Stardust: A Scalable and Extensible Simulator for the 3D Continuum

Thomas Pusztai
Distributed Systems Group, TU Wien
t.pusztai@dsg.tuwien.ac.at

Jan Hisberger
Distributed Systems Group, TU Wien
e12126323@student.tuwien.ac.at

Cynthia Marcelino
Distributed Systems Group, TU Wien
c.marcelino@dsg.tuwien.ac.at

Stefan Nastic
Distributed Systems Group, TU Wien
snastic@dsg.tuwien.ac.at

***Abstract*—Low Earth Orbit (LEO) satellite constellations are quickly being recognized as an upcoming extension of the Edge-Cloud Continuum into a 3D Continuum. Low-latency connectivity around the Earth and increasing computational power with every new satellite generation lead to a vision of workflows being seamlessly executed across Edge, Cloud, and space nodes. High launch costs for new satellites and the need to experiment with large constellations mandate the use of simulators for validating new orchestration algorithms. Unfortunately, existing simulators only allow for relatively small constellations to be simulated without scaling to a large number of host machines. In this paper, we present Stardust, a scalable and extensible simulator for the 3D Continuum. Stardust supports i) simulating mega constellations with 3x the size of the currently largest LEO mega constellation on a single machine, ii) experimentation with custom network routing protocols through its dynamic routing mechanism, and iii) rapid testing of orchestration algorithms or software by integrating them into the simulation as SimPlugins. We evaluate Stardust in multiple simulations to show that it is more scalable than the state-of-the-art and that it can simulate a mega constellation with up to 20.6k satellites on a single machine.**

***Index Terms*—3D continuum, edge-cloud continuum, orbital edge computing, LEO satellites, simulator**

I. Introduction

Low Earth Orbit (LEO) satellites are rapidly increasing in number in recent years. As of 2024, there are more than 8,000 LEO satellites in orbit [1], with about 7,000 of these belonging to the Starlink mega constellation [2], which plans to grow to more than 12,000 satellites by 2028 [3]. Amazon intends to have its Kuiper mega constellation with more than 3,236 satellites [4] complete by 2029 and the FCC has called for more competition in this sector [2].

LEO mega constellations provide low latency communication between LEO and terrestrial nodes and among terrestrial nodes. For example, Starlink's median client-LEO-Cloud round-trip latency has recently been measured to be 40-50 ms [5]. The low altitude of LEO satellites compared to geostationary satellites allows for low latency with terrestrial nodes that are directly in range. Inter-satellite laser links (ISLs) enable the creation of large orbital networks [6], with ISL speeds demonstrated up to 100 Gbps [7].

Since LEO satellites get more computational capabilities with every new generation, several uses beyond bent pipe communication are being investigated. Processing of Earth observation (EO) data on a single satellite has been shown by ESA [8] and plans for processing data in clusters of satellites have been proposed under various names, such as Orbital Edge Computing (OEC) [9], [10], satellite computing [11], [12], or Edge-Cloud-Space 3D Continuum [13], [14]. Possible use cases include federated learning in space [15]–[18], EO data compression for efficient downlinking [19], smart agriculture [20], and disaster response [13]. The actual use of satellites varies between the ideas, from preprocessing EO data to full-fledged compute nodes that enable seamless execution of workloads across the 3D Continuum. All proposals have in common that they require ways to evaluate their designs.

Since launching new satellites is expensive and the computing capabilities anticipated for the near future are not available in space yet, evaluation of LEO computing systems must be performed using simulators or emulators. Various simulators already exist, e.g., Hypatia [21], Celestial [22], and StarryNet [23]. While all of them compute satellite trajectories and node-to-node latencies, most simulators have a number of shortcomings. Many solutions are emulators, e.g., Celestial [22] and StarryNet [23], which execute microVMs or containers for each node of the 3D Continuum. This has the advantage of enabling tests of real software systems, but executing one microVM or container for every node, regardless of whether it is used or not, limits the maximum infrastructure size that can be evaluated due to the resource usage. Celestial allows suspending all VMs that are not in a certain area, but it still creates a microVM for each node. Conversely, simulators, such as Hypatia [21], do not execute nodes and are used for network simulation only, hence software testing or evaluation of placement algorithms that require node resource information are not possible. Many simulators and emulators focus on the network latencies, precompute them before the experiment, and disregard the positions of satellites during the experiment. Hence, it is often impossible to determine where a particular node is located during the experiment, although this information may be needed, e.g., for location-aware scheduling or for picking a satellite that will be in range of a certain ground station when it completes the next workload. Many solutions account only for LEO and ground station or Cloud nodes, but not for terrestrial Edge nodes, such as drones. Typically, each simulator or emulator is designed for a single purpose only, e.g., testing software under resource and network constraints or evaluating network routing algorithms. Existing solutions often lack extensibility, such as allowing custom logic to execute after every simulation step, which could, e.g., be used to add a new deployment to the experiment.

In this paper, we present Stardust, a scalable and extensible open-source[1] simulator for the 3D Continuum. Our main contributions are:

1) *Stardust, a scalable and extensible next generation simulator* for the 3D Continuum with support for simulating LEO-, Cloud-, and Edge nodes in a scalable manner. Stardust enables experiments for evaluating networking and orchestration algorithms for the 3D Continuum. It supports simulating mega constellations three times the size of the currently largest constellation, with almost 7k satellites on a single machine.
2) A *dynamic routing mechanism* that enables experimentation with different routing mechanisms by making the ISL routing protocol and the network path computation changeable. This allows, e.g., changing the default +Grid ISL routing to a different protocol or to introduce caching or hypergraph algorithms as a replacement for Dijkstra's algorithm to calculate node-to-node network paths.
3) *SimPlugin, a plugin mechanism* that serves as the integration point for custom logic that Stardust should execute at every step of the simulation. A SimPlugin has access to the complete infrastructure state and, thus, allows integrating, e.g., orchestration algorithms/software that should be evaluated using Stardust.

The rest of this paper is structured as follows: Section II presents a motivating use case and requirements for a next-generation simulator for the 3D Continuum. Section III explores other 3D Continuum simulators and emulators and Section IV presents the design of the Stardust simulator. In Section V we evaluate Stardust in multiple simulations and in Section VI we conclude the paper and present future work.

II. Motivating Use Case & Simulator Requirements

In this section, we first present a motivating disaster response scenario for the use of the 3D Continuum and, subsequently, we define requirements for an extensible next-generation simulator for the 3D Continuum.

