) the MA-DRL first proposed in [4], which was then further tested and extended to continual learning with Satellite Federated Learning (SFL) in [1]. The source code, a *Jupyter notebook* with some post-processing results and analysis, and the documentation for the MA-DRL Routing Simulator are available on *GitHub* [5].

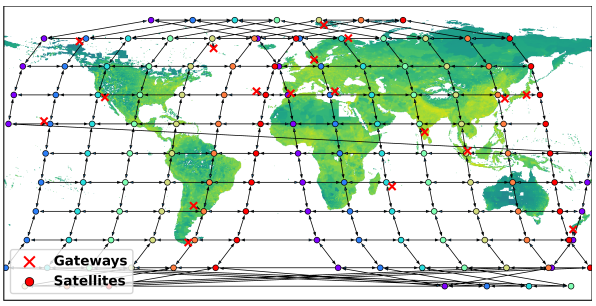

Figure 1: Kepler constellation deployed and their corresponding inter-satellite links (ISLs) established following the *Greedy matching* with 18 active gateways over the population maps [6], where the green tone depends on the population density. Each satellite's colour is a different orbital plane.

2 Simulator architecture

The event-based simulation environment was developed in *Python* using the *SimPy* module, chosen for its effectiveness in discrete event modeling [7]. Time in the simulator progresses by jumping from one scheduled event to the next, rather than continuously. Each action, including creating, routing, and queueing of

*F. Lozano-Cuadra (flozano@ic.uma.es) and B. Soret are with the Telecommunications Research Institute, University of Malaga, 29071, Malaga, Spain. M. D. Thorsager and I. Leyva-Mayorga are with the Connectivity Section, Aalborg University, 9220, Aalborg, Denmark. The work of F. Lozano-Cuadra and B. Soret is partially funded by the Spanish Ministerio de Ciencia, Innovación y Universidades ("TATOOINE", PID2022-136269OB-I00) and by ESA SatNEx V (prime contract no. 4000130962/20/NL/NL/FE). The view expressed herein can in no way be taken to reflect the official opinion of the European Space Agency (ESA).

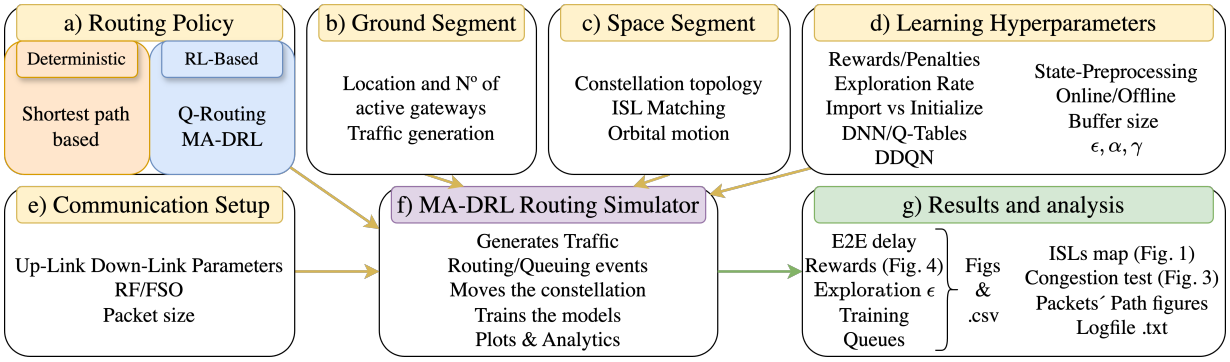

Figure 2: Input-Output MA-DRL Routing Simulator workflow.

individual data packets, is explicitly simulated as a *SimPy* event, providing accurate real-time tracking of queues and latency. Packets are unique entities (objects) existing from creation at a generating gateway until they arrive at the destination gateway. The transmission time is calculated based on the packet size and current link rates, propagation time based on the exact distance between transmitter and receiver at transmission time, and queue time based on the time a packet spends in the queue. This detailed level of simulation is essential for representing the states and computing the rewards of the environment in our RL-based routing algorithms.

The simulator emulates a realistic scenario where ground gateways gather the nearby terrestrial traffic that is assumed to be generated by mobile users and distribute that to each other gateway equally through a LSatC, integrating space and ground segments into the communication network, as shown in Fig. 1. The environment is built as a time-variant dynamic graph $\mathcal{G}_t(\mathcal{N},\mathcal{E})$ with nodes $\mathcal{N}$, representing satellites and gateways, and edges $\mathcal{E}$, representing the transmission links between them, which can be either ISL or ground-to-satellite link (GSL), implemented as Radio Frequency (RF) or Free Space Optical (FSO).

