





Internet of Drones Simulator: Design, Implementation, and Performance Evaluation

Giovanni Grieco , *Graduate Student Member, IEEE*, Giovanni Iacovelli , *Graduate Student Member, IEEE*,
Pietro Boccadoro , *Member, IEEE*, Luigi Alfredo Grieco , *Senior Member, IEEE*

Abstract—The Internet of Drones (IoD) is a networking architecture that stems from the interplay between Unmanned Aerial Vehicles (UAVs) and wireless communication technologies. Networked drones can unleash disruptive scenarios in many application domains. At the same time, to really capitalize on their potential, accurate modeling techniques are required to seize the fine details that characterize the features and limitations of UAVs, wireless communications, and networking protocols. To this end, the present contribution proposes the Internet of Drones Simulator (IoD-Sim), a comprehensive and versatile open source tool that addresses the many facets of the IoD. IoD-Sim is a Network Simulator 3 (ns-3)-based simulator organized into a 3-layer stack, composed of (i) the Underlying Platform, which provides the telecommunication primitives for different standardized protocol stacks, (ii) the Core, that implements all the fundamental features of an IoD scenario, and (iii) the Simulation Development Platform, mainly composed of a set of tools that speeds up the graphical design for every possible use case. In order to prove the huge potential of this proposal, three different scenarios are presented and analyzed from both a software perspective and a telecommunications standpoint. The peculiarities of this open-source tool are of interest for researchers in academia, as they will be able to extend it to model upcoming specifications, including, but not limited to, mobile and satellite communications. Still, it will certainly be of relevance in industry to accelerate the design phase, thus reducing the time to market of IoD-based services.

Index Terms—Internet of Drones, ns-3, network, simulator.

I. INTRODUCTION

The Internet of Drones (IoD) [1] is one of the hottest research topics in telecommunications today [2]. At first, it might appear as an extension of the Internet of Things (IoT), with Unmanned Aerial Vehicles (UAVs) playing the role of smart objects able to fly. Nevertheless, in the IoD, drones are tasked with completing mission plans with multiple objectives. Since they can also fly in organized groups, namely swarms, it is worth remarking that they are made able to continuously optimize their trajectory, and coordinate among themselves. Drones are currently involved in the delivery of value-added

services in many applications, especially in Smart Cities [3]–[6], including goods delivery, environmental surveying, first-aid units in disruptive events [2], [7], and Flying Base Station in Fifth Generation (5G) & Beyond scenarios, with multiple users requesting connectivity at the same time and in the same area [2], [7]–[9]. Smart cities are among the most challenging application scenarios, with everchanging players and behavioral patterns, which makes it hard to address public safety requirements, especially at scale [10]. All this turned the IoD from a niche subject to a mainstream research topic in networking. It must be noted that the adoption of drones in industry is also a huge commercial opportunity, as testified by the several billions forecasts already available for multiple business sectors [2].

Even though several applications are now including drones, and they may look like off-the-shelf utilities, the design of complex IoD systems still requires advanced methodologies to effectively unleash the potential of services based on networked drones. In 5G & Beyond scenarios, ubiquitous connectivity and relaying capabilities are required to interact with both terrestrial entities, i.e., ground Base Stations (BSs) and users, as well as aerospace ones, such as satellites [11]. In this regard, channel capacity, available/required data rates, dedicated bandwidth and frequencies must be characterized, bearing in mind that every link may be realized with different telecommunication protocols. Moreover, given the variety of available drones on the market, an accurate suitability assessment based on their characteristics is required.

Current state of the art IoD simulators [12]–[18] do not cover all the aforementioned aspects. In light of the foregoing, the key contributions of this work are hereby introduced.

- A comprehensive open-source simulation platform, namely IoD-Sim*, is proposed in this work. IoD-Sim is able to create realistic simulations by extending the available features of ns-3 to address the relevant aspects of the IoD. Since its first release [19], it has been carefully re-designed and thoroughly refactored. The overall architecture of the proposal is designed as a 3-layer stack: (i) the *Underlying Platform*, which includes a set of technologies and libraries able to perform high-precision numerical computation, (ii) the *Core*, which embeds a set of unique IoD-related features, and (iii) the *Simulation Development Platform* that allows high-level mission design and analysis of simulation results.

Copyright (c) 2022 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

This work was partially supported by the Italian MIUR PON projects Pico&Pro (ARS01_01061), AGREED (ARS01_00254), FURTHER (ARS01_01283), RAFAEL (ARS01_00305) and by Warsaw University of Technology within IDUB programme (Contract No. 1820/29/Z01/POB2/2021).

G. Grieco, G. Iacovelli, P. Boccadoro and L.A. Grieco are with the Department of Electrical and Information Engineering, Politecnico di Bari, Bari, Italy (email: *name.surname@poliba.it*) and with the Consorzio Nazionale Interuniversitario per le Telecomunicazioni, Parma, Italy.

*https://github.com/telematics-lab/IoD_Sim

TABLE I: Summary of the comparison of the available solutions

	[12]	[13]	[14]	[15]	[16]	[17]	[18]	IoD-Sim
Open-source code				✓		✓		✓
Modularity	✓	✓	✓					✓
Scalability		✓	✓					✓
Visual scenario configuration	✓	✓				✓		✓
Network simulations		✓	✓		✓	✓	✓	✓
Support multiple networking standards		✓			✓			✓
Multi-stack protocols support	✓							✓
Graphically-assisted trajectory design	✓	✓						✓
Aerodynamics simulations	✓	✓		✓				
Power consumption models					✓			✓
Hardware in the Loop support	✓	✓						
High-level application development support		✓		✓	✓		✓	✓
Ready-to-use IoD applications								✓
Machine-readable results		✓	✓	✓				✓
Human-readable results	✓	✓	✓	✓			✓	✓

- Upon the presented architecture, radical improvements have been made in key areas, such as mission design, trajectory planning, and application configuration. Furthermore, new features are provided for hardware configuration, energy consumption models, on-board peripherals, and integration with other network entities. Moreover, a high-level mission design tool grants a welcoming user experience via a convenient interface.
- An extensive and diversified simulation campaign is carried out to validate its manifold functionalities. To this end, three scenarios are conceived to evaluate different configurations of network topologies, communication technologies, drones' equipment, and software applications. Thoughtful insights are derived by analyzing the obtained results in terms of Signal-to-Interference-plus-Noise Ratio (SINR), throughput, power consumption, latency, and Packet Loss Ratio (PLR). Moreover, a performance analysis is conducted to assess the computational load and its scalability.

