Machine Learning for Autonomic System Operation in the Heterogeneous Cloud-Edge Continuum

Contract Number 101092912

D4.3 Final Version of System Simulators

Lead Partner	UNICAL
Contributing Partners	UTH, INRIA, NVIDIA
Owner / Main Author	UNICAL (Gianluca Aloi, Claudio Savaglio)
Contributing Authors	UNICAL (Raffaele Gravina, Antonio Iera, Michele Gianfelice, Giancarlo Fortino), UTH (Christos Antonopoulos, Foivos Pournaropoulos), INRIA, (Valeria Loscri, Aya Moheddine, Jiali Xu), NVIDIA (Dimitris Syrivelis)
Reviewers	NTT DATA (Roberto Zambetti), FhP AICOS (Filipe Sousa)
Contractual Delivery Date	M24 (31 December 2024)
Actual Delivery Date	24/12/2024
Version	1.0
Dissemination Level	Public

HORIZON EUROPE

The research leading to these results has received funding from the European Community's Horizon Europe Programme (HORIZON) under grant no. 101092912.

Change Log

Version	Summary of Changes
0.1	Initial skeleton prepared by UNICAL
0.2	Added contributions for all the simulators
0.3	Refined and homogenised all sections
0.4	Draft edited according to Reviewers' comments
0.5	Final Draft for Internal Reviewers
0.6	Last proof-reading before submission
1.0	Final version for release to the EC

Table of Contents

List of Figures

List of Tables

Summary

This document follows up on D4.1, where the aims, features and limitations of the simulators identified for the data generation in MLSysOps were analysed. In particular, this document discusses how these simulators (i.e., CloudSim as a Cloud-based simulation environment, AeroLoop as a Drone simulation environment, EdgeCloudSim as an Edge-Node simulation environment, Open5gs/srsRAN as Network simulation environments, and OCS solver as an Optical switch environment) have been customised/extended, by presenting main interventions, their rationale and usage examples.

Abbreviations

BTS	Base Transceiver Station
CPU	Central Processing Unit
CI	Closed-In
dB	Decibel
FCD	Floating Car Data
GSM	Global System for Mobile Communication
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
KVM	Kernel-based Virtual Machine
IoT	Internet of Things
MMP	Markov Modulated Process
ML	Machine Learning
OCS	Optical Circuit Switching
PLE	Path Loss Exponent
POIs	Points of Interest
RG	Requirement Group
RSU	Roadside Unit
SUMO	Simulation of Urban Mobility
SF	Shadow Fading
SC	Street Canyon
SITL	Software In The Loop
SLA	Service-Level Agreements
UMi	Urban Micro Cellular

UAV	Unmanned Aerial Vehicle
VM	Virtual Machine
WLAN	Wireless Local Area Network

1 Introduction

The foundational idea of MLSysOps is the AI-driven resource management and application deployment/orchestration mechanisms across the computing continuum. In the MLSysOps approach, Machine Learning (ML) models play a key role in achieving such an autonomic SysOps scenario. However, for training, validating, and evaluating ML models, a large amount of high-quality (i.e. relevant, well-balanced, up-to-date, and bias-free) data is needed, whore retrieval comes with various challenges and issues that can significantly impact the final performance.

The collection of real-world data (raw or augmented ones) is an approach which typically ensures valuable data, but, especially in the context of System Operations (SysOps), collecting, storing, and managing large volumes of real-world data can incur significant costs with also relevant privacy, security, and quality issues. Therefore, the process of data collection has been combined with the data generation one for obtaining the so-called "synthetic data", which are totally generated in-vitro according to the property exhibited by the real data. In particular, the **simulation-based approach to data generation** is widely recognised and has a strong foundation across various research fields, with numerous established solutions developed over time. Simulators offer a cost-effective and low-risk means of creating large datasets for algorithm training, circumventing the limitations of real data and allowing for meticulous control over environmental conditions. Moreover, this approach enables the creation of privacy-preserving datasets. It facilitates the modelling of specific scenarios, rare events, and challenging conditions that may be difficult to capture in real-world data. Additionally, simulators can generate annotated data with ground truth labels, enhancing dataset quality for ML training and allowing iterative testing to bolster model robustness and reliability before real-world deployment. Finally, simulators support scalability, complementing real testbeds by providing data at varied scales, simulating critical cases that may be impractical to replicate, and producing detailed traces essential for informed management and configuration decisions. Wrapping up, the integration of data collection from testbeds and simulation-based data generation enables access to unique and high-detail data for ML model refinement; the reason why in MLSysOps we decided **to select a set of simulators** (the rule *"one size does not fit all"* always holds in complex scenarios, and we are no exception) and individually **customise/extend** them (there is no *"out-of-the-box"* simulator specifically and simultaneously purposed for both the SysOps domain and the computing continuum). The rationale for such activities of simulator selection and customisation is to match with the *Requirement Groups (RGs)* reported in Table 1.

Simulation Scenario	Involved Partner(s)	Selected Simulator(s)	Document Section(s)
SIM-1	UNICAL-INRIA	EdgeCloudSim-Open5gs/srsRAN	4-5
SIM-2	UTH	CloudSim	2
SIM-3	UTH	AeroLoop	3
SIM-4	NVIDIA	OCS solver	6

Table 1 Simulation scenarios, simulators, and discussion are included in this document.

Specifically, the first scenario, **SIM-1,** "network anomalies and system performance at the smart and far-edge", deals with networking- and service performance-related activities. Therefore, requirements like application deployment and management (*RG2*), node-level resource usage and management (*RG3*), wireless network management and security (*RG6*), and machine learning (*RG10*) can be evaluated by simulation. Then, **SIM-2** "power management and green energy exploitation" allows for assessing the node-level resource usage and

management (RG3), the energy-efficient and green computing in datacenters (*RG9*), and machine learning (*RG10*). The third simulation scenario, **SIM-3** "Drone Activity", simulates drone behaviour and addresses structured system and application descriptions (*RG1*), application deployment and management (*RG2*), and machine learning (*RG10*). Finally, **SIM-4**, "Optimization of optical network infrastructure under workload conditions", concentrates on activities associated with optical networks, such as optical networking in the data centre (RG8).

The rest of the document mirrors the D4.1 structure, being its follow-up, and briefly introduces the simulators that have been selected for the MLSysOps project, describes the interventions carried out for their extensions, and describes some usage examples to showcase them. In detail, for each simulator and for the sake of the deliverable self-consistency, we first report its main technical features, general strengths and its specific benefit/link to the MLSysOps project. Then, we present the implemented extensions through a valid and informative section, without claiming to be exhaustive technical documentation but rather a compendium of the most impactful interventions (which could have been either outlined from D4.1 or which emerged in progress). Finally, we show how the updated simulators can be used.

2 Cloud Datacenter Simulation Environment

2.1 Overview

The simulation of cloud server infrastructure is intended to enable **(1)** scale-out experiments, which is not possible with the small-scale physical cloud testbed, **(2)** experimentation with configuration parameters not available to end-users of public cloud infrastructures (such as jobs consolidation, node allocation control, node performance/power step configuration), and **(3)** controlled experimentation with varying green energy availability, which is not possible to perform (in a reproducible way) even if one had access to real cloud infrastructures. Accordingly, the selected cloud simulation environment is supposed to scale to hundreds of datacenter nodes to support customizable policies for provisioning host resources to virtual machines and to support the modelling and simulation of federated clouds. In this direction, the simulator's ability to dynamically switch nodes on/off and to support multiple Central Processing Unit (CPU) voltage/frequency steps, multiple memory capacities, and pluggable CPU and memory power profiles based on the characterization of real systems (ARM and Intel) are of utmost importance. Given these elements, the cloud datacenter simulation environment can be used to generate data aimed at training the ML model with respect to power efficiency and throughput/performance at the granularity of the datacenter for different single-/multi-cloud resource allocation and configuration strategies, Service Level Agreements (SLA) violations / total operating cost for different resource allocation and configuration strategies, and green energy utilization vs throughout, SLA violations, total operating cost.

The framework that was chosen as a basis for this work is CloudSim[1]. Developed by CLOUDS[2] Lab and distributed as a Java-based open-source software, CloudSim models and simulates cloud infrastructures and their use by cloud services. It is the de facto standard for cloud simulations in the research community. Most of the cloud simulators available in the literature, indeed, are either extensions of CloudSim (with a much smaller community), cover rather niche corner-cases, or have a limited adoption. There are many active versions of CloudSim (from 3.0.3 to 6.0), the last two (5.0 and 6.0) being in the beta stage. As an alternative simulation framework, we considered CloudSim Plus[3], which started as a fork of CloudSim 3.0 and has evolved with additional features, most notably support for modelling migration costs (also between different datacenters) and SLA agreements. However, CloudSim Plus has a significantly smaller adoption level, a smaller and less active community, and fewer publicly available resources and documentation compared with CloudSim. Therefore, we opted to build the cloud simulation environment using CloudSim as the base framework. More specifically, we use version 3.0.3, which is the most stable and enjoys the widest adoption by independent research groups – beyond the authors and maintainers.

CloudSim main advantages include the modelling of cloud resources at different levels of granularity (datacenters, nodes and node resources, VMs, network), its scalability in simulating large-scale infrastructures, and its ease of customization with user-defined models. The latter aspect is of paramount importance since the set of default models is straightforward and may miss important aspects of system behaviour for specific scenarios. Moreover, CloudSim supports event-driven (rather than real-time) simulations of Infrastructure as a

[1] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, “Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” Software: Practice and experience, vol. 41, no. 1, pp. 23–50, 2011.

[2] The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, http://www.cloudbus.org/.

[3] M. C. Silva Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio, and M. M. Freire. CloudSim Plus: a Cloud Computing Simulation Framework Pursuing Software Engineering Principles for Improved Modularity, Extensibility and Correctness, in IFIP/IEEE International Symposium on Integrated Network Management, 2017, pp. 400-406.

Service (IaaS) scenarios, which makes it an excellent choice to simulate a large-scale infrastructure that is managed based on decisions taken by MLSysOps.

The main features of CloudSim that are of particular importance to MLSysOps are briefly as follows:

- ***Support for multiple datacenters with multiple nodes each***. In CloudSim, the user can programmatically instantiate one or more datacenters during simulation configuration. Each datacenter instantiated is automatically assigned to the simulation. After creation, each datacenter is associated with a set of (one or multiple) hosts.
- ***Support for node heterogeneity***. Each host has one or more Processing Elements (PEs), which correspond to CPU cores. CloudSim can model heterogeneity in CPUs across different datacenters via extensions[4]. Other extensions introduce support for GPU simulation[5].
- ***Plugin policies***. CloudSim comes with resource allocation policies at two levels: Hosts and VMs. At the host level, each host allocates fractions of its resources (mainly PEs and memory) to VMs running on it. At the VM level, the scheduler allocates fractions of the resources allocated to VMs to "Cloudlets" (namely jobs) running on top of each VM. At an even higher level, the simulated "Datacenter Broker" assigns Cloudlets to VMs. There are both default space- and time-sharing policies. New policies can be quickly introduced by extending the respective classes of CloudSim, which is one of the main reasons the research community is driving the adoption of CloudSim.
- ***Power modelling***. CloudSim implements a rather simplistic power model. Its main parameter is node utilization, and it does not account for different CPU architectures, varying resource pressure, and different system configurations.