Space segment. The satellite constellation consists of N satellites evenly distributed across O orbital planes. Each satellite functions as a router and learning agent for RL-based solutions. Satellites are positioned at specific and configurable altitudes, longitudes, and orbit inclinations, moving according to orbital mechanics and Earth's rotation [1]. Satellites move periodically, at the beginning of each time interval, rather than continuously. After a fixed time interval, each satellite is placed in the exact position that it would reach if it had moved continuously during that period. This periodic movement impacts latency calculations by updating transmission and propagation times at each position update. Each satellite has one antenna for GSL and four for ISL (two for inter-plane and two for intra-plane links). Selecting the best ISL is a dynamic matching problem and consists of establishing the best ISLs among satellites. Links are bidirectional, and the network is reconfigured as satellites move, i.e, $\mathcal{G}_t$ is built again maintaining previous queue states.

Ground segment. The ground segment consists of a set of configurable ground gateways, which gather the terrestrial traffic from mobile devices. Each gateway aggregates this traffic into large packets for transmission to its nearest satellite, with which it maintains a GSL.

Data rate. The communication data rate between nodes i and j, $R(i,j)$, is determined by the highest modulation and coding scheme that ensures reliable communication based on the current signal-to-noise ratio (SNR), and zero otherwise, using DVB-S2 technology [8] for realistic data rates assuming free-space loss [1].

Traffic generation. We consider a scenario with realistic packet generation, queuing, and transmission, where each gateway transmits data equally split among the other gateways through the LSatC, then data is assumed to be distributed to the nearby connected users. The total traffic load ℓ in the network is determined by the uplink data generation rate at each gateway and the maximum supported traffic load ℓ, derived from uplink and downlink rates. The traffic generation follows a Poisson distribution and ℓ is configurable by the user. As each gateway sends traffic to each other, the total number of unidirectional flows U_f can be expressed as: $U_f = n_g \cdot (n_g - 1)$, where n_g is the number of active gateways.

Routing. The routing algorithm at each satellite i aims to relay each received packet $p(d)$ towards its destination d. Each satellite has a transmission buffer with a maximum capacity of $Q^{\max}$, operating under a first-in first-out (FIFO) strategy. If the buffer is not empty, the satellite takes the Head of Line packet and delivers it to one of its linked nodes following the chosen routing policy. Any packet arriving at a full buffer

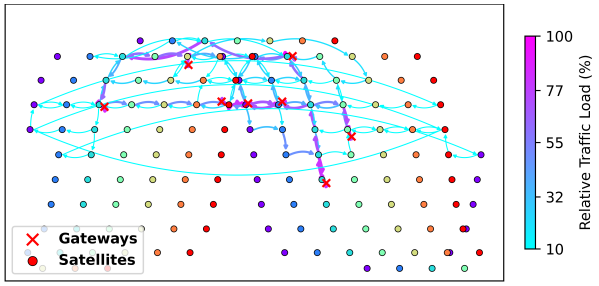

Figure 3: MA-DRL's exploitation phase congestion test for all routes output with 8 active gateways.

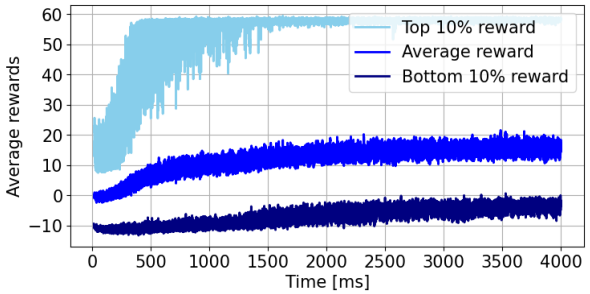

Figure 4: Rewards over time of the offline phase of MA-DRL with 8 active gateways. The highest rewards are given after a packet has been delivered to the receiving gateway.

is dropped.

Latency. The one-hop latency to transmit a packet from i to j depends on three factors: queue time, transmission time, and propagation time [1]. The queue time at the transmission queue is the elapsed time since the packet is ready to be transmitted until the beginning of its transmission. The transmission time is the time taken to transmit the packet based on the transmission rate. The propagation time is the time it takes for the signal to travel the distance between i and j, $\|ij\|$. This latency model considers varying traffic loads, where propagation time is significant in low traffic but queue time increases under high traffic conditions [9].