The proposal guarantees ease of use to reliably simulate advanced IoD systems, with the goal of thoroughly testing new proposals and applications, especially related to their employment in densely populated urban environments.

The present contribution is structured as follows: Section II summarizes the reference state of the art of simulators in order to cover their peculiarities and motivate the proposal. Section III presents a general overview of the simulator architecture. Section IV describes the underlying platform and its design rationale. Section V discusses the core of the simulator in detail, with dedicated subsections focused on the main building blocks of the project. A thorough explanation of the involved mobility models is given together with all the supported communication technologies, and the included logical entities. Section VI focuses on the simulation design, discussing the details of helpers, thus explaining the role and importance of scenario configurations. Section VII is dedicated to the simulation campaign; after an initial focus on scenarios description, the outcomes are discussed to highlight the main findings. Finally, Section VIII concludes the work and outlines future work possibilities.

II. RELATED WORKS

To improve and speed up both the design and the prototyping phases of IoD systems, simulations are widely conceived as a useful aid. Even though simulating drones is a challenging task, it has been dealt with by many contributions so far [12]–[18]. Overall, these works approach IoD simulations from two different points of view. The first focuses on the dynamics of the flight, thus including mechanical energy and kinetics; they employ Robot Operating System (ROS) [20] and Gazebo [21] as base platforms [15], [17]. The second, instead, focuses on accurate drone networking simulations [12]–[14], [16], [18], mainly based on ns-3 [22] and OMNeT++ [23], in which UAVs are envisioned as nodes exchanging data at certain frequencies using well-known protocols belonging to wireless networks, which can either be cellular or Wi-Fi.

The contribution presented in [12] models UAVs and discusses their functionalities and possible applications. In particular, the proposal introduces FlyNetSim, a software that aims at simulating not only flight operations but also networking communication primitives and principles. The simulator can work with a group of drones operating together in a reference ecosystem. The most interesting functionalities are: (i) UAV control over Wi-Fi, (ii) multi-network communications, (iii) Device-to-Device communications for swarms, and (iv) IoT scenarios and data streaming.

In [13], instead, it is proposed CUSCUS, a simulation architecture for control systems in the context of drones' networks. The proposal is able to simulate the mechanisms for the control of drones operations and it is claimed to be highly flexible and scalable. The proposed simulator leverages the ns-3 capabilities to work with virtual interfaces simulating real time systems, eventually composed by swarms.

The contribution presented in [14] describes AVENS, which is a hybrid network simulation framework specifically designed to evaluate the performance of intelligent aerial vehicles. Here, drones are communicating using some of the most well-known communication protocols for Flying Ad-hoc NETworks (FANETs). Differently from other contributions, AVENS is focused on modeling realistic flight conditions. On top of that, it uses a layered architecture that acts as an interpreter and code generator, namely LARISSA, thanks to specified simulation parameters and settings. All the results

are obtained by the integration and interoperability with the OMNeT++ simulator.

The proposal in [15] is a simulation framework for unmanned aircraft systems traffic management. It leverages both ROS and Gazebo to implement high-level flight services. The simulator can be used for prototyping missions and controlling both rotary and fixed-wing drones flying in the same environment.

The work presented in [16] discusses a Java-based simulation framework for FANET networks and their applications. In particular, it models the coverage area of each device in the scenario. At the same time, it considers a mobility model for ground entities, i.e., humans in the operating area. Drones' characterization is herein discussed in terms of limited autonomy and battery recharging needs. To achieve this aim, an energy consumption model has been included to evaluate the footprint associated with the flight of a drone. For the sake of completeness, it must be said that this work neglects the contributions due to collision issues and consequent behaviors.

CORNET 2.0 [17] is a middleware to simulate robots in general, both in physical and networking contexts. It reaches the aim of designing a path planning solution that is simulated by Gazebo and Mininet-WiFi.

The work presented in [18] proposes a discrete-time, event-based, co-simulation scheme for networks composed by multiple drones, also configured in swarms. The simulator can carry out both flight and network simulations. This solution is of interest because there is an intrinsic codependency between the flight status and the networking operations carried out by each drone in the scenario. This contribution is of relevance because it is claimed to guarantee reliability and real-time availability thanks to the possible integration of existing simulators. This work also claims that other available simulators do not implement realistic and reliable mobility models for drones.

A comparison among the main characteristics and features of the aforementioned contributions is provided in Table I. Specifically, only the latter [18] shows some similarities with IoD-Sim. For example, both of them operate as discrete-time and event-driven simulators. Nevertheless, it is worth noting that the discrete-time operating mode of this work is motivated by the adoption of ns-3 [22] as a core network simulator. Another aspect is related to the synthetic trajectories that are implemented in IoD-Sim, that are described by closed-form mathematical expressions. Hence, in case the network simulator is substituted, the mobility models provided by IoD-Sim could be used even with continuous-time simulators.