2.2 Extensions for MLSysOps

To cover the requirements of MLSysOps, we have extended CloudSim by implementing the following additional capabilities:

- ***Host configuration control at simulation time***. We have introduced explicit Voltage and Frequency Scaling (VFS) control for PEs. This control knob is available to host-level policies, which can trade off performance for power consumption and vice versa.
- ***Enhanced power modelling***. We have substituted CloudSim's default power model with more sophisticated, pluggable ones. The new power models consider—beyond CPU utilization—the operating voltage and frequency of the CPU, as well as its architecture. We have derived, validated, and implemented CloudSim power models (to the plug) for an ARM and an Intel-based multicore CPU.
- ***Enhanced performance modelling***. We have implemented support for different classes of jobs in terms of performance sensitivity to frequency scaling. More specifically, we distinguish between compute-intensive jobs (almost linear relation between CPU frequency and observed performance), memory- and I/O-intensive jobs (observed performance practically unaffected by CPU frequency reduction) and hybrid jobs (sublinear relation between CPU frequency and observed performance).
- ***Modelling of SLAs and SLA violation costs***. We have implemented support for Service Level Agreements (SLAs) and SLA violation costs. Service Levels are expressed as maximum CPU capacity

[4] M. Zakarya, L. Gillam, “Modelling resource heterogeneities in cloud simulations and quantifying their accuracy”, Simulation Modelling Practice and Theory, Volume 94, 2019, Pages 43-65, ISSN 1569-190X, https://doi.org/10.1016/j.simpat.2019.02.003.

[5] Siavashi, A., Momtazpour, M “GPUCloudSim: an extension of CloudSim for modelling and simulation of GPUs in cloud datacenters”. Journal of Supercomputing, Volume 75, Pages 2535–2561 (2019). https://doi.org/10.1007/s11227-018-2636-7

and maximum memory requirements for each job. If the simulated system cannot satisfy those requirements, a Service Level violation occurs, and the cloud provider must pay a penalty to the job owner. Our implementation is flexible and can support multiple penalty calculation methods.

- ***Support for popular input traces***. CloudSim supports the programmatic description of job mixes (arrival times, CPU and memory requirements in time and life duration). Alternatively, it can digest PlanetLab traces[6], which, however, are deprecated. We have introduced support for using Google Cloud traces[7] as input to CloudSim. This involves a script to preprocess trace files and code in the simulator to read the pre-processed trace and introduce jobs to the system at the points in time and with the characteristics specified in the trace. We expect our methodology to be applicable to other sources of cloud traces as well (for example, Alibaba traces[8])
- ***Inter-datacenter migrations and modelling of migration cost***. CloudSim supports VM and cloudlet migrations within the same datacenter. We have introduced support for inter-datacenter migration as well. This makes it possible for a management/scheduling policy to request the migration of cloudlets and VMs between hosts that belong in different datacenters.
- ***Brown vs green energy consumption & operation cost.*** We have implemented support to vary the cost of energy in a datacenter as a function of time in a controllable way during the simulation. The operational cost of the datacenter is calculated based on the energy consumption and current price of energy at each scheduling period. Note that in modern energy markets, the price of energy varies with a frequency from once every 15 minutes down to once per minute. The availability and exploitation of green energy in a datacenter (or a wider geographical region) is modelled similarly by reducing the energy price during the periods when green energy is available for consumption. This way, the volatility of green energy is captured via the respective price fluctuations, allowing management/scheduling policies to adopt a uniform cost-oriented optimisation approach.

2.3 Usage Examples

To functionally validate CloudSim enhancements, we designed and executed 6 experimental scenarios, described in the following paragraphs. Table 2 summarizes the new functionalities tested by each of the experimental scenarios.

Feature	Validation / usage scenario					
	1	**2**	**3**	**4**	**5**	**6**
Host configuration control at simulation time	X		X			
Enhanced power modelling	X	X	X			
Enhanced performance modelling	X	X	X			
Modelling of SLAs and SLA violation costs		X		X		X
Support for popular input traces	X	X	X	X	X	X

[6] https://planetlab.cs.princeton.edu/datasets.html

[7] https://github.com/google/cluster-data

[8] https://github.com/alibaba/clusterdata

Feature	Validation / usage scenario					
	1	2	3	4	5	6
Inter-datacenter migrations and modelling of migration cost				X	X	X
Brown vs green energy consumption & operation cost	X				X	

Table 2: Functional validation of CloudSim enhancements with different experimental scenarios

1. Dynamic energy-efficient workload management

This experiment tests the functionality at the simulated node level. We use workload traces, and energy availability (caps) and pricing data as input to the simulator. We assume different types of jobs (compute-, memory- and I/O-bound) to simulate different effects of node frequency management on performance. We vary node frequency and validate the effects on SLA violation risks (higher for compute-bound jobs), on power consumption, and on energy cost.

2. SLA-aware resource allocation

This experiment highlights the effect of different levels of resource availability guarantees in SLAs and of different SLA models on the number and cost of violations. We use workload traces and the associated SLAs (models, resource requirements, violation costs) as input, and simulate nodes with different reliability levels (less / more prone to errors) and different operating costs. We validate the functionality of the simulator with simple job placement and job migration heuristics. We observe jobs with tighter SLAs being placed / migrated to more reliable nodes to reduce penalty costs, whereas jobs with less strict SLAs are placed on less reliable yet more cost-effective nodes to optimize operating cost.

3. Adaptive workload scheduling for diverse job types

This scenario combines the two previous experiments to validate the simulator functionality for scheduling diverse workloads (compute-intensive, memory-intensive, hybrid), considering SLAs, and evaluating power consumption. We use workload traces and the associated SLAs (models, resource requirements, violation costs), as well as the time-series of varying energy costs as input. We apply heuristics and validate that jobs are assigned to CPUs appropriate for their characteristics (compute-intensive jobs to CPUs configured at high frequencies, memory-/IO-intensive jobs to lower clocked CPUs) and are dynamically migrated away from overcommitted nodes. We also validate that node configuration and node utilization affect power consumption, and that energy pricing affects the simulated energy cost.

4. Inter-datacenter migrations

This experiment validates the inter-datacenter VM migration functionality of the simulator. More specifically, given a workload trace and the associated SLAs, we initially place the jobs on multiple datacenters. One datacenter abruptly loses a significant percentage of its resources (simulating a disaster). We validate the capability of the simulator to migrate / restart VMs allocated on the affected nodes to other datacenters. We also validate that the simulator accounts for the respective SLA violation costs, and that energy consumption and costs per datacenters reflect this abrupt change in workload distribution.

5. Multi-datacenter cost optimization

We simulate datacenters with different energy costs / green energy availability. The input, beyond VM traces, is energy cost and green energy availability per datacenter, and a graph of inter-datacenter migration cost factors.

We experiment with extreme variations in energy costs and availability and validate that they trigger VM migrations across datacenter and that the associated migration costs are accounted for. We also validate the impact on operating costs due to energy consumption and SLA violations.

6. Usage scenario: ML-driven migration cost optimization

As a first step in exploiting the enhanced CloudSim to implement and evaluate ML-driven policies, we have designed a deep surrogate model (PeakLife) to predict, for each VM, the future CPU utilization (average and maximum) per scheduling period and the remaining execution life. The model uses an architecture based on a Gated Recurrent Unit (GRU), as a member of an encoder-decoder structure, combined with a multihead attention mechanism. We combine the model with a simple VM selection heuristic, which uses the aforementioned predictions to select VMs to be migrated away from oversubscribed hosts. The initial results, using traces from the Azure publicly available dataset[9], indicate that our ML-based deep surrogate predictor outperforms existing VM selection policies not based on predictions (LRMMT), as well as simpler (LSTM) ML-based predictors. Moreover, our predictor attains results close to those attained by using information from an oracle (that knows the future behavior of VMs) as input to the VM selection heuristic. Table 3 summarizes the results in terms of the reduction of migrations and SLA violations.

	Migrations reduction (% compared with LRMMT)	**SLO violations (% of scheduling periods)**
LRMMT	-	6.02
PeakLife	*29.25*	*4.46*
LSTM	13.14	6.03
Oracle	34.16	4.41

Table 3: Performance improvement using an ML predictor to control VM migration decisions on CloudSim (lower is better)

2.4 Simulator Release

The original CloudSim is released under the Apache v2.0 license. Our extensions will follow the same open-source approach and will be available on the MLSysOps website (https://mlsysops.eu/simulators/).

[9] https://github.com/Azure/AzurePublicDataset

3 Drone Simulation Environment

3.1 Overview

The main purpose of the drone simulation environment is to enable fast, hassle-free, and safe testing of scenarios where some of the nodes of the system are vehicles, with a special focus on drones. In particular, it should enable **(1)** system developers to test the MLSysOps framework for scenarios that involve mobile nodes and, in particular, Unmanned Aerial Vehicles (UAVs) in a flexible, controlled, safe, and efficient way by using virtual nodes instead of real ones; **(2)** researchers to run a wide range of mission scenarios using virtual drones to collect data that can be used to train the ML models as well as to evaluate the ability of such models to take (good) application and system management decisions. Both functions are essential, given the substantial overhead and limitations when conducting experiments with real mobile nodes and UAVs in the field, which require extensive preparations, are very time-consuming and come with several constraints due to weather conditions and safety reasons.

The drone simulation environment is based on AeroLoop[10], which UTH developed as a virtual test ground in the context of the RAWFIE H2020/FIRE+ project[11]. AeroLoop was tested within the RAWFIE project, and since then, it has been used by UTH in different variants to develop and test its own system and application software for drone-based systems[12,13,14]. The main concept of AeroLoop is not to reinvent the wheel by developing new simulators but to combine existing well-known simulators into a more unified simulation environment. More specifically, AeroLoop is designed in a modular fashion, making it possible to plug in state-of-the-art simulation technologies to reproduce the behaviour of mobile nodes and the performance of wireless communication. Compared to other drone simulation environments, such as FlyNetSim[15] and the more recent DroNS-3[16] and ArduSim[17], the main advantage of AeroLoop is that the whole software stack of each virtual node (e.g. UAV) runs within an isolated runtime environment (e.g. VM) in the same way as this would be the case in a real node, and communicates with other virtual nodes via standard network interfaces as these would be exposed to the software stack in a physical machine/node. This makes it possible to port the entire software

[10] M. Koutsoubelias, A. Grigoropoulos and S. Lalis, "A Modular Simulation Environment for Multiple UAVs with Virtual WiFi and Sensing Capability", IEEE Sensors Applications Symposium (SAS), March 2018.