A. Motivating Scenario

While there are various use cases for the 3D Continuum, some of the most compelling ones involve running distributed AI using a combination of observation data from EO satellites and in-situ data from terrestrial sensors in a compound AI scenario [24]. EO data is large in size, e.g, each of the ESA Sentinel 2 satellites produces about 1.5 TB of data per day [25], [26], while downlink speeds to ground stations are typically approximately 300 Mbps [27]. Thus, especially in cases where EO data must be processed quickly and possibly be augmented with data from in-situ sensors, it is beneficial to preprocess it in a cluster of LEO satellites to gain insights more quickly.

[1]https://github.com/polaris-slo-cloud/stardust and https://doi.org/10.5281/zenodo.15484629

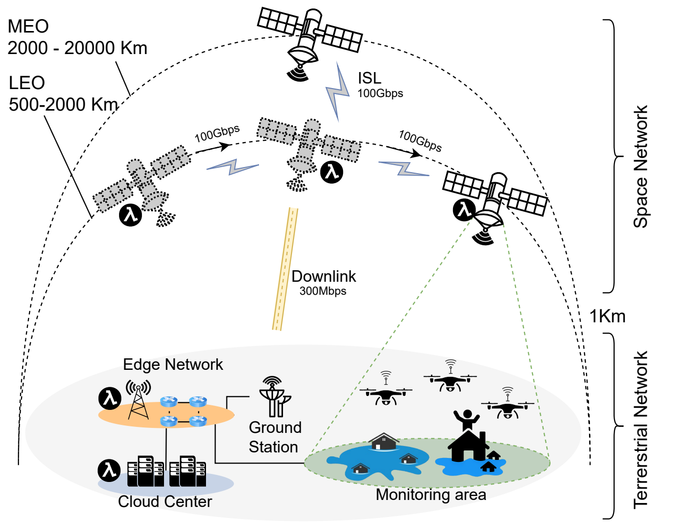

Fig. 1: Motivating Use Case: Flood Disaster Response with Edge- and LEO-based Serverless Computing.

Responding to natural disasters requires quick response times. For example, if a suburban area is flooded after a hurricane, it is important to quickly identify people or animals that are in need of rescue. Fig. 1 shows a use case, where a combination of EO satellites and drones is used to run a serverless workflow to find people and animals in need of help after a hurricane. After the storm, drones fly over the affected area and record video data. The drones are not powerful enough to run the ML model needed to detect people, so they need to offload this computation. However, the cellular network has been damaged, so the video feed must be uplinked to LEO satellites. These LEO satellites also receive data from an EO satellite and combine that with the video from the drones to detect probable locations of survivors. The identified locations are downlinked to a Cloud for detailed analysis and, if the presence of survivors is confirmed, are forwarded to rescue teams.

Each of the serverless functions in this workflow has resource and latency requirements and must be placed on an appropriate node for execution. This is done by a scheduler for the 3D Continuum. Such schedulers are a hot research topic at the moment, and their evaluation requires a simulator that covers all node types of the 3D Continuum and provides node positions and network latencies. Stardust allows such schedulers to be evaluated efficiently because its extensibility enables quick integration of the scheduling algorithms into the simulator.

But Stardust is not limited to the evaluation of scheduling algorithms. It can also be used for evaluating resource man-

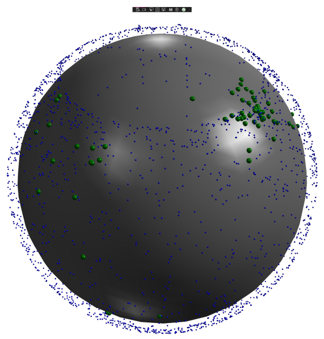

Fig. 2: Visualization of the Simulated 3D Continuum with Earth (gray), Satellites (blue), and Ground Stations (green). Ground Stations are Located in Selected Major Cities of Europe (right) and America (left).

agement, network routing, or other orchestration algorithms. Additionally, Stardust will support the execution of workloads in the future, so the entire use case will be executable on the simulator. Fig. 2 shows a visualization of the 3D Continuum simulated by Stardust with ground stations indicated in green and satellites shown in blue.

B. Requirements

Based on the advantages and disadvantages of existing simulators/emulators, we define the following requirements for a next-generation simulator for the 3D Continuum:

R1 *Simulate entire 3D Continuum*: The simulator must support simulating Edge, Cloud, and LEO satellite nodes to allow evaluation of algorithms for the entire continuum.

R2 *Configurable simulation steps*: The amount of simulated time that elapses in a single simulation step must be configurable, because some scenarios may require very fine-grained simulation steps (e.g., real-time use cases), while other scenarios need coarse-grained simulation steps for simulations that span multiple hours.

R3 *Extensibility*: The simulator must be easily extendable with custom logic to be executed at every simulation step. This enables the fast implementation of experiments for new algorithms.

R4 *Information availability*: Custom logic must have access to all relevant data, such as node positions, node resources, and network routes.

R5 *Choice between simulation and emulation*: Users must have the choice whether the experiments should execute as a simulation that tracks node positions, resources, and network state, and that executes custom logic or if selected nodes should be emulated to allow execution of workloads in containers or VMs. In emulation mode, only nodes that host a workload must execute a container or VM to keep resource usage to a minimum.

As we will explain in Section IV, Stardust focuses on and fulfills R1-R4. The complexity of R5 merits a distinct in-depth evaluation. Hence, we defer it to future work.

III. Related Work

In this section, we discuss existing serverless platforms that explore workloads in the Edge-Cloud-Space 3D Continuum and LEO simulators that enable the emulation and simulation of workflows in LEO edge constellations.

A. LEO Edge Simulators

Existing simulators for the 3D Continuum can be divided into two main categories: network-only simulators and emulators.

Network-only simulators focus on simulating the network of a satellite mega constellation and their connections to ground stations, but they typically fail to account for the computational capabilities of LEO satellites as nodes that can execute workloads. The authors of [28] simulate satellite mega constellations with ISLs and integrate them with ground stations, however, focusing purely on the network and not on workloads. Hypatia [21] is a framework for simulating and visualizing the network behavior of LEO satellite constellations. It incorporates satellite-specific characteristics such as high-velocity orbital motion, ISL, and ground-satellite links, enabling the evaluation of transport protocols, such as TCP and UDP, in a LEO-specific environment. Xeoverse [29] is a scalable and high-fidelity real-time simulation platform designed specifically for LEO satellite mega-constellations. It models user terminals, satellites, and ground stations as lightweight VMs, pre-computing topology changes and focusing on relevant ISL updates while streamlining link adjustments as needed. Xeoverse provides detailed network characteristics, including latency, capacity, signal-to-noise ratio (SNR), weather conditions, and antenna configurations. The popular and extensible network simulators ns-3 [30] and OMNeT++ [31] have also been used as bases for satellite simulators. For example, SNS3 [32] and ns-3-leo [33] are built on top of ns-3, while OS^3 [34] and its successor by Valentine and Parisis [35] are based on OMNeT++. All four enable the simulation of satellite and ground station networks with control over details such as network protocols, packets, and radio frequencies.