III. ARCHITECTURAL OVERVIEW

The overall architecture of the proposed IoD-Sim simulator is depicted in Figure 1. This diagram frames the complexity and clarifies the organization of the main building layers, each providing peculiar functionalities that are depicted as blocks. The joint adoption of these components enables different simulation scenarios, which are configurable by higher-level entities.

From the bottom-up, the first layer, i.e., *Underlying Platform*, has been created to group all the necessary components

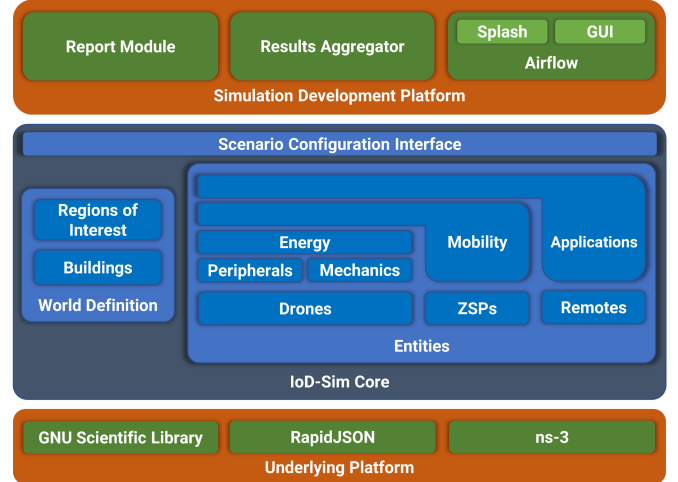


Fig. 1: Overview of IoD-Sim Architecture.

to carry out simulations. In particular, it includes (i) the GNU Scientific Library for advanced mathematical operations, (ii) the RapidJSON library which is specifically designed for high-performance parsing operations, and (iii) the ns-3 simulator engine for a robust foundation of networking facilities. A detailed description of these components is given in Section IV.

The second layer, namely *Core*, implements IoD-related simulation facilities and it is organized into three main subgroups. The *World Definition* is motivated by the fact that a realistic network simulation must be modeled, taking into account cyber-physical aspects. Hence, IoD-Sim allows to simulate the physical space in which the simulation takes place, such as *Buildings* and *Regions of Interest*. Furthermore, *Entities* details all the aspects related to drones, Zone Service Providers (ZSPs), and remote hosts, spanning from their peculiar characteristics, i.e., peripherals and mechanics for UAVs, to general ones, such as applications. Finally, the *Scenario Configuration Interface* allows the entire layer to be highly configurable thanks to the definition of a high-level language. In light of the above, the entire layer can be fine-tuned to dig as deep as possible for the use case of interest, as explained in Section V.

The third and last layer, i.e., the *Simulation Development Platform*, provides high-level configuration and data analytics facilities to the end-user. It includes *Airflow*, a high-level visual configuration environment that drastically eases the interaction between the user and the simulator, i.e., scenario set-up and management. Moreover, a *Report Module* guarantees the readability of simulation results in a clear eXtensible Markup Language (XML) schema. This module, together with the *Results Aggregator*, eases data processing with third-party tools. A detailed description of these components is given in Section VI.

It is of upmost importance to clarify that the above architecture is a multifaceted solution which applies the concept of modularity-by-design across the entire software implementation, thus proposing the peculiar functionalities described in what follows.

IV. UNDERLYING PLATFORM

The *Underlying Platform* is a foundation composed of the GNU Scientific Library (GSL), *RapidJSON*, and ns-3.

GSL is a numerical computing framework that implements numerous routines and low-level data structures, such as complex numbers, linear algebra, data analysis, and interpolation. Furthermore, it is offered in Linux-derived distributions with first-class support [24].

RapidJSON is a parser and generator of JavaScript Object Notation (JSON) code. It is one of the most adopted JSON libraries available for C++ projects. It eases the creation, traversal, validity check, and analysis of JSON codes [25]. *RapidJSON* has been chosen for its high performance and its extensive and flexible high-level Application Programming Interfaces (APIs).

Finally, ns-3 emerges as the most relevant component: it is a solid and mature discrete-time event-based network simulator. ns-3 is an open-source project that provides a solid simulation engine and various models for network design and testing. Started in 2006, it is a collection of different C++ and Python objects that implements several aspects of networking elements. The fundamental building block of ns-3 is `ns3::Node`, an abstract object which represents a generic host in a network. It can be aggregated with other objects and models, e.g., the common TCP/IP stack over Ethernet, to simulate networking behaviour. Other interesting features in ns-3 are (i) `ns3::Channel`, which simulates the communication channel between `ns3::Node` objects, (ii) `ns3::NetDevice`, which represents the node networking interface, and (iii) `ns3::Application`, which sits on top of the protocol stack to produce or consume high-level information.

Furthermore, a `ns3::Node` can be aggregated with *Mobility Models*, *Energy Consumption Models*. This possibility is not limited to those models, since the support can be extended to any other model that adds new features beyond basic networking. To this end, nodes have the potential to move in space and, hence, drain current from a `ns3::EnergySource`.

Besides, traces and probes allow to track and record simulation data in log files that are typically encoded in textual ASCII, or PCAP. In a nutshell, IoD-Sim treats ns-3 as a foundation, extending it with new features that are focused on accurate drone simulations, mobile wireless communications, energy consumption, and their integration with on-board peripherals and ground communication infrastructures.

V. CORE OF IOD-SIM

This Section presents the building blocks of the IoD-Sim Core, which is the main part of the simulator.