[11] https://www.rawfie.eu/

[12] N. Grigoropoulos and S. Lalis, "Simulation and Digital Twin Support for Managed Drone Applications", IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT), September 2020.

[13] M. Koutsoubelias, N. Grigoropoulos, G. Polychronis, G. Badakis and S. Lalis, "System Architecture for Autonomous Drone-based Remote Sensing", International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous), November 2021.

[14] N. Grigoropoulos and S. Lalis, "Fractus: Orchestration of Distributed Applications in the Drone-Edge-Cloud Continuum", IEEE Computers, Software, and Applications Conference (COMPSAC), June 2022.

[15] S. Baidya, Z. Shaikh and M. Levorato, "FlyNetSim: An Open Source Synchronized UAV Network Simulator based on ns-3 and Ardupilot", ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 2018.

[16] P. Hall, J. Diller, A. Moon and Q. Han, "DroNS-3: Framework for Realistic Drone and Networking Simulators", 9th Workshop on Micro Aerial Vehicle Networks, Systems, and Applications, 2023.

[17] J. Wubben, C. T. Calafate, F. Granelli, J.-C. Cano and P. Manzoni, "Real-time co-simulation framework for multi-UAV environments offering detailed wireless channel models", IEEE International Conference on Communications (ICC), 2023.

environment of a virtual node to real platforms with small effort, which is very important when the goal of the simulation is to test software intended for use in real drones, as this is the case in the research testbed of UTH and the smart agriculture application use case of the project (see D5.1 Research and Application Testbeds).

The most important features of AeroLoop are as briefly follows:

- ***Virtual mobile and edge nodes.*** The simulation environment can support different system configurations, with one or more virtual UAVs (vUAVs) and/or virtual non-mobile edge nodes (vEdgeNodes).
- ***Real Autopilot with SITL support.*** vUAVs use the popular open-source Ardupilot autopilot[18]. Ardupilot natively supports a software-in-the-loop (SITL) configuration[19], in which case it runs coupled with a flight-dynamics engine. Notably, the same autopilot stack is used in real UAVs (drones), including UTH's custom drone(s) (see D5.1 Research and Application Testbeds).
- ***vUAVs and vEdgeNodes as separate VMs.*** The autopilot and any additional system/application software run in a standard Linux environment within a VM on top of a Kernel-based Virtual Machine (KVM) environment. This allows the whole software stack of a vUAV to be tested in AeroLoop before being used on a real drone. Similarly, edge-based system/application software runs in a separate VM embodying an edge node. It can be directly ported to run bare metal on real nodes with minimal changes.
- ***Simulated wireless communication.*** The system/application software running in vUAVs and vEdgeNodes communicates through first-class network interfaces. These are interconnected under the hood via KVM to a virtual Wi-Fi network implemented using the NS3 network simulator[20], which runs in a separate VM and provides realistic WiFi network performance as a function of the distance between the communicating nodes.
- ***Management Network Interface.*** All VMs have a network interface dedicated to management operations. All these interfaces are interconnected via a bridge to a separate management network within KVM, which is used to access the VMs and run configuration commands. The management network is also used to forward vUAV position updates to the wireless network simulator via the ZeroMQ library[21] using a pub/sub scheme.
- ***Flexible Customization.*** Thanks to the modularity of the environment, it is possible to run simulations using a more straightforward and lightweight configuration depending on the needs of a given test scenario. For example, suppose it is not important to focus on realistic Wi-Fi behaviour. In that case, the Wi-Fi network interfaces of the vUAVs and vEdgeNodes can be interconnected via a simple bridge without running NS3. As another example, suppose it is not important for the vUAV and vEdgeNode environments to be as close as possible to the one physical drone/node. In that case, they may be launched as containers instead of VMs to reduce the resources (mostly the amount of memory) needed for the host environment to run the simulation.
- ***Data Generation.*** During execution, the status of each vUAV and vEdgeNode can be retrieved by logging into the respective VMs via SSH and inspecting the desired log files or running state inspection commands. Also, the system/application software running in a vUAV and vEdgeNode can independently generate telemetry and send the data to any external application via the Internet interface, just as this would happen in a setup with real drones/nodes. As an example, the simulation environment

[18] ArduPilot. http://ardupilot.org/

[19] ArduPilot – SITL Simulator. http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html

[20] NS3 Network Simulator. https://www.nsnam.org/

[21] ZeroMQ. https://zeromq.org/

can be used to collect flight-related data on vUAVs, performance data for the Wi-Fi links between vUAVs and vEdgeNodes, and end-to-end delays for processing and interactions.

Notably, AeroLoop does not come with a Graphical User Interface (GUI). Nevertheless, the system/application software running in the virtual nodes can generate telemetry and send the data to any external application. Thus, such data can be captured and displayed using self-developed or existing third-party GUIs. As a concrete example, one can capture and visualise the current position and movement of each vUAV and observe the status and sensor readings of the autopilot via the MissionPlanner application[22].

3.2 Extensions for MLSysOps

To cover the requirements of MLSysOps, additional extensions to AeroLoop were made to support the following functionality:

- ***virtual ground vehicles/rovers (vRovers) in addition to UAVs***, which can be used to emulate the behaviour of ground vehicles/rovers coupled to corresponding autopilot SITL configurations.
- ***vUAVs and vRovers with an additional network interface***, which can be used as a proxy for wireless Internet connectivity via mobile communication technology such as 4G and 5G.
- ***vEdgeNodes with an additional network interface***, which can be used as a proxy for wired Internet connectivity via a LAN or VDSL link.

These **additional interfaces are interconnected** via suitable bridges to the Ethernet network interface of the KVM host environment, which provides LAN/Internet connectivity so that vUAVs, vRovers, and vEdgeNodes can communicate over LAN and the public Internet with real nodes and VMs running in remote machines/clusters.

External physics simulation. The physics-related simulation is performed externally to the vUAVs and vRovers using Gazebo[23] which runs on a separate machine equipped with a suitable graphics card. Notably, all virtual vehicles are part of the same physics simulation context. This makes it possible, among other things, to have physical interactions between vehicles and other objects and between vehicles and other vehicles. For example, a vehicle may not be able to perform a certain movement due to another object/vehicle standing in the way.

Virtual cameras. Through Gazebo, vUAVs and vRovers can be equipped with virtual cameras that capture video/images based on the current position of the vehicle, the mounting of the camera on the vehicle, the view angle of the camera, and the terrain used for the simulation. Moreover, one can place additional virtual cameras to capture the terrain and moving vehicles from a “bird’s-eye view” perspective.

Realistic simulation of mobile communication. The simulation of mobile communication for virtual nodes (vUAVs and vRovers) is supported through a dedicated network interface that is mapped to the Ethernet/LAN interface of the KVM host. In addition, suitable adapters are added within the respective VMs for all traffic that goes through this interface using the Linux traffic control subsystem’s tc[24] utility, which introduces extra latency and bandwidth capping. Suitably configuring the adapter can achieve realistic mobile 4/5G communication performance.

[22] Mission Planner. https://ardupilot.org/planner/

[23] Gazebo Simulator: https://gazebosim.org

[24] tc: https://man7.org/linux/man-pages/man8/tc.8.html

3.3 Usage Examples

As described in the D2.1 Requirements and Evaluation Plan, the drone simulation environment will be used to validate and evaluate functionalities related to the requirements groups RG1 Structured System and Application Descriptions, RG2 Application Deployment and Orchestration, and RG10 Machine Learning.

In fact, the simulation environment has already been used to test functionalities related to RG1 and RG2 in conjunction with the smart agriculture application use case, which involves two mobile nodes, a tractor (vRover) and a drone (vUAV), described in more detail in the following. On the one hand, this significantly helped debug the MLSysOps framework and ensure the smooth integration of all components. On the other hand, it enabled testing the full control loop for the smart agriculture application use case under a variety of different scenarios, which are practically impossible to replicate (and even less so in a reproducible way) in the real world/field. It is important to stress that the same setup can be used to develop and evaluate ML models that will take system/application management decisions through a wide range of suitably designed scenarios, e.g., varying weed detection performance at different points in time/locations in the field, something that is not possible to do in real field tests in a controllable way.

The concrete setup of the simulation environment for the smart agriculture use case is illustrated in Figure 1. It includes two virtual mobile nodes, one for the tractor (vRover) and one for the drone (vDrone), which feature (simulated) 4G and WiFi network interfaces. Both virtual vehicles run the node-level components of the MLSysOps framework. These virtual nodes are managed by an MLSysOps cluster-level agent running in an external VM hosted on a server machine of the UTH testbed, as this will be the case when using the real tractor and drone in the smart agriculture testbed.

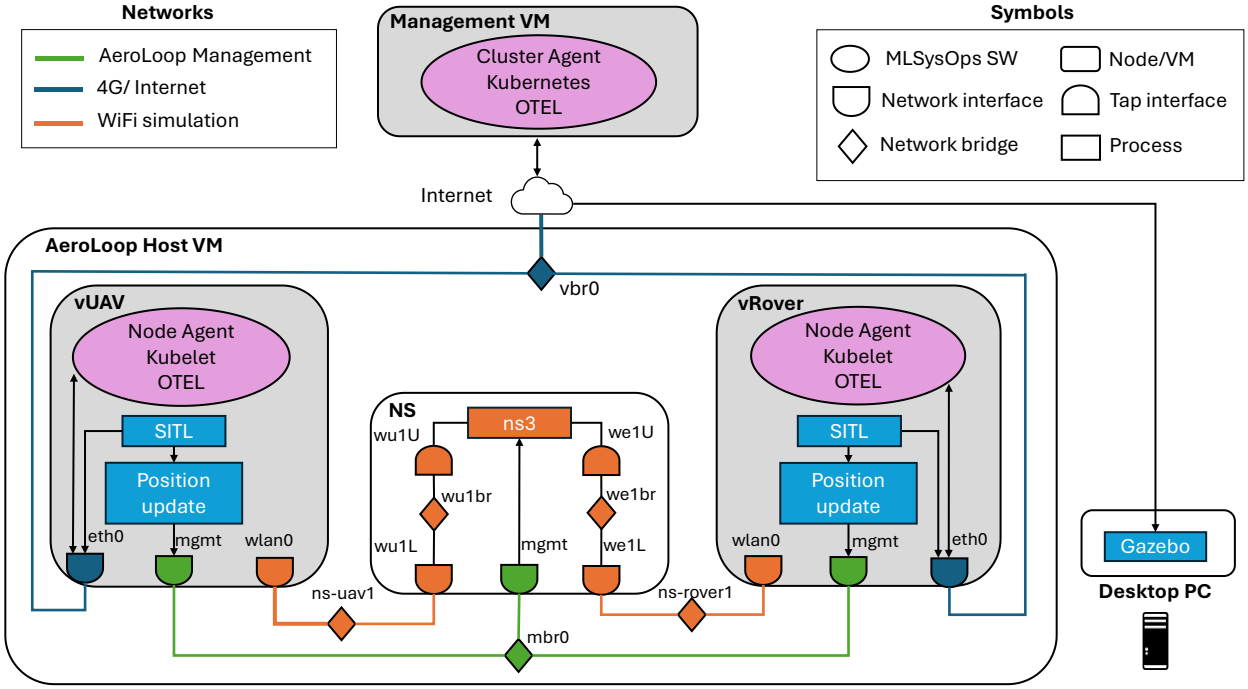

Figure 1: Setup of a drone simulation environment for the smart agriculture use case.