Emulators simulate the network of a satellite constellation and provision containers or VMs for the nodes of the 3D Continuum to allow executing workloads on them. However, emulators often suffer from limited scalability because the containers/VMs consume too many resources as the satellite constellation grows. Celestial [22] is a virtual testbed that emulates LEO Edge satellite networks using microVMs. It precomputes satellite trajectories, bandwidth, and latencies between nodes at different points in time, allowing the orchestrator to manage network configuration requirements, such as SLOs, and to dynamically control microVMs based on the positions of the satellites. However, Celestial lacks real-time satellite orbit positioning, as it relies on pre-calculated latencies. Even though Celestial allows suspending microVMs, whose nodes are currently outside of a bounding box, e.g., the space above Europe, its approach is still resource-intensive for large satellite constellations, limiting the number of nodes that can be simulated on a single machine. StarryNet [23] integrates real constellation data, including satellite trajectories, ground station distributions, and ISL configurations, reproducing the spatial and temporal dynamics of mega-constellations while allowing researchers to deploy unmodified system code and simulate interactive network traffic. StarryNet ensures that its experiments reflect the scale and behavior of real-world LEO networks, including time-varying connectivity and delays. StarryNet simulates constellations comprising thousands of satellites using a distributed, containerized setup across multiple machines.

Although these simulators and emulators offer space-ground integration, they have various shortcomings. Simulators typically focus on LEO-specific network simulation and do not account for the ability to execute workloads. They also often implement many low-level networking details, which slow down large-scale simulations. For example, the authors of the OS^3 derivative [35] report that a 5-minute simulation of 1,400 satellites with a step granularity of one second took more than four hours. Emulators focus on actual workload execution (not simulation), which makes them very resource-intensive, thus limiting their scalability for large satellite constellations, as we show in our experiments in Section V-C.

B. *Evaluation Methods for LEO Platforms*

HyperDrive [13] proposes a serverless platform that integrates devices across the Edge, Cloud, and space, creating a seamless continuum. HyperDrive enables serverless workflows to be executed across any layer within the 3D Continuum. The scheduling mechanisms in HyperDrive consider processing capacities, such as CPU and memory, as well as specific properties of each layer, including satellite temperature and the battery levels of edge devices during the function placement process. However, HyperDrive relies on StarryNet's [23] network simulation to determine node positions, which does not include Edge devices.

Komet [36] introduces a serverless platform tailored for LEO Edge computing, seamlessly integrating serverless functions with data replication to enable dynamic serverless function execution against satellite trajectories. By decoupling compute and state, Komet ensures virtual stationarity, allowing functions to maintain proximity to data despite the orbital movement of satellites. However, Komet's reliance on Celestial [22] for network emulation focuses solely on satellite and ground station interactions, omitting broader integration with terrestrial Edge and Cloud nodes.

IV. Stardust Simulator Design

We now explain the architecture and core mechanisms of Stardust, which enable scalable simulations of the 3D Continuum.

A. *Stardust Architecture*

Stardust is designed to use a modular architecture to enable extensibility. The core abstractions are shown in Fig. 3. The central type from which all 3D Continuum nodes derive is `Node`.

Every `Node` has computational capabilities, communication links to other nodes, and the ability to route traffic. The abstract `Node` class is currently implemented by the

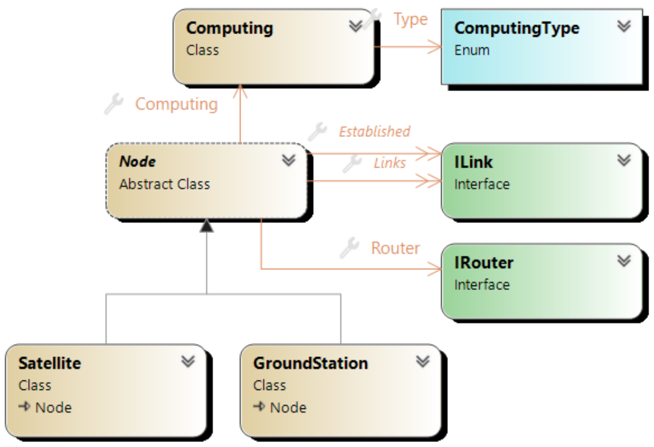

Fig. 3: Stardust Simulator Core Abstractions.

`GroundStation` and `Satellite` subclasses, required for different behaviors in movement. For `GroundStation`, a simplified movement (without inclined Earth axis and exact rotation duration) along the latitude every 24 hours is implemented. `Satellite` calculates the position by solving Kepler's equation for eccentric anomaly, considering only the gravitational interaction between the Earth and the satellite, i.e., it computes an unperturbed orbit. Since we focus on simulations of a few hours, this simplification helps reduce the computational complexity and hardware requirements, while providing sufficient accuracy. Incorporation of perturbations, such as J2 and atmospheric drag [37], is planned as future work. The `Computing` class is responsible for tracking the available and used compute resources (e.g., CPU and memory) of each node. Each `Computing` can be configured individually, such that simulated hardware can be set up to reflect specific conditions, e.g., no general-purpose compute hardware on older satellites, but GPU resources on the newest satellites. This can simulate the fact that the hardware of satellites in orbit cannot be repaired, replaced, or upgraded as easily as data center hardware on Earth. To ensure realistic simulation of resource availability, the allocation of resources to tasks is handled as in Kubernetes, i.e., a task is assigned exclusive ownership of the requested resources for its entire execution duration. Each computing has a `ComputingType` which tags a Computing and further a Node and can potentially be used in routing, scheduling, or other algorithms. To create an Edge Node, an instance of `Node`, i.e., either `Satellite` or `GroundStation`, gets assigned an instance of `Computing` tagged with `ComputingType.Edge`.

To simulate the available network connections with 100% accuracy with respect to the node positions in the 3D Continuum, Stardust constructs a network graph that captures all nodes and the physical connections between them. A direct physical connection between two nodes, e.g., a cable, radio, or laser link, is modeled by a link (`ILink`) between those two nodes in the network graph. Depending on the type of connection and other factors, such as distance, each link has particular latency and bandwidth properties. Network routing relies on these links and their properties. `IRouter` is the abstraction for our dynamic routing mechanism, which may use either pre-route calculations, e.g., for algorithms that construct a routing table each step, or on-route calculations, for direct node-to-node routing like A*.