A simulation scenario requires the definition of a simulated world, described by Regions of Interest (RoIs) and buildings. In this world, entities, i.e., drones, ZSPs, and remotes, are simulated in a network topology defined by a set of communication models. Each drone is characterized by a mission plan defined by a set of points of interest, which in turn describes a curvilinear trajectory. Furthermore, a drone can be equipped with an energy consumption model, which relies

on a set of mechanical properties and a set of peripherals. Entities in general can host one or more communication stacks and applications. While drones and ZSPs are connected together according to the configuration of the IoD infrastructure, remotes are reachable through a backbone that simulates Internet behavior. All these blocks are configurable through an abstraction interface focused on interpreting a high-level description of the scenario encoded in JSON format.

A. World Definition

IoD-Sim offers the possibility to define parameters related to the simulated world, i.e., the environment in which the simulation takes place. The two main features are the buildings and the Regions of Interest.

The virtual world in IoD-Sim is a theoretically infinite space. The space can be filled with entities, which could be Drones, ZSPs and Remotes, but also with RoIs and *Buildings*.

1) *Buildings*: The virtual world can be enriched with obstacles, i.e., *Buildings*. They are used to represent urban scenarios, thus making simulations that are particularly suitable for research in *Smart Cities*. IoD-Sim provides an abstraction layer to configure and place buildings in the virtual world, relying on `ns3::BuildingsHelper` and `ns3::Building` objects. A `ns3::Building` is a collisionless 3D object with the following properties:

- boundaries, which defines the box dimension in the space. Boundaries can be defined by an array of two points organized as $[P_x^{(1)}, P_x^{(2)}, P_y^{(1)}, P_y^{(2)}, P_z^{(1)}, P_z^{(2)}]$. A representation of these two points is given in Figure 2.
- type of building, which can be either commercial, residential, or office.
- type of walls material, which can be wood, concreteWithWindows, concreteWithoutWindows, and stoneBlocks.
- number of floors.
- number of rooms along the x and y axis, per floor. The rooms are placed in a grid position.

Such a feature is important for Long-Term Evolution (LTE) communication fading, which varies according to the characteristics of each building.

2) *Regions of Interest*: A `ns3::InterestRegion` is a 3D box placed on the simulated world defined, as for buildings, by a vector of two points. Throughout the simulation, it is possible to retrieve and to update the current set of coordinates with `GetCoordinates()` and `SetCoordinates()` methods, respectively.

The whole set of these areas is managed by `ns3::InterestRegionContainer`, which helps to create RoIs and group them. This utility object provides a (i) `Create()` method to generate and index RoIs, (ii) `GetN()` to report the number of created regions, (iii) `GetRoi()` to retrieve the i^{th} , and (iv) `Begin()` and `End()` iterators to traverse the entire container. Moreover, the `InterestRegionContainer::IsInRegions()` method acknowledges the presence of a drone in multiple areas, thus granting the possibility to trigger specific events during the simulation. For instance, *Drone* operations can be

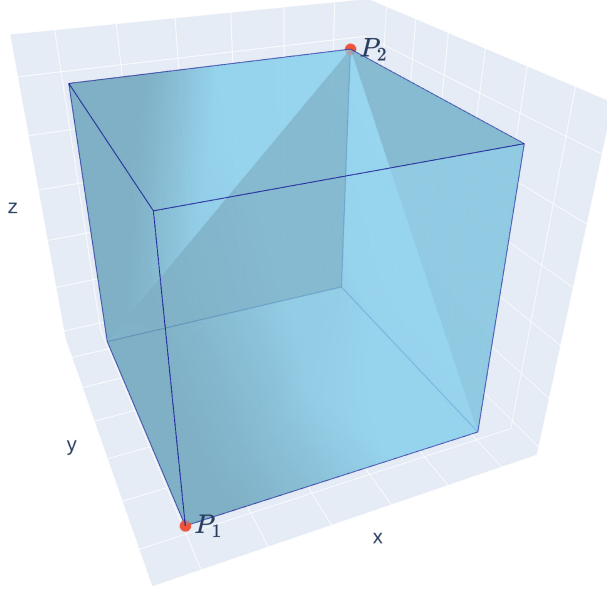


Fig. 2: Example of box placement with two points, P_1 and P_2 , in order to create a Building or a RoI in the simulated world.

TABLE II: `ns3::Drone` properties in IoD-Sim.

Name	Unit of Measurement
Mass	kg
Rotor Disk Area	m ²
Drag Coefficient	(dimensionless)
Peripherals	
Weight Force	N
Air Density	kg/m ³

restricted to a limited space, leading to an optimization of *Drone* power consumption.

B. Drones

IoD-Sim provides `ns3::Node` derivatives to consider the characteristics of key actors commonly found in an IoD simulation. The `ns3::Drone` class characterizes a rotary-wing UAV and it is registered as a new `TypeId` in ns-3, along with its mechanical properties, shown in Table II. While the first four properties can be defined by the user, the last two are a direct consequence of the given characterization. `ns3::Drone` properties can be set by means of ns-3 attributes or by its public object interface. Its mass can be updated at any time by means of `SetMass()`. Upon update, the drone weight force is also updated in cascade by multiplying the new mass with the constant gravity acceleration. The rotor disk area and its drag coefficient can be updated in the same manner by means of `SetArea()` and `SetDragCoefficient()` methods, respectively. Furthermore, `ns3::Drone` properties can always be read any time during the simulation through ns-3 attributes and object getters, such as `GetMass()`, `GetWeight()`, `GetArea()`, and `GetDragCoefficient()`.

Drones can be grouped together in `ns3::DroneContainer` and can be statically referenced by their unique identifier in the simulation through `ns3::DroneList`.