Figure 2 shows a screenshot of a dashboard that was built using Grafana[25], which visualises the camera views of the tractor and drone during the simulation. The dashboard also includes a gauge showing the current performance of the WiFi communication between the two nodes (which varies as a function of their position).

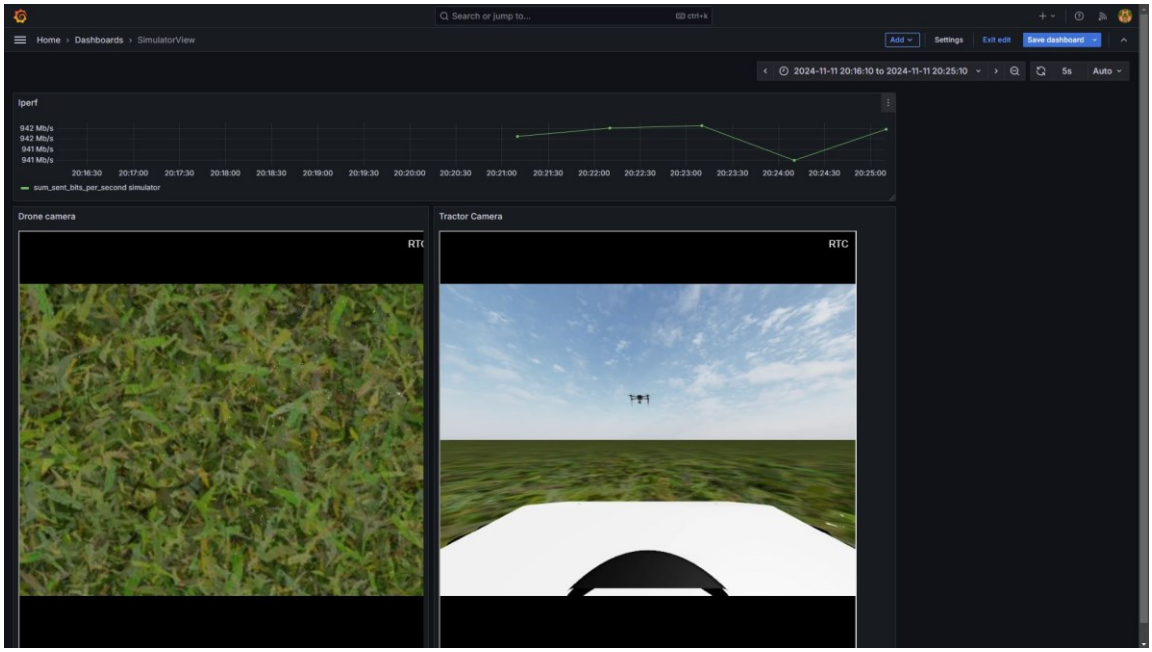

Figure 2: Dashboard visualising the camera streams and WiFi performance of the virtual tractor and drone.

Figure 3: Screenshot of the MissionPlanner view for the virtual tractor and drone.

[25] https://grafana.com/

Figure 3 shows a screenshot of the MissionPlanner tool, which can be used to track the state of the tractor and drone during simulation through the MavLink protocol. Note that this is done in the same way as for real nodes that support MavLink, e.g., the drone used in the real-world smart agriculture testbed.

The virtual nodes run real application components (containers) that are deployed via MLSysOps based on proper application descriptions. For practical purposes, the application components are mere proxies. Nevertheless, they generate the expected telemetry, which the MLSysOps agents use to make system and application management decisions.

The simulation environment was used to test the whole system for a wide range of scenarios, where:

- The tractor scans a given area following a predefined path defined as a sequence of waypoints.
- The drone is dynamically engaged, in which case it takes off, approaches and follows the tractor.
- The drone is dynamically disengaged, in which case it stops following the tractor and returns to land.
- Application components are dynamically deployed on the tractor and the drone.
- System and application performance metrics are captured via MLSysOps telemetry and are used to decide the engagement and disengagement of the drone.

Notably, the simulation environment can be used to collect realistic data on drone behaviour. For instance, in the case of the smart agriculture application, an important metric that can be measured using the simulation environment is the time it takes for the drone to take off and reach the tractor so that it can start effectively contributing to weed detection. In turn, this information can be used to learn the overhead of drone engagement (as a function of the drone's home position and the tractor's position in the field) and consider this to make more informed/intelligent decisions regarding the engagement of the drone.

3.4 Simulator Release

As NS-3, the modified NS-3 TAP/Bridge code (tap-bridge.cc) is released under the Open GPLv2 licence[26]. The AeroLoop scripts for the configuration of the virtual nodes (VMs) and NS are released under the Open GPL3 license[27]. Pointers to the software and the documentation for installation and configuration will be available on the MLSysOps website (https://mlsysops.eu/simulators/).

[26] https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html

[27] https://www.gnu.org/licenses/gpl-3.0.en.html

4 Edge Nodes Simulation Environment

4.1 Overview

Aiming for a reliable outcome, the simulation of edge nodes in the computing continuum demands an environment which (1) covers the whole simulation space by taking into account those infrastructural, computational and network aspects which simultaneously impact real Internet of Things (IoT) systems operations and performance; (2) supports the modelling of heterogeneous scenarios in which edge devices, possibly mobile, can either locally execute tasks or autonomously decide to offload them according to customisable policies (e.g., prioritise Quality of Service (QoS), green energy, cost, etc.); (3) allows monitoring and collecting data about important SysOps metrics related, for example, to hardware utilisation, service performance and failures. After a careful state-of-the-art analysis, **EdgeCloudSim** has emerged as one of the richest edge–cloud simulators that can address these requirements. Developed in 2017 as an extension of CloudSim with additional functionality specifically aimed at Edge and Continuum Computing, EdgeCloudSim is a discrete event simulator distributed with a GPL-3.0 license, written in Java and already used by UNICAL from 2019 in several research works of heterogeneous Smart-* domains, aiming to preliminary assess the performance of IoT services in many configuration settings before its actual deployment[28,29,30] but also its inclusion in application-agnostic methodology as in this paper[31]. In the last 4 years, other IoT simulators have been proposed. Still, none of them matches these built-in properties with the continuum computing aspects or with the ease of customisation featuring EdgeCloudSim, the reason why it has also been chosen for the MLSysOps project, albeit no more with the goal of disclosing IoT system configurations rather for the data generation and validation task. In detail, EdgeCloudSim allows simulating systems comprising (mobile) devices, edge servers and cloud servers, which are individually modelled in terms of computer power and memory space. Devices can execute or offload computing Tasks in the continuum according to policies that consider the dynamic resource availability and the position of the devices themselves with respect to the edge/cloud servers. The goal of the simulation is to support the decision-making process by evaluating the QoS and the resource utilisation related to a given IoT system's configuration provided as input. In particular, the quality of service is assessed in terms of service time (with the possibility of distinguishing between computation and networking time), success ratio service failures (which can be due to lack of network or computation resources or mobility) and cost. In such a way, the analysis of the output of the simulation supports the engineering or re-engineering of the target scenario by providing insights into infrastructural, architectural or algorithmic aspects. With reference to MLSysOps, EdgeCloudSim allows generating datasets and system traces about important SysOps metrics like system, network and resource utilisation in different settings of workloads, infrastructures and offloading policies in the continuum. Moreover, it can be integrated with third-party tools for defining and simulating ad-hoc ML-driven offloading policies: in particular, external tools can interact with

[28] Casadei, Roberto, et al. "A development approach for collective opportunistic edge-of-things services." Information Sciences 498 (2019): 154-169.

[29] Aloi, Gianluca, et al. "Simulation-driven platform for Edge-based AAL systems." IEEE Journal on Selected Areas in Communications 39.2 (2020): 446-462.

[30] Barbieri, Alessandro, Fabrizio Marozzo, and Claudio Savaglio. "Iot platforms and services configuration through parameter sweep: a simulation-based approach." 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2021.

[31] Savaglio, Claudio, and Giancarlo Fortino. "A simulation-driven methodology for IoT data mining based on edge computing." ACM Transactions on Internet Technology (TOIT) 21.2 (2021): 1-22.

the EdgeCloudSim and provide inputs for its configuration files (for example, some parameters in this paper[32] take value from a classification activity carried out by Weka). Finally, since managing parameters programmatically can be difficult, EdgeCloudSim separates the system configuration files from the simulation engine: in particular, a configuration can be defined as a list of <parameter, value> entries and multiple configurations executed in parallel and easily compared through a parameter-sweep approach. Moreover, EdgeCloudSim requires no installation, and the simulation results are saved in comma-separated value (CSV) data format by default, but they can be graphically displayed through MATLAB scripts.

4.2 Extensions for MLSysOps

EdgeCloudSim is designed with a modular architecture to implement key functionalities (such as wireless LAN and WAN transmission, device mobility, realistic and tuneable load generation, etc.) separately within classes which are loosely coupled. EdgeCloudSim has five main modules available: Core Simulation, Networking, Load Generator, Mobility and Edge Orchestrator. To ease fast prototyping efforts, each module contains a default implementation that can be easily extended according to the software engineering good practices defined in the factory pattern. This ease of customisation of main EdgeCloudSim modules is extremely relevant since the mobility, the load generation and the orchestration policies have a great impact on the SysOps metrics. To provide reliability in the simulation and compliance with the MLSysOps objectives, some interventions have been implemented with respect to the current version (4.0) of EdgeCloudSim. An online GitHub repository is available at https://github.com/CSpemeLab/Mlsysops.git, and the main extensions with respect to the original version are briefly described below.