To allow exploring the behavior of the 3D Continuum over multiple hours within a reasonable timeframe, the speed at which time progresses in the simulation is configurable. Within the simulation, time passes in discrete steps, called *simulation steps*. A simulation step can be configured to cover an arbitrary amount of time in the simulation, depending on the required granularity, e.g., one second, one minute, or five minutes.

Algorithm 1 Simulation Step Progression.

```
 1: Input: $t$: datetime; $N$: nodes; $P$: plugins
 2: for $n \in N$ do
 3:     n.CalculatePosition(t)
 4: end for
 5: for $n \in N$ do
 6:     n.UpdateLinks()
 7: end for
                  ▷ Optional/only for protocols with routing tables
 8: for $n \in N$ do
 9:     n.CalculateRoutingTable()
10: end for
                                    ▷ Run the plugins on step end
11: for $p \in P$ do
12:     p.PostSimulationStep()
13: end for
```

At every simulation step, the state of the simulation is refreshed, i.e., the node positions and the network graph are updated. Algorithm 1 shows the high-level progression of a simulation step.

1) Positions are calculated within the subclasses of `Node`. Current implementations of `Satellite` and `Ground-Station` handle their distinct movement: orbiting Earth and points on Earth rotating around its axis. The resulting positions are Earth-centric coordinates to get unified positions.
2) After all unified node positions are calculated, the physical links between nodes are established to update the network graph.
3) Using established links, a routing protocol can either calculate routing tables (e.g., using Dijkstra's algorithm) for constant route lookup times or, alternatively, routes can be calculated on-demand (e.g., a protocol using the A* algorithm).

Steps 2 and 3 are central to the network simulation, as we will discuss in the next subsection.

B. Dynamic Link Protocols and Routing Mechanism

Simulating network communication in the 3D Continuum consists of two steps, which can be realized in various ways. To enable experimentation with different algorithms, Stardust relies on a dedicated abstraction to encapsulate the algorithm of each step:

1) A *link protocol* determines which pairs of nodes have a direct physical connection, such as a cable-, radio-, or laser link (i.e., physical and data-link layers of the OSI model). These are links in the network graph, each with a bandwidth and latency.
2) A *routing protocol* finds a route through the network graph for two nodes that want to communicate with each other (i.e., routing on the network layer of the OSI model). These are simple paths through the network graph, with bandwidths and latencies determined by the links along the path.

Currently, Stardust supports the following link protocols: `mst` (Minimum Spanning Tree), `mst_loop`, `mst_smart_loop`, `pst` (Parallel Spanning Tree), `pst_loop`, and `pst_smart_loop`. A spanning tree ensures that the network graph forms a single connected component. The MST protocol runs Kruskal's algorithm on a single core to find the minimum spanning tree in the current constellation. The `pst` protocol filters for links that are eligible to connect and sorts links of satellites in parallel for pre-processing just before building the spanning tree. The `_loop` suffix indicates that nodes with few links also add loops to the closest other nodes with few links. The `smart_loop` variant adds loops to nodes with few links as well, but it attempts to find links that are in the opposite direction (relative to links previously established by the spanning tree) of existing links. The smart loop variant chooses these additional links to approximate a +Grid-like structure at those nodes. For each step, the `GroundSatelliteNearestProtocol` establishes a link from a ground station to its nearest satellite. Inter-satellite and ground-satellite link protocols run in parallel.

Routing protocols operate based on the established links and provide the latency of the shortest path through the built network graph. Currently, there are two `IRouter` implementations: `DijkstraRouter` supports pre-route calculations, so the step calculation includes the calculation of the routing tables per node. The routing table gets filled using Dijkstra's algorithm. When routing information is requested, only a simple routing table lookup is required to obtain the result. `AStarRouter` does not support pre-routing calculations, as it is a point-to-point path search algorithm. On a requested route, an A* algorithm searches for the shortest path to the target node or service.

C. SimPlugin Extensibility

To enable experimentation with different orchestration algorithms for the 3D Continuum, Stardust allows plugging custom code into simulations as *SimPlugins*. A SimPlugin is a lightweight mechanism to execute custom code at the beginning and at the end of every simulation step. To this end, a SimPlugin has full access to the simulation state, such as node positions, resources, and the network graph. Additionally, it can leverage simulator services to compute network routes on demand and deploy (simulated) workloads on nodes.

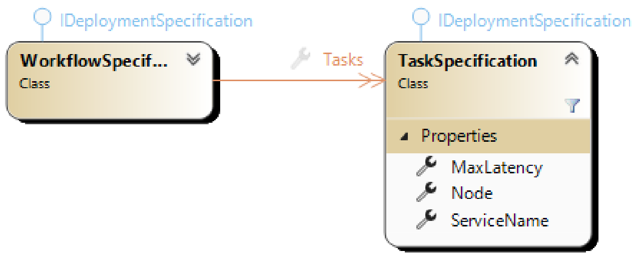

Fig. 4: Class Diagram of `WorkflowSpecification` with a `TaskSpecification`. `TaskSpecification` has a max Latency to either a Node or a Service Name Configured as a Requirement.

To deploy workloads in the simulation, there is the `IDeploymentOrchestrator` interface. There can be any number of such implementations registered to offer maximum flexibility for deployment algorithms and workloads. The implementation can access the simulation or other components by dependency injection in the constructor. A resolver delegates the requested workloads to the appropriate implementation that matches the workload requirements. Fig. 4 shows that a `WorkflowSpecification` consists of a list of `TaskSpecifications`. Each `TaskSpecification` can be configured with a max latency SLO, which, by default, applies to the connection from the predecessor task. However, the SLO can also be configured to refer to a specific service or node. Deploying a Workflow first resolves to the `WorkflowOrchestrator`, which is responsible for handling a `WorkflowSpecification` and scheduling each of its tasks. For each `TaskSpecification` it finds the most suitable node and deploys the task there using the `TaskOrchestrator`.

To create an additional orchestrator with new properties and scheduling strategy, e.g., to schedule a workload directly at the uplink satellite of a ground station, only a new class `DirectUplinkOrchestrator` implementing the `IDeploymentOrchestrator` interface, which handles instances of `DirectUplinkSpecification` with a `GroundStation` property, is needed. The new orchestrator has to be registered as a singleton to the `HostApplicationBuilder` of the hosting framework.