According to the peculiar workflow of ns-3, to properly instantiate a `ns3::Drone` object, a `ns3::DroneContainer` is needed. The creation process consists of a call to the `ns3::Object::CreateObject<T>` function, where `T` is replaced with `Drone`. In order to ensure full compatibility with all ns-3 methods involving `ns3::Node` or `ns3::NodeContainer` classes, a dedicated mechanism has been developed. Every `ns3::Drone` goes through a static cast procedure, i.e. `ns3::StaticCast`, which generates a `ns3::Node` object that is pushed into a `ns3::NodeContainer`. In this way, for each drone, two smart pointers refer to the same memory location but cast to the two required types. Besides, the `ns3::DroneContainer` class provides a specific iterator, together with two further methods which return the number of instantiated drones and a smart pointer to each. It is worth mentioning that only drones must use a `ns3::DroneContainer`, while ZSPs, together with other entities, must still be modeled as `ns3::Node` objects.

1) *Peripherals*: A UAV is usually equipped with a set of peripherals able to extend its capabilities. Such peripherals include a wide range of devices, implemented in IoD-Sim through new specific classes. The `ns3::DronePeripheral` object represents a general-purpose on-board peripheral with the following properties:

- *Peripheral state* – which can either be set to ON, OFF, or IDLE. This simple Finite State Machine (FSM) allows the development of intelligent algorithms to find optimal energy management.
- *Power consumption* – how much instantaneous power is required by the peripheral, expressed in Watts, for each state.
- *Reference RoIs* – where the peripheral should be operating. This is extremely useful for modeling certain peripherals and missions that depend on particular regions in space. For instance, a photo camera can be used and activated only when the drone is in the RoI, thus leading to an optimized use of power, storage, and data. If this parameter is not defined, the reference peripheral will be active over time.

`ns3::DronePeripheral` has been specialized in two sub-classes.

`ns3::StoragePeripheral` represents a generic storage device characterized by an attribute describing the initial amount of memory, which can be traced at runtime to record the empty space left. Device total capacity can be queried through `GetCapacity()` method. If a drone peripheral, e.g., a camera or any other sensor, wants to interact with the storage, it is possible to request space by specifying the amount of data through `Alloc()`. The inverse can be done with `Free()`. These operations can fail if there is no memory left or there are no data to be freed, respectively. For this reason, a boolean value is returned by these methods to indicate if the requested

operation was successfully completed or not. In this work, it is assumed that at most one `ns3::StoragePeripheral` is installed on each drone.

`ns3::InputPeripheral` describes a generic input device, characterized by an acquisition `DataRate`, constant over a `DataAcquisitionTimeInterval`. Once it is created, installed on a drone, and attached to a particular storage peripheral with `Install()` method, the storage peripheral of reference can be changed with `SetStorage()`. If the peripheral is ON, `AcquireData()` simulates data acquisition at the given `DataRate`.

These two peripheral types are strongly connected since a `ns3::InputPeripheral` can offload acquired data to a `ns3::StoragePeripheral` through a boolean attribute. Nonetheless, the association between input and storage is not mandatory. In fact, in a real-world scenario, an `ns3::InputPeripheral` can deliver data directly to a processing unit or to a remote host, thus neglecting the need to permanently store the information.

A complete list of the attributes of these classes is given in Table III. It is worth specifying that all peripherals hold a reference to the drone they are equipped with.

Moreover, for each `ns3::Drone`, a `ns3::DronePeripheralContainer` object is created to manage all its peripherals. This container is responsible for the creation of peripherals and, through the `ns3::DronePeripheralContainer::InstallAll()` method, sets the correct references to the host drone, and, eventually, to the target `ns3::StoragePeripheral`.

2) *Mechanics and Energy Consumption*: ns-3 already models and manages all the energy-related aspects, such as consumption, harvesting, and monitoring, through the abstract class `ns3::EnergySource`. Although there is no specific energy source model available that is suitable for drones, the `ns3::LiIonEnergySource` is sufficiently general to be employed for simulation purposes [26], [27].

The `ns3::DeviceEnergyModel` class describes the `ns3::NetDevice` energy consumption by means of the drawn current. The installation procedure is eased by the helper class `ns3::DeviceEnergyModelHelper`, which employs the `Install()` method that links a `ns3::EnergySource` to a `ns3::NetDevice`.

When the battery object is initialized, it schedules an `ns3::Event`, which calls `ns3::EnergySource::CalculateTotalCurrent()`. This function retrieves the current drawn of every device associated with the `ns3::EnergySource`, by calling `ns3::DeviceEnergyModel::GetCurrentA()`. Subsequently, the energy consumption value is calculated and subtracted from the remaining one. Finally, the `ns3::Event` reschedules itself.

In this work a specialization of `ns3::DeviceEnergyModel`, i.e. `ns3::DroneEnergyModel`, is developed along with the helper class `ns3::DroneEnergyModelHelper`. Given a simulation duration T , the model splits it into $n = 1, \dots, N$ equal discrete intervals. The power consumption model of the

drone flying at speed $\mathbf{v}[n] = (v_x[n], v_y[n], v_z[n])$, in the n -th time slot, is the following [28]:

$$P_{UAV}[n] = P_{level}[n] + P_{vertical}[n] + P_{drag}[n], \quad (1)$$

where

$$P_{level}[n] = \frac{W^2}{\sqrt{2}\rho A} \frac{1}{\sqrt{\Omega + \sqrt{\Omega^2 + 4V_h^4}}}, \quad (2)$$

being

$$\Omega = \|(v_x[n], v_y[n])\|^2 \quad (3)$$

$$P_{vertical}[n] = Wv_z[n], \quad (4)$$

$$P_{drag}[n] = \frac{1}{8} C_{D0} \rho A \|(v_x[n], v_y[n])\|^3, \quad (5)$$

$W = mg$, with m defining the mass of the drone and g as the gravitational acceleration. Moreover, ρ is the air density, A is the total rotor disk area, C_{D0} is the profile drag coefficient depending on the geometry of the rotor blades, and $V_h = \sqrt{\frac{W}{2\rho A}}$ uses parameters to calculate the power required for hovering operations.