Advanced Energy Models. Default EdgeCloudSim energy models of edge nodes, edge- and cloud servers are quite simple, and they have been improved since MLSysOps gives great relevance to the energy awareness of the target system. We have introduced an energy consumption model at all layers, including mobile, edge, and cloud datacenters. Indeed, battery powering was an option originally not considered in EdgeCloudSim. Therefore, according to its philosophy (and also to software engineering best practices), which contemplates pairs of abstract-concrete java classes, we have extended the original classes “EdgeHost”(deputed to manage VMs and other resources within an EdgeServer) in “EdgeHostEnergy", “CloudHost” (deputed to manage VMs and other resources within a Cloud Server) in “CloudHostEnergy”, and “MobileHost” in `MobileHostEnergy` (deputed to manage VMs and other resources within a mobile node), so as to programmatically (i) set, statically or randomly, the initial battery level of a given device according to its type; (ii) monitor its energy level in a certain period or during the whole simulation time; and (iii) update the remaining energy by setting the device as dead at the moment of its complete energy depletion, which is also recorded. These interventions required the definition of proper fields in all the extended classes, as well as the definition of conventional get/set methods and proper constructors. Moreover, the three classes “MobileServerManager”, “CloudServerManager” and “EdgeServerManager” have been extended (i.e., new methods defined) with the aim of retrieving, respectively, the energy consumed by all the mobile, edge and server instances: this is useful for having an overview of the whole simulation scenario and also to properly manage the death devices (i.e., stop their task generation, their movement and immediately mark as failed their on-going tasks). Indeed, the option of a suddenly unavailable node was not considered in the original EdgeCloudSim. Thus, it needed an additional intervention. Instead, the rules governing the energy flow within a device have been defined by the “DefaultEnergyComputingModel”, an extension of the “EnergyComputingModel” class and compliant with the energy model defined by

[32] Sonmez, Cagatay, et al. "Machine learning-based workload orchestrator for vehicular edge computing." IEEE Transactions on Intelligent Transportation Systems 22.4 (2020): 2239-2251.

PureEdgeSim[33] (another well-known simulator in the edge scenario). The introduced energy model is considered the main energy-consuming activities of both computation and networking: in particular, it is worth noting that CPU-related energy consumption varies according to the time spent in computation or idle mode, while the per-bit energy, either in transmission or reception, depends on the type of LAN-WLAN connection and communication protocol type. Such a level of detail was necessary to implement an effective yet realistic energy module consumption: indeed, the factors mentioned earlier, especially the communication-related ones, are often the dominant root of energy consumption for IoT devices. Whereas also mobility contributes, directly or indirectly, to energy depletion, it is not currently considered: such a choice is due to the heterogeneity of mobile nodes (for example, the energy spent by drones, UGV and tractors for moving is very different) and to the peculiar needs of mobile-carriable/wearable devices (e.g., smartphone) which may deplete energy in mobility passively, for example, due to networks change.

Finally, at run-time, the novel class "SampleScenarioFactoryEnergy" (implementing the "ScenarioFactoryEnergy") creates an instance of the simulation scenario, uploads all the needed modules and launches the novel class "SimManagerEnergy". In its turn, this initialises the simulation by extending the "SimManager" class. As a result, devices are created with the associated parameters and simulation events, including the ones related to energy management. Hence, the events start being triggered to feed the event-driven simulation engine, and, in parallel, log data are created both during the simulation and at its end.

Enhanced events tracing. A deep-trace event modality has been enabled to trace events online and not exclusively when the simulation ends. Moreover, fine-grained log mechanisms have been defined in the class "SimLogger.java" with the twofold aim of *(i)* providing both a "per-node" perspective and a "per-service" perspective; and *(ii)* logging some intermediate events (i.e., step-by-step device movements) which are not currently tracked, but they are key to reconstruct the simulation dynamics but also to simplify the monitoring of the data generation process. This, in turn, is aimed at feeding the ML models. As a side effect of this intervention, the simulation takes some more seconds/minutes since the Input/Output file management in Java is not very efficient, mainly due to the buffer management system. Still, overall, the usability of the simulator is not compromised.

Fine-grained Node Modelling. By default, EdgeCloudSim allows the exclusive modelling of CPUs as computing resources. However, GPU and other accelerators are key devices in MLSysOps and can be preferable or mandatory for specific applications. In this regard, we have enriched the devices' description (in the file "default_config.properties" for cloud and mobile nodes and "edge_devices.xml" for edge server) and the application's requirement (in the file "applications.xml") with the GPU tag to consider such an aspect. Along with the energy tag, this intervention allows for fine-grained node modelling. It enables the implementation of offloading policies, which specifically consider GPU as a first-class element to assign tasks.

Realistic Mobility Patterns. EdgeCloudSim provides simple models based on random mobility patterns. Given the relevance of the mobility aspect in the computing continuum, more advanced mobility coming from the specific-purpose simulators SUMO has been introduced. In particular, the integration of SUMO with EdgeCloudSim occurs in three distinct steps. To support the simulation of realistic scenarios, we used a real map of the city of Cosenza (Italy) city. Our edge datacenters are placed next to the Vodafone cellular operator's base stations (BTS) towers. The real latitude and longitude positions of BTSs were taken from https://lteitaly.it/, and the use of actual positioning of edge datacenter helps to guarantee that the model of vehicular mobility accurately matches the situations that occur in the real world.

33

We utilise *OSMWebWizard.py* to extract the map. This information is provided to SUMO to generate floating car data based on points of interest (POI). The information on floating car data is stored in the *fcd-data-xml file*, which contains detailed information about vehicle movements at different time steps. The extended mobility model tracks vehicle positions during the simulation. Its major methods are Initialize and getLocation. The Initialize method retrieves FCD using the FCDReader (a class developed to track vehicle movement in the EdgeCloudSim).

The getLocation method in the VehicularMobilityModel.java class obtains and logs the exact geographic coordinates of a designated vehicle at a certain moment during the simulation. The function necessitates two parameters: deviceId, which denotes the distinct identifier of the vehicle, and time, which specifies the exact point in the simulation for which the position is being requested. The method begins by identifying the vehicle object associated with the provided deviceId. The program systematically analyses each timestep in the list to find the desired time. After the time is determined, it retrieves the <x, y> coordinates of the vehicle at that specific time interval. The getNearestDataCenterByPower method is used to compute the nearest edge datacenter based on the power threshold.

Initially, the task is passed to the getDeviceToOffload method, which is defined in the VehicularEdgeOrchestrator class, to retrieve the device ID. The device ID specifies the destination for the task transfer, which can be either the edge datacenter or cloud datacenter via RSU or cloud datacenter via a 5G cellular network for offloading the task. Vehicles connect to the closest edge datacenter by selecting the one with the highest signal strength. We performed the simulations taking one vehicle into consideration, which moves into the map extracted from the SUMO and five edge datacenters. The vehicle is assigned the edge datacenter or base station ID on the basis of signal.

Additional network features. Finally, additional features have been implemented to make the network modelling more realistic. By default, the Network model provided by EdgeCloudSim handles the transmission delay in the WLAN and WAN by considering both upload and download data, available bandwidth and the number of connected devices based on a single server queue model. To obtain a more realistic representation of the communication channel and make it compliant with advanced technologies and protocols, such as 5G, a close-in (CI) free space reference distance Path Loss (PL) model (PL^{CI}) that incorporates a realistic path loss exponent and considers both frequency and distance, has been implemented. Thanks to this model, latency evaluations are more accurate and better support decisions regarding the task offloading. The reference PL^{CI} model is given in (1), where f is the frequency in Hz, n is the PL exponent (PLE), d is the distance in meters, X_{σ}^{CI} is the shadow fading (SF) standard deviation and d is the distance between transmitter and receiver.

The term FSPL (2) stands for Path Loss in Free Space, which is the reduction in signal power between a transmitter and receiver when they are 1 meter apart at the carrier frequency f and with the speed of light c. To simplify calculations and ensure consistency over a wide range of scenarios, the model is primarily dependent on the path loss exponent (PLE). The signal strength along a particular path may be diminished due to obstructions such as buildings, trees, and other barriers. Using the Close-in (CI) path loss model's standard deviation improves signal estimation. Within the context of the CI Path Loss Model, our simulation utilised the Urban Micro-Cellular (UMi) Street Canyon (SC) scenario. It depicts congested, challenging urban conditions used in vehicle edge computing systems. The CI model considers many elements. These include path loss exponent (PLE), shadow fading standard deviation (SF), and non-line-of-sight conditions.

$$PL^{CI}(f, d)[dB]=FSPL(f, 1m)[dB]+10n\log10(d)+ X_{\sigma}^{CI} \quad (1)$$

$$FSPL(f, 1m)[dB]=20\log10(4\pi f/c) \quad (2)$$

4.3 Usage Examples

The mobility model is validated through the process depicted in Figure 4. Points of Interest POIs of Vodafone 5G base stations are collected in terms of latitude and longitude points, which reflect the real-world situation. On the other hand, SUMO generated a realistic vehicular mobility dataset that is stored in an XML file, which includes information about the location and speed of the vehicles. The entire integration process of SUMO and EdgeCloudSim is depicted in the Figure 4.