Listing 1 shows an outline of a simple workload scheduler SimPlugin implementation. All plugins must implement the `SimPlugin` interface, which provides handler methods for executing actions at the beginning and at the end of every simulation step. As a scheduler, the plugin must also implement the `IDeploymentOrchestrator interface`. Using dependency injection, the program has access to simulator services, such as the simulation state and simulation controllers. The simulation state allows inspecting all aspects of the simulation, e.g., the current node positions and the network graph. The simulation controllers provide interfaces to modify the simulation state, e.g., deploy a task on a node. The `SimpleScheduler` performs most work in `PostSimulationStep()`, where it dequeues the next task to be scheduled, then finds the most suitable node that fulfills the task's requirements, and, finally, deploys the task using the task orchestrator service of the simulation.

```
public class SimpleScheduler
    : ISimPlugin, IDeploymentOrchestrator {
  public SimpleScheduler(Simulation sim) {
    // Store sim and do
    // other initialization.
  }

  public void PreSimulationStep(
      int stepIndex, DateTime simTime) {
    // Do work before the next step at the
    // specified simTime executes.
  }

  public void PostSimulationStep(
      int stepIndex, DateTime simTime) {
    TaskSpecification task =
        this.DequeueNextTask();

    if (task) {
      // If there is a new task to be
      // scheduled on this iteration,
      // find a target node that satisfies
      // the task's requirements.
      // ...

      this.sim.TaskOrchestrator
          .deploy(task, targetNode);
    }
  }
}
```

Listing 1: Simple Workload Scheduler Stardust SimPlugin.

V. Evaluation & Implementation

In this section, we evaluate Stardust by integrating a simple network SLO-aware scheduler for the 3D Continuum as a SimPlugin and then conducting scalability experiments with the simulator. All code required to run the experiments is part of our open source repository[2].

A. Implementation

Stardust is implemented in C# using the cross-platform .NET 8 framework. The widely used `Microsoft.Extensions.Hosting` library simplifies the configuration and modularization of the application and facilitates the realization of the SimPlugin extensibility mechanism. Since each component is provided by dependency injection, replacing one component with another implementation only requires changing the hosting provider configuration. This simplifies the process of switching implementations and configurations of the simulation and all components.

B. Experiment Design

To evaluate Stardust, we mainly focus on its scalability, assessing its simulation performance and the performance of a simple workload scheduler plugin. To this end, we implement

[2]https://github.com/polaris-slo-cloud/stardust and https://doi.org/10.5281/zenodo.15484629

a simple scheduler SimPlugin that places serverless workloads while fulfilling the resource requirements for every workload.

We use a serverless workflow that is based on our flood disaster response use case from Section II. It consists of four functions that are meant to be executed in sequence, each with distinct resource requirements.

The experiments are grouped into three categories. A single iteration of an experiment comprises 100 simulation steps, with each simulation step covering one minute of simulated time. These 100 minutes of simulated time are the period required for all satellites to complete one orbit around Earth, rounded up to the next multiple of 10. The used link protocol in Stardust is `pst_smart_loop`, unless otherwise noted. The three experiment categories are the following:

1) *Simulator performance with respect to the infrastructure size*: We increase the total number of satellites for every experiment iteration, starting with 250 satellites and going up to 20k satellites, which is about three times the current size of Starlink. The simulation runs without any SimPlugins to focus fully on the simulator performance. We measure the end-to-end runtime of every iteration and the system resource usage. Additionally, we compare Stardust's end-to-end runtime with Celestial [22] and StarryNet [23], two state-of-the-art LEO mega constellation emulators.
2) *Scheduling performance with respect to the infrastructure size*: This experiment is the same as the first one, except that we schedule one workflow instance with our scheduler SimPlugin in every simulation step.
3) *Scheduling performance with respect to the workload*: We use our scheduler SimPlugin to deploy an increasing number of workflow instances per simulator step on a LEO mega constellation consisting of 6,882 satellites. We measure the scheduling time on each simulation step.

To set up realistic satellite orbits, we use TLE data on the orbits of 6,882 Starlink[3] satellites obtained from CelesTrack[4] on December 17, 2024. For iterations that require more satellites, we duplicate existing satellites and offset their epoch, such that each duplicate is on the same orbital plane and altitude as the original satellite, but at a different position. Since our focus is the satellites and since they require the most computational effort, because their positions need to be updated, we fix the number of ground stations (Clouds) to 85, which are distributed roughly equally across the globe.

All experiments are run on an Ubuntu 24.04 LTS VM with 32 CPU cores and 48 GiB RAM. The underlying server is running an Intel Xeon processor of the Skylake generation.

C. Experimental Results

1) Simulator performance with respect to the infrastructure size: We execute the experiment with seven satellite constellation sizes. The end-to-end execution time of an experiment iteration consisting of 100 simulation steps is indicative of

[3]https://www.starlink.com

[4]https://celestrak.org/NORAD/elements/

TABLE I: Simulators End-to-end Results

Sat Count	Stardust (sec)	StarryNet (sec)	Celestial (sec)
250	3	503.67	115
500	3	1400.33	126
1k	15	3457.67	163
2k	69	-	328
3k	162	-	673
6.8k	877	-	-
13.8k	2995	-	-
20.6k	6464	-	-

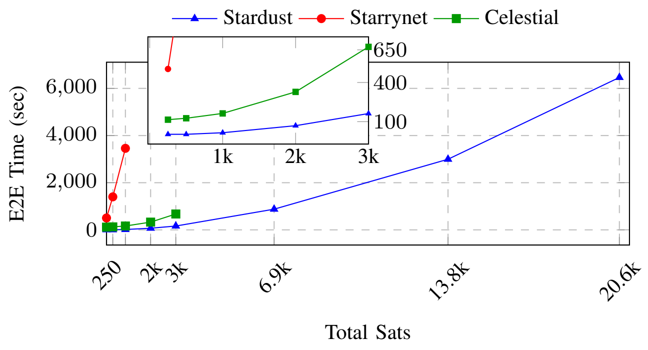

Fig. 5: Experiment End-to-End Runtimes for Total Satellite Counts.

the simulator's performance, because it shows how well the satellite position computations and network graph updates scale. Using the end-to-end execution time is also necessary to be able to compare Stardust to Celestial and StarryNet, because the latter two precompute all satellite positions before the simulation and launch a Firecracker microVM or a Docker container, respectively, for every node. Additionally, they both advance simulation time in real time. Thus, we configure them to simulate 100 seconds instead of minutes to avoid prolonging the end-to-end time artificially. Celestial allows saving computational resources by suspending the microVMs of all satellites that are currently outside of a bounding box. We use Europe as the bounding box, like in the example configuration supplied by Celestial