The energy model can be aggregated to a drone by means of the `ns3::DroneEnergyModelHelper`, which provides an `Install()` method that aggregates it to `ns3::Drone`. In this way, it is possible to simulate the energy characteristics of a drone, both for its mechanics and its peripherals, in addition to its networking operations.

Such mechanical power consumption model is implemented in the method `ns3::DroneEnergyModel::GetPower()`.

Similarly, the method `ns3::DroneEnergyModel::GetPeripheralsPowerConsumption()` returns the cumulative power consumption of all peripherals on board.

The `ns3::DroneEnergyModel` object, registered as a new `ns3::TypeId` with no attributes, implements `ns3::DoGetCurrentA()` inherited from `ns3::DeviceEnergyModel`. Such method returns the total drawn current related to both mechanics and peripherals, in addition to networking operations. The energy model can be aggregated to a drone by means of `DroneEnergyModelHelper`, which provides an `Install()` method that aggregates it to `ns3::Drone`.

It is worth specifying that `ns3::DroneEnergyModelHelper` implements the installation procedure in a different manner with respect to its parent, i.e., `ns3::DeviceEnergyModelHelper`. In fact, the `ns3::DroneEnergyModelHelper::Install()` method links a `ns3::EnergySource` to a `ns3::Drone` instead of a `ns3::NetDevice`. This aspect distinguishes the aim of IoD-Sim from the ns-3 one: to simulate all the relevant aspects of the drone, beyond the networking perspective. This justifies the implementation divergence from the ns-3 main goals.

During the simulation, it is possible that the drone runs out of energy. To this end, the event is propagated through the execution of `HandleEnergyDepletion()` of the energy model, for which the time of depletion is logged for successive data analysis.

TABLE III: Drone Peripherals Properties.

Class	Attribute	Description
DronePeripheral	PowerConsumption	Power consumption of the peripheral in J/s
StoragePeripheral	Capacity	The capacity of the disk in bit
	DataRate	The acquisition data rate of the peripheral in bit
	InitialRemainingCapacity	The starting remaining capacity in bit
InputPeripheral	DataAcquisitionTimeInterval	The time interval occurring between any data acquisition
	HasStorage	Acquired data are offloaded to the StoragePeripheral

C. Other Simulation Entities: ZSPs and Remotes

Entities beyond `ns3::Drone` are *ZSPs* and *Remotes*. *ZSPs* are smart entities, modeled as `ns3::Node` objects, equipped with multiple `ns3::NetDevice` which provide multi-protocol radio access, thus enabling communications between drones and the rest of the Internet. Typically, they are configured as ground entities that maintain a constant position in time [1], by means of `ns3::ConstantPositionMobilityModel`. Nonetheless, in IoD-Sim, their mobility model can be customized to fit simulation purposes, envisioning the adoption of dynamic wireless infrastructure proposed in 5G & Beyond architectures. *Remotes*, instead, are `ns3::Node` objects with no mobility model and only rely on installed applications which provide remote services to consumers. Remotes and ZSPs are interconnected through a backbone, simplified as a Carrier Sense Multiple Access (CSMA)-based bus network, that represents the *Internet*. This architecture allows service provisioning on different classes of nodes, employing *Remotes* in the case of applications with high computational costs, e.g., multimedia data processing, and *ZSPs* in the case of low latency requirements, e.g., traffic management.

D. Mobility

`ns-3` provides a basic foundation to represent the movement of drones (e.g., `ns3::WaypointMobilityModel`, `ns3::ConstantAccelerationMobilityModel`, and `ns3::ConstantVelocityMobilityModel`). However, an important gap arises when such models are analysed in details: none of the available ones are able to construct a curve trajectory that take into account how much a spot is relevant for the mission plan. Another aspect to consider is that models such as `ns3::WaypointMobilityModel` couples the position of the drone with a given time instant, without taking into account the limitations imposed by the maximum speed of the UAV. Therefore, if the user does not properly design the path, this could lead to a simulation which does not reflect the reality. Moreover, in the setup phase it is necessary to specify all the points that create the trajectory.

To overcome these limitations, dedicated mobility models have been developed. In particular, the trajectory has been modeled using Bézier curves by specifying a set of Points of Interest (PpoI). These are decoupled from the time of arrival, and the resulting trajectory is bounded to the mechanical characteristics of the drone. A specific structure implemented in IoD-Sim, namely `ns3::CurvePoint`, describes the 3D position vector of the Bézier curve together with the distances from the previous point and the starting one. Besides, a

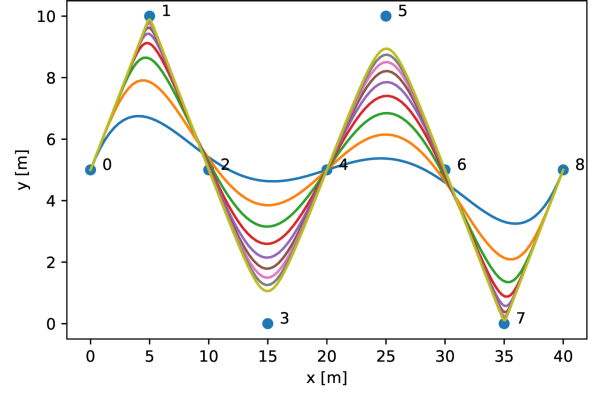


Fig. 3: A set of trajectories, generated with (6), with different Interest Levels (from 1 to 10, incrementally) for PpoI 1, 3, 5, and 7. The other points have constant Interest Level set to 1.

container object, i.e., `ns3::Curve`, is in charge of managing the points of the curve, i.e., `ns3::CurvePoint`, that are defined according to the interest points contained in a `ns3::FlightPlan`. When a `ns3::Curve` is instantiated, it populates the container according to the following.