3 Routing algorithms

Different routing policies are implemented in the simulator. On one hand, we have the deterministic ones, all of them based on shortest path Dijkstra's algorithm [2], where the edge weights are minimized in centrally with full knowledge of the constellation. Each method minimizes a different weight: (1) **Data Rate**, where the edge weights between two nodes i and j are determined by the inverse of the data rate between nodes, namely $w_{i,j} = 1/R(i,j)$. This is a traditional routing approach that leads to choosing routes with high data rate links; (2) **Slant Range**, where the edge weights between i and j nodes are defined by the distance between them, $\|ij\|$, in order to minimize propagation times, and the (3) **Hop**, where all edges have the same weight, 1, where the total number of jumps is minimized.

On the other hand, other two RL-based routing policies are implemented, specifically the ones developed in our previous work. Firstly, the **Q-Routing** policy, developed in [3]. Q-Tables are created automatically with NumPy [10] to increase efficiency. They will store the learnt knowledge during the training process. The user can choose if it wants the algorithm to explore and make random routing actions or import pre-trained Q-Tables and exploit its knowledge to use them as routing policy.

Secondly, **MA-DRL** from [1, 4] is implemented. The Deep Neural Networks (DNNs) are initialized and trained with *Keras* [11]. Double Deep Q-Learning (DDQN) [12] is implemented and its usage is configurable. It is also possible either to import pre-trained DNNs or not and choose between the offline and the online phase of the algorithm.

4 Setup and general settings

The simulator, running in **Python 3.9**, is multiplatform and has been tested on Windows, Linux, and Mac systems. The user can install the required packages listed in the *requirements.txt* file using pip. It is advisable to create a virtual environment or an Anaconda environment for better management.

The simulator is highly configurable, allowing users to adjust various parameters to suit their specific needs, as illustrated in Fig. 2. Key configurable parameters include: (a) **Routing Policy**, including the shortest path-based, where the Data Rate, Slant Range or Hop can be set as weights, and the RL-based options; (b) **Ground Segment** settings, such as the number and locations of active gateways, as well as traffic generation ℓ; (c) **Space Segment** parameters, which cover constellation design (configurable elements include the number of orbital planes and its inclination angle, satellites per plane, and the choice between Walker delta and Walker star designs), ISL matching (Greedy or Markovian [13]), and orbital motion (which can be sped up or slowed down); (d) **Learning Hyperparameters**, including rewards and penalties, exploration ϵ, learning α and gamma γ rates, state preprocessing, and training modes (Import pre-trained models for RL-based policies and choosing between online and offline phases for MA-DRL); and (e) **Communication Setup**, such as physical constants, uplink and downlink parameters, and packet size. Additionally, the user can configure the simulator to plot the environment every time the constellation moves to visu-

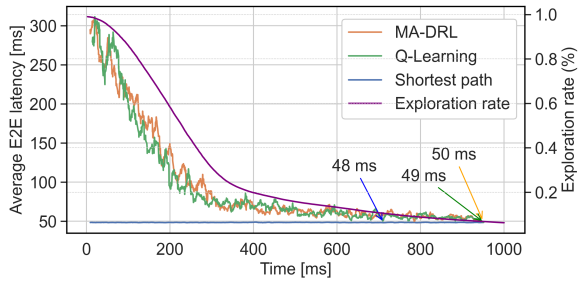

Figure 5: E2E latency vs time vs ϵ connecting one gateway in Malaga, Spain and another one in Los Angeles, USA, through the Starlink constellation during the offline phase of the RL-based methods. It can be appreciated how both methods learn to find the optimal path in less than 1 second.

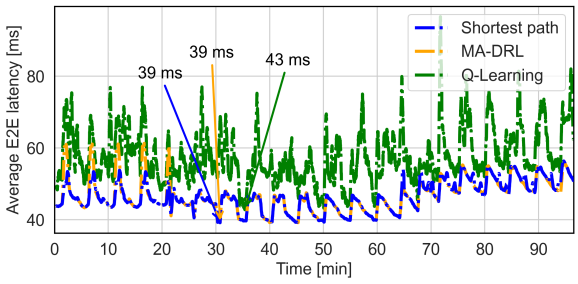

Figure 6: Average E2E latency over an orbital period. The fluctuations are given by the movement of the satellites and the resulting changes in the routed followed by the packets.

alize system motion, as in Fig. 1, and to plot the path of each delivered packet over this to track its journey through the network.