Fig. 5 compares the end-to-end experiment execution time of the simulators; the detailed results are in Table I. Due to the memory consumption of the microVMs, Celestial crashes during the iteration with 6.8k satellites when run on a single machine. StarryNet fails at the experiment with 2k satellites on a single machine, because Docker is limited to attaching at most 1,024 virtual Ethernet adapters to a network bridge. Thus, Celestial executes only the iterations up to 3k satellites and StarryNet up to 1k satellites. Since Stardust does not execute any VMs or containers for the simulated nodes, it can handle much larger scenarios. For better readability, Fig. 5 is split after 2k satellites. Even for small scenarios, Stardust executes experiments much faster than StarryNet. For 250, 500, and 1,000 satellites, Stardust takes 3, 3, and 15 seconds, respectively. Celestial requires 115, 126, and 163 seconds, respectively, whereas StarryNet needs 504, 1,400, and 3,457 seconds. We observe that the Firecracker microVM setup and teardown of Celestial is much faster than the respective Docker actions of StarryNet, which may, however, be attributed to the fact that

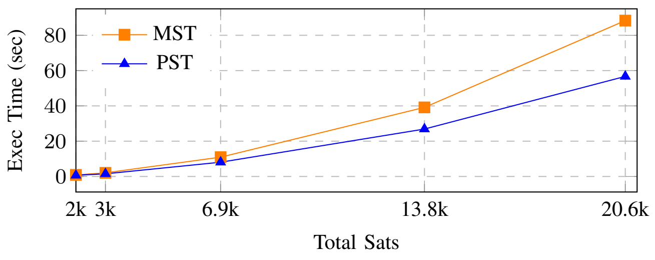

Fig. 6: Stardust Mean Execution Time for Single Simulation Step.

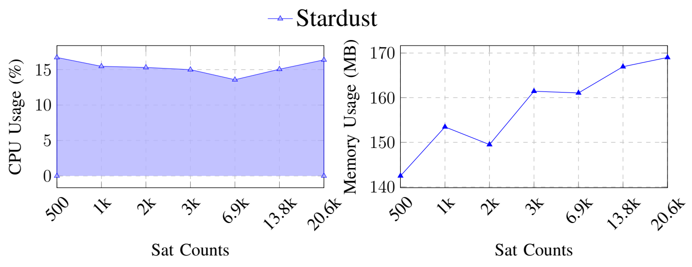

Fig. 7: Stardust Resource Usage for Total Satellite Counts.

StarryNet sends each command through an SSH tunnel, while Celestial uses its own protobuf protocol. Since Celestial and StarryNet progress the simulation in real time, 100 seconds is the lower bound for their end-to-end time. But due to state precomputation and microVM/container setup/teardown, even the rest of StarryNet's execution time significantly exceeds that of Stardust and also grows faster than Stardust's. For example, for 3k satellites Celestial needs 673 seconds in total, while Stardust only needs 162 seconds. The difference cannot only be attributed to microVM operations, because already the precomputation of the simulation state takes Celestial 432 seconds for this iteration. This suggests that Stardust is more efficient at computing the simulation state, e.g., Celestial and StarryNet precompute the latencies between all pairs of nodes. Stardust computes routes and latencies between a particular pair of nodes only when requested by a SimPlugin. Altogether, Stardust's end-to-end experiment execution time scales with log-linear complexity up to the largest experiment with 20,646 total nodes.

Fig. 6 analyzes the mean execution time of a single simulation step in Stardust, executing with the `mst_smart_loop` and `pst_smart_loop` link protocols. Both scale quadratically, with PST having a more gentle slope. For 20k total satellites, a simulation step takes approximately 88 seconds with MST and about 57 seconds with PST. The reason for PST not being even faster is that only the sorting part of Kruskal's algorithm is parallelized, indicating an avenue to improvement in the future.

Fig. 7 shows the mean resource usage of Stardust for 500 to 20k satellites. The mean CPU usage never exceeds 25% and memory usage goes up to 170 MB for 20k satellites. The resource usage for Celestial and StarryNet is not comparable, because they execute microVMs or containers for nodes, e.g., the 6.8k satellites experiment failed with Celestial, because the host's 48 GB of RAM was exhausted. Stardust intentionally does not execute nodes, hence, it can simulate much larger constellations on a single machine.

Stardust's resource usage in combination with the quasilinear scalability of the end-to-end experiment runtime indicates that it can scale to even larger constellation sizes on a single machine. Thus, it is well suited for the upcoming expansions of LEO mega constellations.

2) Scheduling performance with respect to the infrastructure size: In this experiment, we execute the same iterations as for the previous one, but, additionally, we use our scheduler SimPlugin to deploy one instance of the flood disaster response workflow, i.e., four functions, on every simulation step. We execute the experiment with both link protocols, i.e., `mst_smart_loop` and `pst_smart_loop`. Since Celestial does not support plugins and StarryNet's node model does not account for compute resources, we run this experiment only with Stardust.

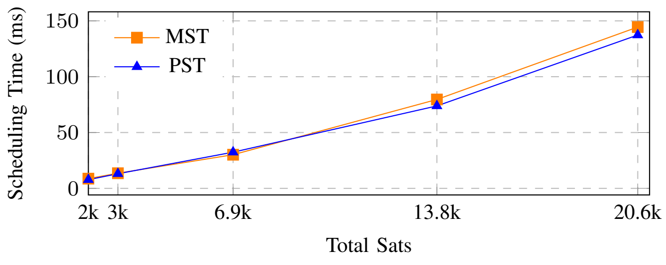

Fig. 8: Scheduling Performance for 1 Workflow on Increasing Constellation Sizes.

The goal of Stardust's SimPlugins is to provide a lightweight mechanism for integrating custom code into the simulation, such as an orchestration or scheduling algorithm that should be validated. Celestial provides no plugin mechanism to extend the simulation, but it offers a REST API that is accessible from within each microVM. This API allows querying network route information between two nodes and provides rudimentary information about the nodes. However, there is no way to query a node's resources and location or to deploy a workload in the 3D Continuum – this would require installing a full-fledged orchestrator, such as Kubernetes, in the scenario. StarryNet can be used as a library in a Python script to design and run a custom scenario. However, its API is limited, e.g., its node model has no concept of computational resources, which makes it difficult to write orchestration algorithms. Stardust's SimPlugins are a lightweight mechanism for executing custom algorithms directly as part of the simulation. SimPlugins have full access to the entire simulation state, which includes the compute resources on nodes and locations, and provides APIs to query network routes and deploy simulated workloads on the nodes. This enables SimPlugins to be used for evaluating orchestration algorithms without using a full-fledged orchestrator, like Kubernetes, which would introduce additional restrictions to the simulation.