Let $\mathbf{P} = \{\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_{N-1}\}$ with $\mathbf{P}_i \in \mathbb{R}^3$, $\forall i = 0, \dots, N-1$ be an ordered sequence of N interest points, $\mathbf{l} = \{l_0, l_1, \dots, l_{N-1}\}$, $l_i \in \mathbb{N}^+$, the interest level associated to each point, $\Lambda = \left(\sum_{i=0}^{N-1} l_i\right) - 1$ and $L_i = \sum_{h=0}^{i-1} l_h$. The *Trajectory Generator* can be expressed as

$$\mathbf{G}(t) = \sum_{i=0}^{N-1} \mathbf{P}_i \sum_{j=0}^{l_i-1} \binom{\Lambda}{L_i + j} (1-t)^{\Lambda-L_i-j} t^{L_i+j}, \quad t \in [0, 1] \quad (6)$$

It is worth noting that (6) is a revised version of the original Bézier equation, which does not practically allow to reach the interest points, except for the first and last one. An increment in the interest level l turns into a trajectory that passes closer to that point, as illustrated in Figure 3. A special case takes place when $l = 0$. A specific mechanism is provided to split the trajectory into two contiguous curves so that the drone is forced to fly over them. In this case, a `restTime` can be defined to set the hovering duration in seconds.

Finally, the obtained trajectory is used by the new implemented models, i.e., `ns3::ConstantAccelerationDroneMobilityModel` and `ns3::ParametricSpeedDroneMobilityModel`.

1) *Constant Acceleration Drone Mobility Model*: This mobility model employs (6) and the uniform acceleration motion

TABLE IV: ns3::ConstantAccelerationDrone MobilityModel TypeId attributes.

Attribute	Description
Acceleration	Drone's constant acceleration, expressed in m/s^2 .
MaxSpeed	Drone's maximum speed, expressed in m/s .
FlightPlan	Interest points for the trajectory.
SimulationDuration	Simulation duration, expressed in seconds.
CurveStep	Discretization step of the curve.

TABLE V: ns3::ParametricSpeedDrone MobilityModel TypeId attributes.

Attribute	Description
SpeedCoefficients	The set of coefficients for the polynomial $v(t)$.
FlightPlan	Interest points of the trajectory.
SimulationDuration	Simulation duration, expressed in seconds.
CurveStep	Discretization step of the curve.

law to retrieve the points of the desired trajectory. Since the speed of the drone cannot increase indefinitely, after the maximum speed is reached, the uniform linear motion law is adopted. This object is implemented as an ns-3 model, and hence, has its own TypeId with attributes described in Table IV.

In each instant of the simulation, IoD-Sim calls two methods, DoGetPosition () and DoGetVelocity (). They return both the position and the speed at current time of the drone, that is recomputed thanks to the Update () method.

2) *Parametric Speed Drone Mobility Model*: Similarly to *Constant Acceleration Drone Mobility Model*, this mobility model is implemented as a ns-3 model with its own TypeId. However, this takes a $v(t)$ speed profile in a polynomial form and, thanks to the modified Bézier equation (6), it retrieves the discretized trajectory. To ease the implementation, a specific attribute, i.e., ns3::SpeedCoefficients, is introduced to serve as a container of the $v(t)$ coefficients. These are elaborated (by employing the GSL) to constantly update the parameters by calling UpdateSpeed () and UpdatePosition () subroutines. A summary of the attributes of this mobility model is reported in Table V.

E. Applications

IoD-Sim offers simple applications that can be used to communicate telemetry from a drone to a ZSP or to a Remote by adopting client-server paradigm, via User Datagram Protocol (UDP). Moreover, relying on the same architecture, two Transmission Control Protocol (TCP)-based applications are available to enable reliable data transfer between hosts. Besides, a Network Address Translation (NAT)-like application is provided to design relaying network architectures.

1) *Telemetry Applications*: These applications are modeled as classes named ns3::DroneClientApplication and ns3::DroneServerApplication. The model asks for the DestinationIpv4Address and a Port of the remote entity that hosts the server application. Data are sent

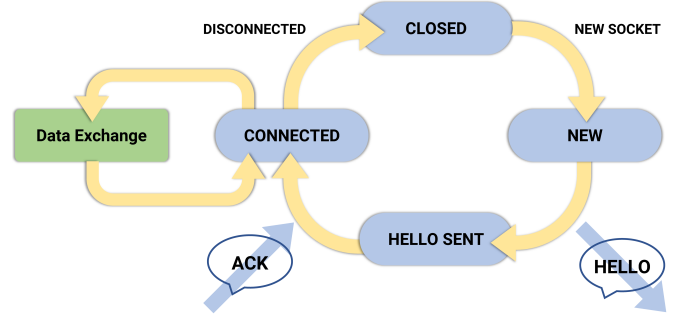


Fig. 4: FSM of the Drone Client and Server Application.

every TransmissionInterval and, whereas the drone has a storage peripheral, it is possible to free an equivalent amount of memory space. The configuration parameters are summarized in Table VI.

When the application is started, through the ns3::Application::StartApplication () method, a UDP-based communication, employing application-level acknowledgements, takes place. It is worth specifying that the application is stateful in order to support the *Rendezvous Process* which discovers the application server in the network, if no address is given. This process starts with the client application in NEW state. Therefore, a HELLO packet is sent to the destination address (or in broadcast), thus implying a state transition in HELLO_SENT. If the application server receives such packet, it replies with an HELLO_ACK packet to confirm the reception. When the client receives the acknowledgement, its state changes again, into CONNECTED, which allows it to periodically send telemetry data. These packets are named UPDATE and UPDATE_ACK. The entire procedure is depicted in Figure 4.