With all these settings configured, the simulator is now ready to run simulations and generate results.

5 Results and analysis

The default ground segment has up to 18 active (transmitting and receiving) gateways distributed across the Earth, mainly following KSAT's deployment[1], but more gateways can be added easily.

Moreover, four real constellations are implemented: (1) *Kepler* constellation design, with $O = 7$ orbital planes at heights $h = 600$ km and $N = 20$ satellites per orbital plane, as illustrated in Fig. 1; (2) *Iridium Next* constellation, with $O = 6$, $h = 780$ km and $N = 11$; (3) the *OneWeb* constellation, with $O = 36$, $h = 1200$ km and $N = 18$; and (4) *Starlink* orbital shell at $h = 550$ km, with $O = 72$ and $N = 22$. The three first constellations follow a *Walker star* architecture, while the Starlink shell follows a *Walker delta* architecture [14]. Moreover, two additional artificial constellations are implemented for testing. Additionally, two ISL matching algorithms are implemented: (1) The Markovian solution proposed in [13] and (2) a *Greedy* approach, where each satellite connects with immediate neighbors within its plane and closest counterparts in adjacent planes in both East and West directions, optimizing for latency and data rates, as shown in Fig. 1.

When a simulation ends, it automatically outputs a set of results in the form of figures and text files (Fig. 2). Within the **figures**, a map like Fig. 1 with the system model information is saved. An update of this figure is also saved as the constellation moves if desired. Then, a congestion test per route and for all routes between gateways is done, in order to see what nodes and edges did the packets went through, as shown in Fig. 3. If one of the RL-based routing policies is chosen, one figure with the exploration rate ϵ and training stamps and another one with the received rewards (Fig. 4) are also saved. Other figures saved are related to the average E2E latency vs time vs ϵ, similar to Fig. 5, but with just one routing policy, and to the queue lengths. On the other hand, the output **files** include several *.csv* with extensive information about each packet's path and its latency, rewards, exploration rates, training stamps, hyper-parameters and a *.txt* log-file, that saves everything that happened during the simulation and gives some statistics like like the latency broken down by average queue, transmission and propagation times, packets delivered vs stuck and/or lost, most used links, etc. Lastly, if either the MA-DRL or the Q-Routing algorithm was chosen for routing, the trained DNNs (57Kb for the Q-Network and 27 Kb for the Q-Target) or Q-Tables (21Kb for 8 active gateways) are saved, respectively.

In the *Jupyter notebook*, we conduct further **post-processing** analysis and explore more complex results. A comparison between the *Shortest path* routing policy, Q-Routing [3] and MA-DRL [1, 4] at their offline phase is shown in Fig. 5. Additionally, a dynamic comparison of these policies at their online phase is shown in Fig. 6, where the constellation has moved to complete one orbital period in 96 minutes, with the satellite positions being updated at intervals of 15 seconds. Notably, even with only partial knowledge of the constellation, MA-DRL consistently maintains the baseline latency obtained with the *Shortest path* policy, which has full knowledge of the constellation. Moreover, we elaborate on the comparison of the four architectures in Fig. 7, where the distribution of the E2E latency is depicted in a box plot when the *Shortest path* is applied among one orbital period too. We observe that *Kepler* and *Starlink* obtain the smallest average latency, although the latter presents more outliers. This figure helps to illustrate the behavior of the constellations and highlights the usage of the simulator to test differ-

[1] https://www.ksat.no/services/ground-station-services/

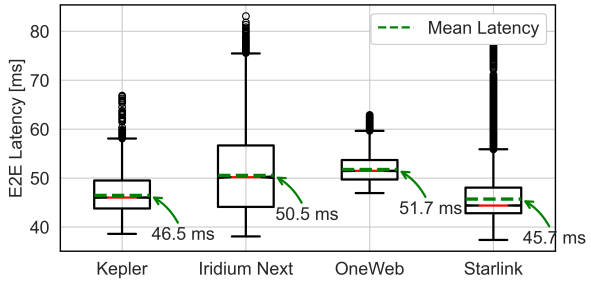

Figure 7: Box plot of the E2E latency of the four constellation topologies with the *Shortest Path* policy after one orbital period is completed.

ent constellation architectures.