Fig. 8 illustrates the mean duration of scheduling one workflow instance for various satellite constellation sizes.

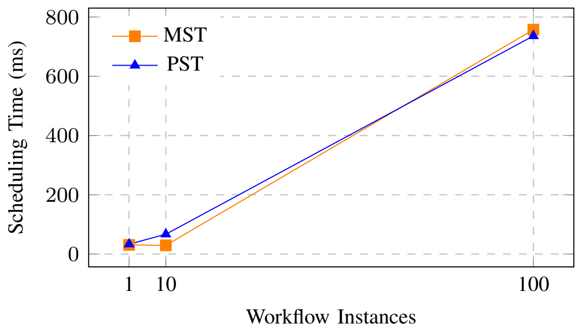

Fig. 9: Workload Scalability Experiment with 6,882 Satellites – Scheduling Time.

This shows that the performance of the simulator state and deployment APIs scales almost linearly with the infrastructure size and does not cause a bottleneck for SimPlugins.

3) Scheduling performance with respect to the workload: In the third experiment, we evaluate the speed and scalability of workload deployment operations in a Stardust SimPlugin. The number of satellites is fixed to 6,882, i.e., the size of the Starlink constellation on December 17, 2024. The simulation consists of 100 steps; in every step we schedule a fixed number of flood disaster response workflow instances. We perform three iterations, with 1, 10, and 100 scheduled workflow instances (four functions per instance) per simulation step. As for the previous experiment, due to the restrictions of Celestial and StarryNet, we run this experiment only with Stardust.

Fig. 9 shows the mean execution times of the scheduler SimPlugin. The time scales linearly from 33 ms when scheduling a single workflow instance per simulation step to 736 ms when scheduling 100 workflow instances per step. This shows that the simulator state API does not present any bottleneck and that it is suitable for validating 3D Continuum orchestration algorithms.

Table II provides a summarizing overview of all experiment results.

TABLE II: Stardust Experiment Results

Sat/Workload Count	Algo	Step (ms)	Sched (ms)	CPU (%)	RAM (MB)
Simulator Performance					
250	MST	13.51	-	7.19	143.41
250	PST	15.86	-	21.21	133.73
3,023	MST	2,056.38	-	8.92	152.38
3,023	PST	1,505.36	-	15.00	161.45
20,646	MST	88,330.54	-	9.05	166.40
20,646	PST	56,710.87	-	16.38	169.02
Scheduling Performance					
250	MST	25.88	1.23	10.53	141.92
250	PST	24.66	1.53	10.76	137.49
3,023	MST	2,194.02	13.54	9.98	160.40
3,023	PST	1,605.45	13.09	14.96	160.15
20,646	MST	87,905.50	144.44	9.35	184.88
20,646	PST	57,598.34	137.31	16.47	165.52
Scheduling Workload Scalability					
1	MST	10,598.31	30.78	9.76	167.36
1	PST	8,457.07	33.17	14.24	165.96
10	MST	10,716.2	29.41	9.58	166.25
10	PST	38,599.87	736.19	23.77	167.21
100	MST	42,901.99	757.21	21.49	168.30
100	PST	38,599.87	736.19	21.49	168.57

VI. Conclusion

In this paper, we have presented Stardust, a scalable and extensible simulator for the 3D Continuum. Stardust aims to provide a platform for evaluating new orchestration algorithms for the 3D Continuum on a single machine. To this end, Stardust simulates all nodes of the 3D Continuum, including the orbital movement of satellites, maintains a network graph, and allows progressing simulated time at a configurable pace for short-term or long-term simulations. Stardust intentionally does not emulate the execution of workloads to allow simulating mega constellations up to three times the current size of Starlink, as shown in our experiments.

In the near future, we intend to implement approximation of perturbed orbits and introduce emulator capabilities to Stardust to allow it to execute workloads using even more realistic network routes, including QoS-based routing, in the 3D Continuum. Users will be able to select whether Stardust should operate in simulator mode or emulator mode. We plan to introduce a lightweight emulator mode using containers and an orchestrated emulator mode, where Kubernetes can be used in the 3D Continuum. Our aim is to employ sparse execution of containers, i.e., only execute containers for nodes that have been assigned a workload, to maintain the scalability of Stardust. A distributed operation mode across multiple machines, in combination with sparse container execution, will continue to allow experiments with the growing LEO mega constellations of the future.

Acknowledgment

This work is partially funded by the Austrian Research Promotion Agency (FFG) under the project RapidREC (Project No. 903884). This work has received funding from the Austrian Internet Stiftung under the NetIdee project LEO Trek (ID 7442). This research received funding from the EU's Horizon Europe Research and Innovation Program through the TEADAL (GA No. 101070186) and NexaSphere projects (GA No. 101192912).

References

[1] Orbit.ing-now.com, "Low earth orbit," 2024. [Online]. Available: https://orbit.ing-now.com/low-earth-orbit/

[2] D. Shepardson, "Fcc chair wants more competition to spacex's starlink unit," 2024. [Online]. Available: https://www.reuters.com/technology/space/fcc-chair-wants-more-competition-spacexs-starlink-unit-2024-09-11/

[3] C. Henry, "Fcc oks lower orbit for some starlink satellites," *Space News*, 2019. [Online]. Available: https://spacenews.com/fcc-oks-lower-orbit-for-some-starlink-satellites/

[4] Federal Communications Commission, "Kuiper systems, llc – application for authority to deploy and operate a ka-band non-geostationary satellite orbit system – order and authorization." [Online]. Available: https://docs.fcc.gov/public/attachments/FCC-20-102A1.pdf

[5] N. Mohan, A. E. Ferguson, H. Cech, R. Bose, P. R. Renatin, M. K. Marina, and J. Ott, "A multifaceted look at starlink performance," in *Proceedings of the ACM on Web Conference 2024*, ser. WWW '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 2723–2734.

[6] D. Bhattacherjee and A. Singla, “Network topology design at 27,000 km/hour,” in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, ser. CoNEXT ’19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 341–354.

[7] European Space Agency, “European space agency-funded projects reach new performance level in groundwork for optical leo to geo data relays,” 2024. [Online]. Available: https://connectivity.esa.int/news/european-space-agencyfunded-projects-reach-new-performance-level-groundwork-optical-leo-geo-data-relays

[8] G. Mateo-Garcia, J. Veitch-Michaelis, C. Purcell, N. Longepe, S. Reid, A. Anlind, F. Bruhn, J. Parr, and P. P. Mathieu, “In-orbit demonstration of a re-trainable machine learning payload for processing optical imagery,” *Scientific Reports*, vol. 13, no. 1, p. 10391, 2023.