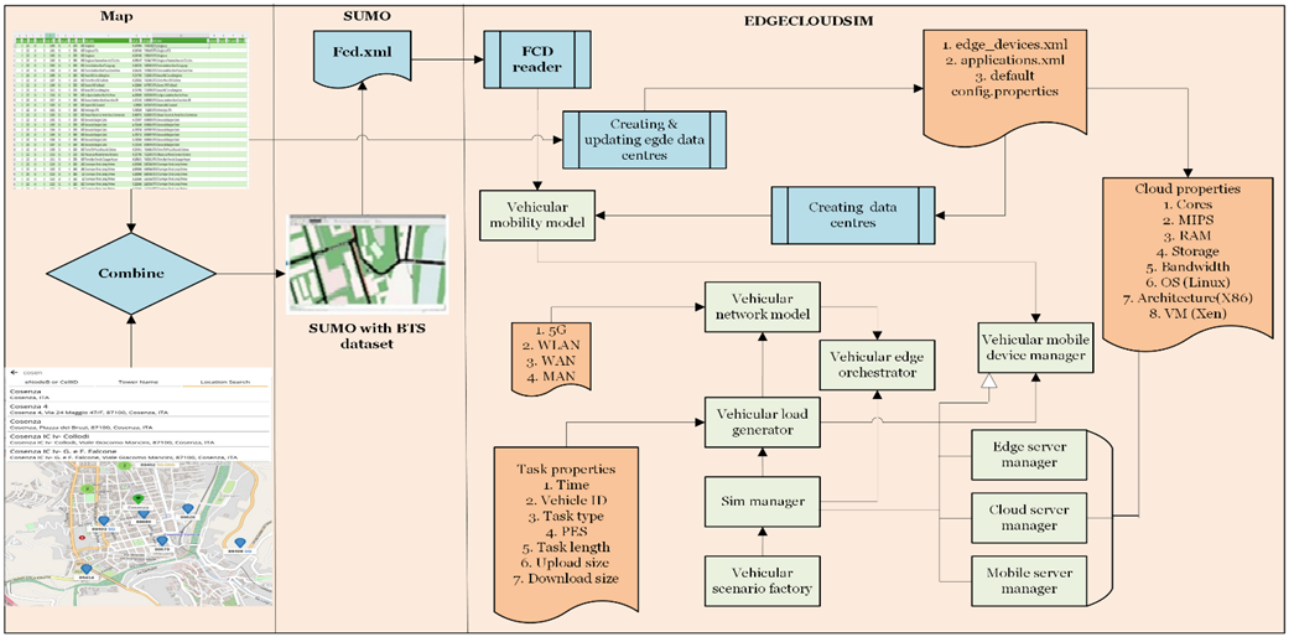

Figure 4: The integration process of SUMO and Edgecloudsim

The validation was performed by generating large-scale mobility patterns based on more than 1000 vehicles and 28 edge data centres. EdgeCloudSim successfully processes the vehicle data to update the edge data centres based on vehicular locations. The range of one KM is defined, which means the power/signal strength of the base station is calculated, and the WLAN ID of the edge datacenter/5G base station is assigned based on signal strength. Different types of networks, including 5G, WLAN, and WAN, are used to show diverse connectivity scenarios and analyse their impact on task offloading. The further validation of the vehicular mobility model is achieved by generating different computational tasks, which include navigation, infotainment, and hazard detection, using different parameters, including task type and task size. The single server queuing model has already been implemented in EdgeCloudSim to simulate different communication standards, including WLAN, MAN, WAN, and GSM-based cellular networks. EdgeCloudSim uses its wrapper class to simulate the behaviour of these various communication standards. We replaced the simple GSM model with 5G characteristics and implemented the realistic channel of 5G through the CI path loss model. The queuing behaviour related to various networking standards is handled in the wrapper class, which stores the different types of information, such as the size of the tasks and Poisson mean. Their transmission delay is handled by MMP/M/1, which is a single server queening model. In addition to considering the Poisson arrival rate, this model also considers the size of the task. The GSM model is incorporated into the simulator's existing network model; however, it likely only partially reflects some of the real elements that make up a GSM channel. This is because it uses simplified queue-based computations and does not include numerous factors found in real GSM networks. Given these specific conditions, the GSM model exhibits a limitation in that it tends to oversimplify, leading to an inaccurate representation of actual propagation delays and unpredictability in the real world. Bandwidth and task arrival rates based on the Poisson distribution are utilised to estimate and compute upload and download delays in GSM task outsourcing. Tasks are transferred, taking into consideration network and propagation delays. One way to significantly improve the accuracy of network simulations is to replace the

fundamental GSM model with a 5G model that makes use of the optimisation equation of the CI model. The portrayal of contemporary urban areas is improved if the 5G model incorporates a realistic path loss exponent and takes into consideration both frequency and distance. Because of this, latency estimates would be more accurate, and decisions regarding task offloading would be more well-informed. We performed the simulations taking one vehicle into consideration, which moves into the map extracted from the SUMO and five edge data centres; the vehicle is assigned the edge datacenter or base station ID on the basis of signal strength received from the base station, which varies with the distance, as shown in Figure 5. In fact, the signal strength decreases when the distance between the vehicle and edge datacenter increases, resulting in increased path loss, which is shown in the figures. Initially, the task is passed to the getDeviceToOffload method, which is defined in the VehicularEdgeOrchestrator class, to retrieve the device ID. The device ID specifies the destination for the task offloading, which can be either the edge datacenter or cloud datacenter via RSU or cloud datacenter via 5G cellular network.

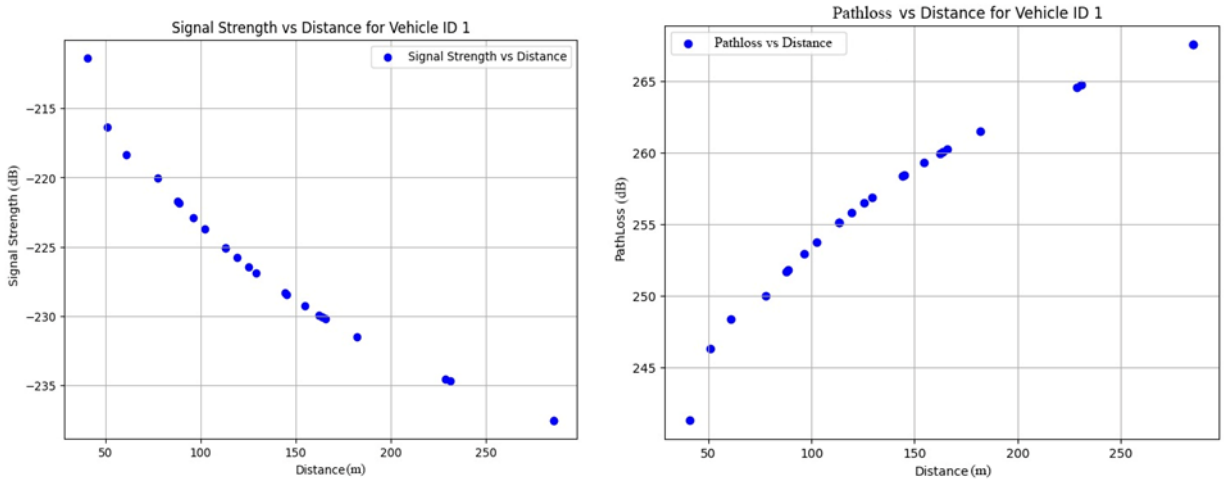

Figure 5: The impact of mobility on signal strength and path loss

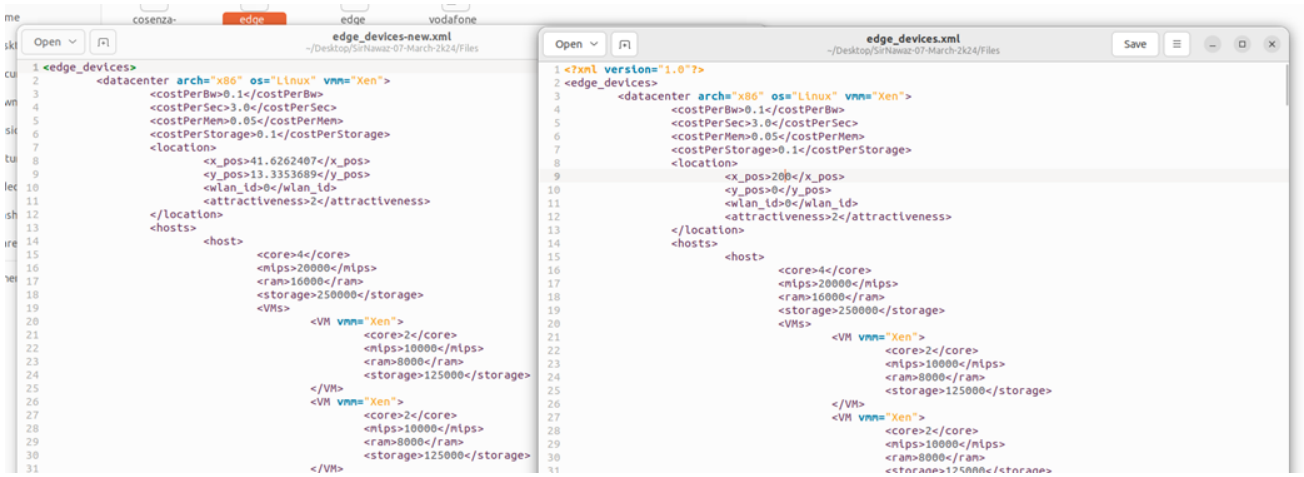

Figure 6: Edge Datacenter's positions in Edgecloudsim simulator

Figure 6 reflects the difference between the positions of edge data centres before and after the integration; the left side shows the x-y positions of edge devices, which are points of interest of Vodafone base stations present in the city of Cosenza, hence reflecting the true positions. In contrast, the right-hand side of the computer snapshot shows the original random positions of edge devices, where the y-axis reflects zero value.

For the energy-related extension, the implemented approaches for computing and networking expenditures follow the well-established linear power models.[34, 35] Therefore, they do not need further or ad hoc validation. Indeed, these models are widely used because they are computationally inexpensive, affordable in large-scale scenarios, and straightforward to implement (unlike Nonlinear Models, which, however, can also be used in the case of devices with variable performance states, i.e., power savings from dynamic voltage and frequency scaling). Finally, the implemented mechanisms for enhanced event tracing have been validated programmatically since they have a limited impact on the simulation engine and a null one in the simulation logic.

4.4 Simulator Release

EdgeCloudSim is released under the GPL-3.0 license. Accordingly, our extensions will follow the same open-source approach and will be available on the MLSysOps website (https://mlsysops.eu/simulators/).

[34] Beloglazov, Anton, et al. "A taxonomy and survey of energy-efficient datacenters and cloud computing systems." Advances in computers 82 (2011): 47-111.

[35] Barroso, Luiz André, and Urs Hölzle. "The case for energy-proportional computing." Computer 40.12 (2007): 33-37.

5 Network Simulation Environment

5.1 Overview

Aiming for a robust and comprehensive analysis of networking metrics in the computing continuum, the network simulation environment has been designed to (1) integrate physical-layer realism with system-level metrics to capture critical factors affecting resource allocation, security, and anomaly detection, (2) enable dynamic modelling of 5G network scenarios, including node mobility, connection management, and real-time responses to network conditions; and (3) provide a foundation for ML-driven anomaly detection and resource optimisation by facilitating the extraction of meaningful features from system logs, network traces, and device metrics. Building on the limitations of existing simulators, a detailed evaluation of state-of-the-art solutions led to the adoption of srsRAN and open5gs, which together offer a modular and realistic framework for simulating 5G network operations, including SNIR, RSSI, PDR, bit rate, etc. These metrics are critical for studying network behaviour under normal operations and adversarial conditions such as jamming attacks.

Previous frameworks like SimGrid, while valuable for generally distributed system simulations, need more physical layer modelling to emulate the network behaviour in a real 5G setting. In contrast, srsRAN and open5gs enable detailed modelling of the RAN and core network, providing both the SA and NSA configurations. This allows the environment to simulate end-to-end 5G operations, bridging gaps between traditional network simulators and the requirements of MLSysOps. By focusing on the interaction between network parameters and system performance, the environment supports the development and validation of ML models capable of anomaly detection and adaptive resource allocation, ensuring alignment with MLSysOps objectives like trust, scalability, and security in the IoT-edge-cloud continuum.

The network simulation environment also supports the generation of rich datasets and traces, which are essential for validating ML algorithms in realistic scenarios. For example, features like bit rate, packet drop rates, and uplink performance indicators can be extracted from gNB and UE logs. These provide essential inputs for ML-based detection of jamming attacks and security threats. To evaluate these capabilities, controlled experiments were conducted using a testbed comprising open5gs for the core network, srsRAN for the RAN, and a software-defined radio (SDR)-based jammer to simulate interference. These experiments demonstrated the environment's ability to identify anomalies through ML-driven analysis and to adjust resource allocation dynamically in response to network conditions.

Furthermore, the network simulation environment provides scalability and flexibility, allowing simulations to span from localised setups to complex edge-cloud systems. It enables the modelling of heterogeneous scenarios, supporting both static and mobile nodes, and offers a detailed representation of 5G-specific dynamics like path-loss, multipath fading, and shadowing effects. These features position the simulator as a key component for generating datasets and testing solutions that address MLSysOps objectives, such as improving trust in IoT-edge systems and enabling cross-layer anomaly detection in complex networks.