Additionally, as in MA-DRL, each satellite is an independent agent during the online phase, we conducted a Centered Kernel Alignment (CKA) [15] analysis to compare the differences between each agent's local model after 1 second with varying traffic patterns around the globe. Each satellite learns and adapts its routing decisions based on these traffic patterns, resulting in distinct updates to their local models. Consequently, these models exhibit differences. To homogenize the models, we applied post-processing SFL techniques: Initially among neighboring satellites, Model Anticipation; then, among orbital planes, Orbital Plane Aggregation (SFL); and finally, across the entire constellation, Full Aggregation (SFL) [1].

6 Conclusion and Future work

The development of an open source MA-DRL simulator for satellite network routing provides a robust platform for testing and implementing various routing algorithms in *Python*, where different machine learning libraries can be leveraged. The simulator's high configurability and realism allows for comprehensive evaluation of different constellation designs and communication setups. The results highlight the effectiveness of RL-based routing policies compared to traditional methods, demonstrating significant improvements in E2E latency and overall network performance.

Future directions include: (1) Developing an SFL framework to enable aggregation during the online phase of MA-DRL rather than implementing it as a post-processing analysis; (2) implementing a two-tier mesh network for UE-satellite-UE communications, enabling ground moving users to connect directly to satellites without the need for gateways; (3) increase the complexity of satellites with regenerative capabilities; and (4) implementing different types of traffic with splittable flows.

References

1. Lozano-Cuadra, F., Soret, B., Leyva-Mayorga, I. & Popovski, P. Continual Deep Reinforcement Learning for Decentralized Satellite Routing. *arXiv preprint arXiv:2405.12308* (2024).
2. Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische mathematik* **1,** 269–271 (1959).
3. Soret, B., Leyva-Mayorga, I., Lozano-Cuadra, F. & Thorsager, M. D. *Q-learning for distributed routing in LEO satellite constellations* in *Proc. IEEE ICMLCN 2024, arXiv preprint arXiv:2306.01346* (2023).
4. Lozano-Cuadra, F. & Soret, B. *Multi-Agent Deep Reinforcement Learning for Distributed Satellite Routing* in *Proc. IEEE ICMLCN 2024, arXiv preprint arXiv:2402.17666* (2024).
5. Lozano-Cuadra, F., Thorsager, M. D., Leyva-Mayorga, I. & Soret, B. *MA-DRL Routing Simulator* `https://github.com/SatCom-TELMA/MA-DRL_Routing_Simulator`. 2024.
6. Center for International Earth Science Information Network - CIESIN - Columbia U. *Gridded Population of the World, Version 4 (GPWv4)* `https://sedac.ciesin.columbia.edu/data/collection/gpw-v4/sets/browse`.
7. Zinoviev, D. Discrete Event Simulation: It's Easy with SimPy! *arXiv preprint arXiv:2405.01562* (2024).
8. *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications (DVB-S2)* Standard (ETSI, France, Oct. 2014).
9. Rabjerg, J. W., Leyva-Mayorga, I., Soret, B. & Popovski, P. *Exploiting topology awareness for routing in LEO satellite constellations* in *Proc. IEEE GLOBECOM* (2021).
10. Van Der Walt, S., Colbert, S. C. & Varoquaux, G. The NumPy array: a structure for efficient numerical computation. *Computing in science & engineering* **13,** 22–30 (2011).
11. Ketkar, N. & Ketkar, N. Introduction to keras. *Deep learning with python: a hands-on introduction*, 97–111 (2017).
12. Van Hasselt, H., Guez, A. & Silver, D. *Deep reinforcement learning with double q-learning* in *Proceedings of the AAAI conference on artificial intelligence* **30** (2016).
13. Leyva-Mayorga, I., Soret, B. & Popovski, P. Inter-Plane Inter-Satellite Connectivity in Dense LEO Constellations. *IEEE Trans. on Wireless Comms.* **20,** 3430–3443. ISSN: 1536-1276 (6 June 2021).
14. Leyva-Mayorga, I. *et al.* NGSO constellation design for global connectivity. *arXiv preprint arXiv:2203.16597* (2022).
15. Kornblith, S., Norouzi, M., Lee, H. & Hinton, G. *Similarity of neural network representations revisited* in *International conference on machine learning* (2019), 3519–3529.