[9] B. Denby and B. Lucia, “Orbital edge computing: Nanosatellite constellations as a new class of computer system,” in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS ’20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 939–954.

[10] D. Bhattacherjee, S. Kassing, M. Licciardello, and A. Singla, “In-orbit computing: An outlandish thought experiment?” in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, ser. HotNets ’20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 197–204.

[11] R. Xing, X. Ma, A. Zhou, S. Dustdar, and S. Wang, “From earth to space: A first deployment of 5g core network on satellite,” *China Communications*, vol. 20, no. 4, pp. 315–325, 2023.

[12] D. Vasisht, J. Shenoy, and R. Chandra, “L2d2: low latency distributed downlink for leo satellites,” in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM ’21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 151–164.

[13] T. Pusztai, C. Marcelino, and S. Nastic, “Hyperdrive: Scheduling serverless functions in the edge-cloud-space 3d continuum,” in *2024 IEEE/ACM Symposium on Edge Computing (SEC)*, 2024.

[14] C. Marcelino, S. Gollhofer-Berger, T. Pusztai, and S. Nastic, “Cosmos: A cost model for serverless workflows in the 3d compute continuum,” in *2025 IEEE International Conference on Smart Computing (SMARTCOMP 2025)*, 2025.

[15] M. R. Jabbarpour, B. Javadi, P. Leong, R. N. Calheiros, D. Boland, and C. Butler, “Performance analysis of federated learning in orbital edge computing,” in *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*, ser. UCC ’23. New York, NY, USA: Association for Computing Machinery, 2024.

[16] H. Chen, M. Xiao, and Z. Pang, “Satellite-based computing networks with federated learning,” *IEEE Wireless Communications*, vol. 29, no. 1, pp. 78–84, 2022.

[17] C.-Y. Chen, L.-H. Shen, K.-T. Feng, L.-L. Yang, and J.-M. Wu, “Edge selection and clustering for federated learning in optical inter-leo satellite constellation,” in *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2023, pp. 1–6.

[18] M. Elmahallawy and T. Luo, “Optimizing federated learning in leo satellite constellations via intra-plane model propagation and sink satellite scheduling,” in *ICC 2023 - IEEE International Conference on Communications*, 2023, pp. 3444–3449.

[19] A. Furutanpey, Q. Zhang, P. Raith, T. Pfandzelter, S. Wang, and S. Dustdar, “Fool: Addressing the downlink bottleneck in satellite computing with neural feature compression,” *IEEE Transactions on Mobile Computing*, pp. 1–18, 2025.

[20] J. Liu, W. Jiang, H. Han, M. He, and W. Gu, “Satellite internet of things for smart agriculture applications: A case study of computer vision,” in *2023 20th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2023, pp. 66–71.

[21] 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*, ser. IMC ’20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 214–229.

[22] T. Pfandzelter and D. Bermbach, “Celestial: Virtual software system testbeds for the leo edge,” in *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, ser. Middleware ’22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 69–81.

[23] Z. Lai, H. Li, Y. Deng, Q. Wu, J. Liu, Y. Li, J. Li, L. Liu, W. Liu, and J. Wu, “Starrynet: Empowering researchers to evaluate futuristic integrated space and terrestrial networks,” in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, 2023, pp. 1309–1324. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/lai-zeqi

[24] M. Gravara, A. Stanisic, and S. Nastic, “A novel compound ai model for 6g networks in 3d continuum,” in *Proceedings of the 2025 EuCNC & 6G Summit*, 2025.

[25] European Space Agency, “Sentinel-2 operations.” [Online]. Available: https://www.esa.int/Enabling_Support/Operations/Sentinel-2_operations

[26] Airbus, “Airbus built sentinel-2c satellite successfully launched,” 2024. [Online]. Available: https://www.airbus.com/en/newsroom/press-releases/2024-09-airbus-built-sentinel-2c-satellite-successfully-launched

[27] European Space Agency, “European data relay satellite system (edrs) overview,” 2024. [Online]. Available: https://connectivity.esa.int/european-data-relay-satellite-system-edrs-overview

[28] J. Qin, T. Dong, Q. Guo, J. Yin, R. Gu, Z. Liu, Y. Tan, T. Zhang, and Y. Ji, “Dynamic simulation platform for software defined optical satellite networking,” in *Fiber Optic Sensing and Optical Communication*, J. Zhang, S. Fu, Q. Zhuge, M. Tang, and T. Guo, Eds., vol. 10849. SPIE, 2018.

[29] M. M. Kassem and N. Sastry, “xeoverse: A real-time simulation platform for large leo satellite mega-constellations,” in *2024 IFIP Networking Conference (IFIP Networking)*, 2024, pp. 1–9.

[30] 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.

[31] A. Varga, “Omnet++,” in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Güneş, and J. Gross, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 35–59.

[32] J. Puttonen, S. Rantanen, F. Laakso, J. Kurjenniemi, K. Aho, and G. Acar, “Satellite model for network simulator 3,” in *7th International ICST Conference on Simulation Tools and Techniques (SIMUtools)*. ICST, 2014.

[33] T. Schubert, L. Wolf, and U. Kulau, “ns-3-leo: Evaluation tool for satellite swarm communication protocols,” *IEEE Access*, vol. 10, pp. 11 527–11 537, 2022.

[34] B. Niehoefer, S. Šubik, and C. Wietfeld, “The cni open source satellite simulator based on omnet++,” in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, ser. SimuTools ’13. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013, pp. 314–321.

[35] A. Valentine and G. Parisis, “Developing and experimenting with leo satellite constellations in omnet++,” in *Proceedings of the 8th OMNeT++ Community Summit*, 2021. [Online]. Available: https://summit.omnetpp.org/2021/assets/pdf/OMNeT_2021_paper_6.pdf

[36] T. Pfandzelter and D. Bermbach, “Komet: A serverless platform for low-earth orbit edge services,” in *Proceedings of the 2024 ACM Symposium on Cloud Computing*, ser. SoCC ’24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 866–882.

[37] M. Di Carlo, S. Da Graça Marto, and M. Vasile, “Extended analytical formulae for the perturbed keplerian motion under low-thrust acceleration and orbital perturbations,” *Celestial Mechanics and Dynamical Astronomy*, vol. 133, no. 3, p. 13, 2021.