By integrating these advanced features, the network simulation environment addresses the gap between traditional simulators and the specialised needs of MLSysOps. It offers a unique platform for studying the interplay of network and system metrics under diverse conditions, ensuring robust validation of ML-based frameworks for resource management and anomaly detection. This comprehensive approach ensures that the MLSysOps framework is well-prepared to meet the demands of modern IoT and edge computing systems while advancing the state of the art in network simulation research.

In brief, the network simulation environment should enable the analysis of networking metrics combined with devices and systems metrics in the continuum and provide realistic modelling of the physical layer for considering key factors that could play a fundamental role in resource allocation and anomaly/attack detection.

5.2 Extensions for MLSysOps

The network simulation environment has been significantly extended to align with the MLSysOps objectives, emphasising security, scalability, and the interplay between networking and system performance metrics. These extensions enhance the simulation capabilities of the environment, focusing on the realistic modelling of 5G networks and facilitating ML-based anomaly detection and resource management. The improvements have been implemented across various modules, ensuring flexibility, modularity, and ease of integration. The primary extension, which reflects the MLSysOps requirements, is detailed below:

Enhanced Feature Extraction

A feature extraction module has been added to support ML-driven anomaly detection and adaptive resource allocation. This module captures key metrics such as SNIR, RSSI, CQI, MCS, bit rate, and uplink/downlink performance from gNB and UE logs. These features are critical for detecting jamming attacks and optimising resource utilisation in dynamic 5G environments. The module integrates with the srsRAN and open5gs frameworks, ensuring seamless data collection during simulation runs.

Dynamic Anomaly Simulation

A dynamic jamming simulation framework has been introduced, leveraging an SDR-based jammer to generate controlled interference scenarios. This extension enables the simulation of various attack patterns, including power-modulated jamming and selective channel interference. The impact of these anomalies is monitored in real-time through network logs, providing a robust dataset for training ML models to detect and mitigate such attacks.

5.3 Usage Examples

As outlined in D2.1, the primary objective of the network simulation environment is to manage node connections and application deployment/adaptation based on ML-driven analysis of security-related parameters. The dynamic nature of 5G networks, where nodes frequently join or leave, poses a challenge in maintaining reliable communication and optimal application performance. The inclusion of power-modulated jammers adds further complexity, disrupting normal operations and threatening data integrity by introducing intentional interference.

Machine Learning-Driven Anomaly Detection

To address these challenges, ML algorithms were integrated into the simulation environment to analyse network behaviour and identify patterns indicative of jamming attacks. This involves continuously monitoring network logs from User Equipment (UE) and gNodeB (gNB) to extract critical features such as Signal-to-Noise-Interference Ratio (SNIR), Received Signal Strength Indicator (RSSI), bit rate, Packet Delivery Ratio (PDR), and packet drop counts. These features are then processed by an ML model to detect anomalies associated with jamming attacks.

The ML-driven framework enables:

- **Real-time Detection**: Identifying abnormal signal degradation caused by power-modulated jammers.
- **Automated Response**: Triggering actions such as re-routing traffic, disconnecting compromised nodes, or deploying additional nodes to maintain service quality and reduce disruption.

Testbed Setup

The extensions have been validated using real-world datasets and testbeds. Open5gs and srsRAN were configured to replicate a SA 5G network, with a real Android smartphone acting as the UE. Logs from these setups were analysed to ensure the reliability and accuracy of the extracted features. The inclusion of SDR-

based jamming scenarios allowed for detailed comparisons between normal and attack conditions, demonstrating the effectiveness of the simulation environment in capturing network dynamics and anomalies.

The key components of the setup are:

1. **5G Core Network and RAN**:
    a. Open5gs was employed for the core network, providing essential 5G core functionalities such as the Access and Mobility Management Function (AMF) and Session Management Function (SMF).
    b. srsRAN was used for the Radio Access Network (RAN) to emulate the behaviour of gNBs and their interactions with the core network.
2. **User Equipment (UE)**:
    a. A real Android smartphone was configured to connect to the 5G network, generating real-world traffic data. The smartphone was used to simulate typical user behaviour under both normal and adverse conditions.
3. **Jammer Setup**:
    a. A Software-Defined Radio (SDR)- -based jammer was deployed to simulate various jamming attacks, including random and targeted power-modulated interference. This setup allowed precise control over jamming power and patterns to replicate real-world attack scenarios.

Experimental Methodology

The validation process was structured into three key phases to ensure a comprehensive assessment of the simulation environment:

1. **Feature Extraction and Analysis**:
    a. Logs from gNB and UE were analysed to extract critical features that could indicate potential jamming activity. Features such as SNIR, RSSI, PDR, bit rate, and packet drops were prioritised due to their sensitivity to network anomalies.
    b. These features were pre-processed and formatted for compatibility with ML models.
2. **Scenario-Based Testing**:
    a. **Normal Operation Scenario**: The 5G network was tested in a standard configuration without any interference. This scenario established a baseline for feature behaviour, providing a reference for identifying deviations caused by jamming.
    b. **Jamming Scenario**: The SDR-based jammer was used in controlled jamming experiments. By varying its power and attack patterns, different types of jamming attacks, including continuous and intermittent interference, were simulated. The experiments captured the impact on network metrics and identified anomalies in the logs.
3. **Behavioral Comparison**:
    a. The extracted features were analysed across normal and jammed scenarios. Key differences, such as sudden drops in SNIR or significant increases in packet loss, were identified as strong indicators of jamming.
    b. These patterns were used to validate the ML-driven anomaly detection framework, ensuring its reliability and robustness in detecting jamming attacks.

Enhanced Validation Techniques

To further enhance the validation process, the following additional steps were included:

1. **Ground Truth Labeling**:
    a. For each scenario, the network state (normal or jammed) was manually labelled to create a ground truth dataset. This dataset was used to evaluate the accuracy of the ML model.

2. **Model Evaluation Metrics**:
   Performance metrics such as precision, recall, and F1-score were calculated to measure the effectiveness of the ML model in detecting jamming attacks. This ensured that the model's predictions aligned with the observed network behaviour.
3. **Stress Testing**:
   The system was tested under high network loads and dense node configurations to evaluate its scalability and robustness. The ability to maintain accurate detection and reliable communication was assessed in these scenarios.

Results and Observations

The experiments demonstrated that the network simulation environment could reliably detect anomalies caused by jamming attacks. Key findings include (as shown in Figure 7 and Figure 8):

- **Feature Sensitivity**: Metrics such as SNIR and PDR were particularly sensitive to jamming, making them strong candidates for anomaly detection.
- **ML Model Accuracy**: The ML model achieved high precision and recall in detecting jamming attacks, with F1 scores consistently above 90%.
- **Resilience to Jamming**: The system's automated response mechanisms effectively mitigated the impact of jamming, maintaining service quality in most cases.

Figure 7: ML performance on testing subset for gNB traces dataset

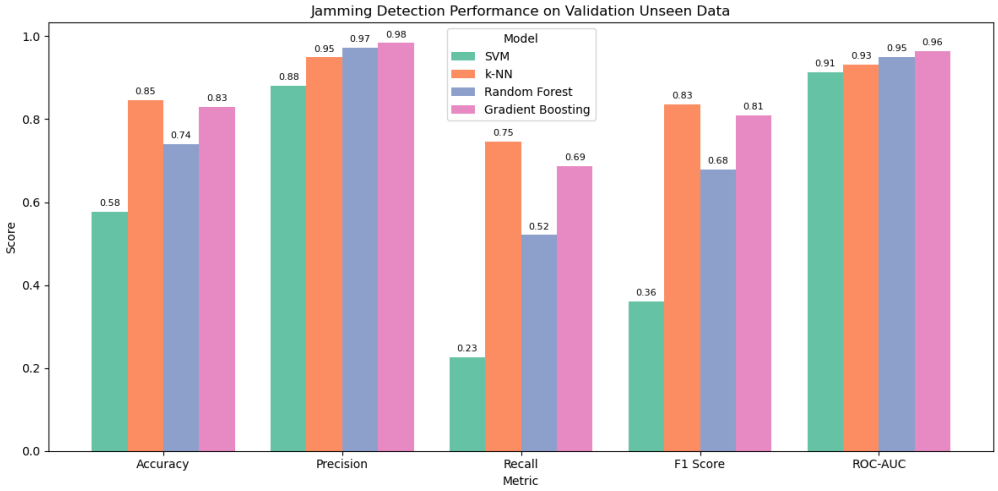

Figure 8: ML performance on unseen validation subset for gNB traces dataset

The validation process confirmed that the extensions to the network simulation environment provide a robust foundation for anomaly detection and adaptive resource management in 5G networks. By leveraging real-world testbeds and comprehensive experimental methodologies, the MLSysOps framework can address dynamic network challenges and ensure secure, reliable communication in diverse scenarios. Future work will focus on further optimising ML models and expanding the range of simulated attacks to enhance the system's versatility and robustness.

5.4 Simulator Release

Open5GS, srsRAN projects are made available under the terms of the GNU Affero General Public License. GNU Radio is released under the GPL-3.0 license. Accordingly, our extensions will follow the same open-source approach and will be available on the MLSysOps website (https://mlsysops.eu/simulators/).

6 Optical Switch Environment

6.1 Overview

The Optical Switch Network simulator is used to investigate the performance of topologies that feature Optical Switches. This new network element enables the dynamic physical topology reconfiguration of the network. The simulator allows network topology dynamic switching, which, for example, enables the formation of a full fat-tree to support the current workload, whereas, for the next workload, it can change a part of the physical network into, e.g. a dragonfly which is more appropriate. It has been very recently identified[36, 37] that the physical reconfiguration of network topologies can lead to substantial power and performance gains for ML training workloads. For this reason, the optical simulator captures many related metrics. The simulations are communication pattern-aware for large AI training jobs, like Large Language Models (LLMs) and Deep Learning Recommender Models (DLRMs). The simulated communication patterns are those of the Message Passing Interface Collective Operations[38] (e.g. all-to-all, ring-all-reduce, etc.). Given that LLMs and DLRMs tend to occupy hundreds of GPUs of a cluster, the simulator algorithms try to place each workload Ranks on the AI cluster in such a manner that the vast majority of the available links of fat-tree topology are never utilised and, therefore, are not needed for such workloads. This is the main reason that the simulator reconfigures the physical topology to provide another degree of freedom to datacenter resource schedulers, i.e. the physical link allocation.

Given that the described functionality of the datacenter network is a very recent trend, no open-source simulators have been developed that can be further modified by MLSysOps. At NVIDIA, we have developed an internal tool we call OCS solver that carries out deployment feasibility analysis of DLRM and LLMs mixed workload execution on different Optical Switch-Based network architectures.

This tool details the AI cluster resources and connectivity and then runs a series of placement algorithms for a given workload trace to determine the average resource utilisation. In MLSysOps, we build upon this solver system and extend it to include simulation capabilities useful for developing an AI model for the problem.

6.2 Extensions for MLSysOps

The OCS solver exposes the following main features, which are all tailored to the MLSysOps needs.

- **AI Cluster Resource Representation**. GPUs, Network Ports, Packets and Optical Switches are considered and modelled through JSON files.

---

[36] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. 2022. Jupiter evolving: transforming Google's datacenter network via optical circuit switches and software-defined networking. In Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22). Association for Computing Machinery, New York, NY, USA, 66–85. https://doi.org/10.1145/3544216.3544265

[37] Hong Liu, Ryohei Urata, Kevin Yasumura, Xiang Zhou, Roy Bannon, Jill Berger, Pedram Dashti, Norm Jouppi, Cedric Lam, Sheng Li, Erji Mao, Daniel Nelson, George Papen, Mukarram Tariq, and Amin Vahdat. 2023. Lightwave Fabrics: At-Scale Optical Circuit Switching for Datacenter and Machine Learning Systems. In Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23). Association for Computing Machinery, New York, NY, USA, 499–515. https://doi.org/10.1145/3603269.3604836

[38] https://www.mpi-forum.org/docs/mpi-1.1/mpi-11-html/node64.html#Node64

- **A collection of placement algorithms**. These algorithms decide placement based on current OCS link resource pressure and the traffic pattern of the arriving workload optimising against different goals (e.g. OCS uplink allocation or GPU utilisation on the same PCI-e domain, etc). Moreover, they can be used to offer ground truth for machine learning scheduling.
- **Real Cluster-in-the-simulation-loop mode**. Generates Optical Switch configuration permutations in case the system is making placement decisions for a real cluster.
-

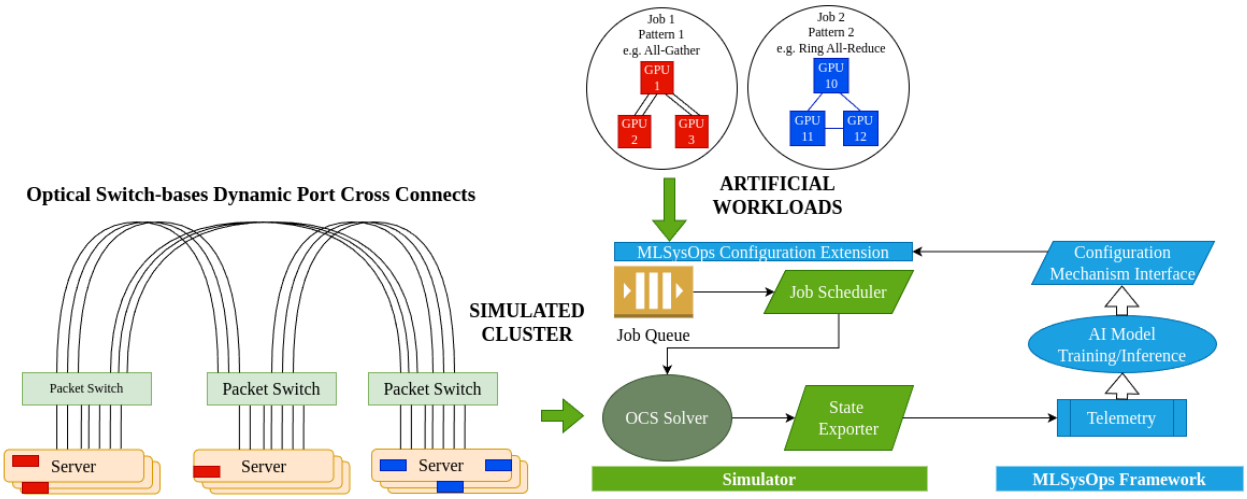

Figure 9: Simulated elements of NVIDIA's OCS solver and interaction with external systems

Figure 9 depicts a high-usage overview of the NVIDIA OCS solver. On the left are the simulated cluster resources, including the compute resources and the first (and only) layer of packet switches. The uplinks of these switches get flexibly interconnected at runtime to reflect the workload mix pattern requirements (depicted at the top of the figure). The simulator does an exhaustive search to fit the workloads on the simulated cluster without violating switch uplink resources. I MLSysOps, we seek to make well-informed decisions for job placement using AI so that the OCS-based cluster utilisation is improved in the long run. To this extent, appropriate extensions were developed in the simulator to support interaction with the MLSysOps framework (depicted on the right). Three basic extensions were developed and are described in the following paragraphs.

Artificial Workload Generator

A trace generator was developed to create artificial workloads. The workloads dispatched in the simulated cluster are very specific and exhibit little variation. Each job represents an ML training session focused on DRLM or LLM models with similar characteristics. Variability in the workloads is centered around the required resources and the duration of each job. The cluster resources are static and uniform in capabilities (e.g. all servers and GPUs are identical). By generating traces with varying sizes and duration mixes, the simulator produces the required data for training ML models across different scenarios.

State Exporter

The process of training ML models to address job scheduling problems requires accurate and detailed data collection. The simulator initially calculated aggregated metrics for the entire duration of a workload trace. This approach provided a single numerical value, such as average cluster utilisation, over a trace that could span days in the physical world. While useful for high-level analysis, this approach needs more granularity to develop sophisticated ML models.

To address this limitation, a new approach has been implemented: the simulator exports granular data representing the state of the simulated cluster during the simulation. The granularity period of the data is implicitly configured by the workload pattern that is simulated and can be therefore managed by the data analyst to follow a required distribution. The data is exported either to files or directly into the MLSysOps telemetry system using the designated interface. The generated data is subsequently utilised to explore, develop, and validate various ML models.

Job Scheduling Configuration Mechanism

The initial approach to modifying the job scheduling process focused on altering the order of job arrival in the system. It was demonstrated that changing the sequence of job arrivals in the queue caused the system to exhibit varying behaviours. A more direct approach involves enhancing the job scheduler to query the MLSysOps framework before making each scheduling decision based on the jobs currently in the queue. Both approaches aim to achieve the same outcome and are similar.

6.3 Usage Examples

Initially, the first validation step was to demonstrate the impact of using an OCS-based network compared to alternative approaches. Below, we provide the mean cluster utilisation for a mixed workload of DLRMs and LLMs that a characterisation function of NVIDIA-internal, month-long workload traces has produced. The total mean cluster utilisation of full-fledged fat-tree-based topology with leaf and spine layer packet switches Vs an OCS-based network without a spine layer of packet switches is depicted in Figure 10. The simulation is currently using fixed link allocation and workload placement algorithms.

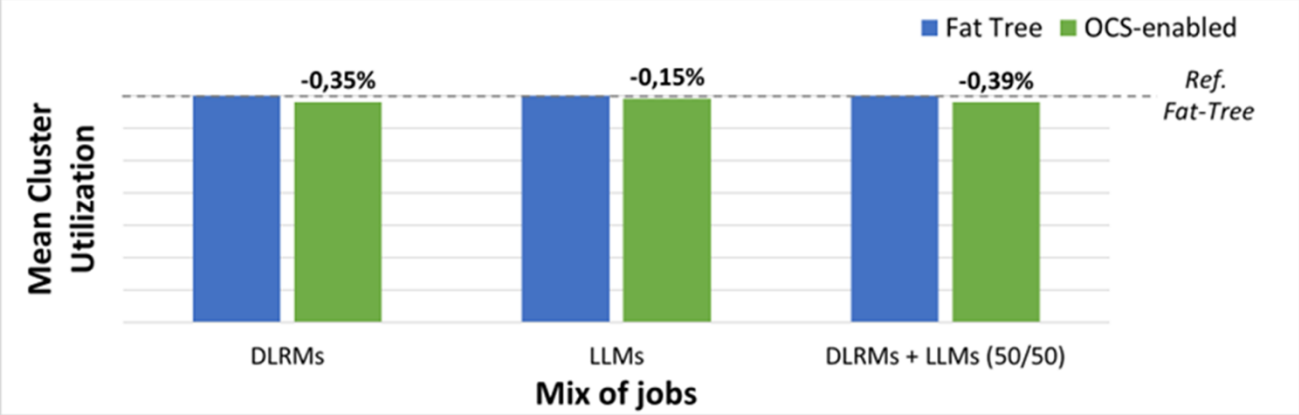

Figure 10: Mean Cluster Utilization of Fat Tree vs OCS-based networks simulation results

The export of the simulator's state and its integration with the telemetry system have been fully validated. The simulator can push metrics to the MLSysOps telemetry system, functioning similarly to any other system. The key difference lies in the simulator's event-based operation, as opposed to the time-based operation of the MLSysOps telemetry system. The simulator processes jobs primarily when new jobs arrive or finish, whereas the telemetry system operates in real-time. To address this, simulation events are interpolated into a timeline, and telemetry metrics are pushed with appropriate delays. In the simulator's context, the time spent on individual jobs is not critical; the focus is on the overall duration of the workload trace, making the event-based logic adequate. Once the data is pushed to the telemetry system, it can be exported, analysed, and utilised for ML model training.

The interaction with the job scheduling component is still a work in progress. Currently, our efforts are focused on changing the order of jobs within each window of the workload trace. A window-splitting mechanism is used, using the workload trace generator, where each window consists of five jobs to be scheduled. An ML model predicts the outcomes for all possible permutations of jobs within a window. Using these predictions, the

optimal job order is identified and configured into the trace, ensuring the most favourable performance results for each window.

6.4 Simulator Release

NVIDIA optical network simulator (NVON) is an internal company project, and work done in MLSysOps is a small part of it. Nevertheless, the part of the simulator source that can be independently used for AI cluster exploration and provides generation of workloads and the ML models developed in MLSysOPS that can be modules to the simulator will be available as an open source in the main repository of the project. At the moment of this writing, the repository is yet to be available due to an internal clearance process that is in progress, but it will be available on the MLSysOps website (https://mlsysops.eu/simulators/).

7 Conclusion

Simulators allow overcoming constraints that are imposed by the collection of real-world data and let the ML models be quickly tested in a controlled environment and a wider variety of conditions (including circumstances that are uncommon but potentially significant or scales that are impractical or even impossible to achieve using real testbeds). With these principles in mind, a selection of simulators which properly capture the salient features of the respective environments (i.e., cloud datacenter, edge node, networking, drone and optical switch) has been performed and documented in D4.1., being followed by an articulated customisation activity, extensively reported in this D4.3 document along with its appendices. Indeed, there is no "out-of-the-box" simulator specifically and simultaneously purposed for both the SysOps domain in the computing continuum, which is why each of the selected simulators is required to be customised.

All the carried-out interventions over the simulators aimed at supporting the data collection processing by introducing novel features (e.g., inter-datacenter task migration mechanisms, generation of networks anomalies, fine-grained data log production) or improving existing ones (advanced energy models, real-world mobility patterns and realistic network profiles, etc.) by directly operating on the simulator or fostering its integration with other ones (e.g., SUMO with EdgeCloudSim or Gazebo with AeroLoop). Usage examples have been provided while operative instructions about how to install, set up and use the simulators will be reported extensively on the project website at https://mlsysops.eu/simulators/.

END OF DOCUMENT