The JSON-encoded telemetry is periodically transmitted, through the SendPacket () method, and received by the application server, through the ReceivePacket () method. HELLO and UPDATE packets transport a payload which is formatted in JSON with ASCII encoding. Its content is a JSON object with the following properties:

- The unique id of the drone in the simulation. This ensures that *Drones* communications can be tracked over complex scenarios.
- An incremental Ssn that refers to the *sequence number*. It is used to easily check if a packet has been lost.
- cmd that refers to the type of packet, whether if HELLO, UPDATE, or an acknowledgment.
- gps coordinates with lat for latitude, lon for longitude, alt for altitude, and vel for the velocity vector. For simulated drones, the GPS location refers to the virtual world coordinates.

The UDP packet payload is summarized in Table VIII. When the application is stopped, the StopApplication () method is called.

Clearly, these applications are developed so that multiple instances can run concurrently on the same entity if different ports are specified. Moreover, they are independent of the particular communication technology adopted.

TABLE VI: Configuration parameters for *Telemetry Applications*.

Application Type	Name	Type	Default Value	Description
Client	DestinationIpv4Address	String	255.255.255.255	IPv4 address of the remote application server.
Client and Server	Port	Unsigned Integer 32-bit	80	Port of the remote application server or listening port in the case of the server.
Client	TransmissionInterval	Double	1.0	Transmission interval of the telemetry updates being sent, in seconds.
Client and Server	StartTime	Double	Start of Simulation	Time at which to start the application, in seconds.
Client and Server	StopTime	Double	End of Simulation	Time at which to stop the application, in seconds.
Client	FreeData	Boolean	false	Free data from the equipped storage peripheral when they are transmitted.
Server	StoreData	Boolean	false	Store data to the equipped storage peripheral when they are received.

TABLE VII: Configuration parameters for *Generic Traffic Applications*.

Application Type	Name	Type	Default Value	Description
All Server and Clients	Address	String	127.0.0.1	Listening or remote address of the server.
All Server and Clients	Port	Unsigned Integer 16-bit	4242	Listening or remote port of the server.
All Clients	PayloadSize	Unsigned Integer 16-bit	65470	Size of the payload for each packet, in bytes. in the case of Storage Client, it is the maximum size to be used when freeing storage memory.
Periodic Client only	Frequency	Double	1.0	Number of times in a second when a new packet is sent to the server.

TABLE VIII: UDP payload.

Field Name	Data Type	Description
id	Unsigned Integer 32-bit	ns-3 Global Node Identifier
sn	Unsigned Integer 32-bit	Packet Sequence Number
cmd	String	Packet Type
gps	Object	Drone location in space
lat	Double	Drone latitude
lon	Double	Drone longitude
alt	Double	Drone altitude
vel	Array of 32-bit Integers	Drone velocity in m/s

2) *Generic Traffic Applications*: These applications model a reliable data transfer between a client and a server, which are implemented as `TcpPeriodicClientApplication` and `TcpEchoServerApplication` objects, respectively. The aim is to transfer a certain amount of information between the two hosts according to the specified `PayloadSize`, expressed in bytes, and `TransmissionFrequency`, measured in Hz, set on the client. The server is characterized by a socket, composed by a listening `Address` and `Port`. These configuration parameters are summarized in Table VII. To facilitate traffic analysis, each packet has a Protocol Data Unit, formed by a 12 bytes header, and the payload. The former contains information-level sequence number and the timestamp of creation; the latter is characterized by a recurring sequence of 16 bits that is incremented over time. These applications provide dedicated `TraceSource` objects that notify communication-related events such as new/closed connections and sent/received packets.

An additional TCP-based client has been created to support drones that are typically equipped with a `StoragePeripheral`. To this end,

`TcpStorageClientApplication` monitors the storage and, if memory is used, it transfers data to the remote server. If the transfer is acknowledged, memory is freed. This mechanism is relevant when drones are equipped with limited on-board memory. Indeed, the client can be used to transfer as much data as possible over the wireless medium to prevent out-of-memory events.

3) *Relaying Application*: The *Relaying Application* is implemented through the class `ns3::NatApplication`. It is a specialized networking application that, given an `InternalNetDeviceId` and an `ExternalNetDeviceId`, provides a NAT-like mechanism to a set of drones placed in an internal network. The `NetDeviceId` is a numerical identifier that uniquely points to a network device mounted on the drone.

During initialization, i.e., `DoInitialize()` method, the application modifies the static routing table of the internal network device to redirect all traffic to the loopback device. A specific callback, namely `RecvPktFromNetDev()`, notifies when a new frame arrives. It contains information such as the Internet Assigned Numbers Authority standard L3 protocol identifier and the sender/receiver MAC addresses.

The NAT forwarding behavior leverages a hash map, i.e., *NAT Table*, where an external port number is coupled with the source IP address and port. Inbound frames are forwarded to the external network by replacing this information with the one of the relaying drone. The same rationale is applied for frames received from the external network.

F. Scenario Configuration Interface

The *Scenario Configuration Interface* is an abstraction layer that allows the configuration of the entire simulation by means

TABLE IX: Memory organization of protocol stacks in the General Purpose Scenario.

		std::array		
std::vector	Stack ID \ Layer	PHY	MAC	NET
	0	WifiPhy	WifiMac	IPv4
	1	WifiPhy	WifiMac	IPv6
	2	LtePhy	LteMac	IPv4
	:	:	:	:
	:	:	:	:

of JSON files. Indeed, they can be decoded and validated through RapidJSON in order to setup the simulation models. The output data classes are then used by the *General Purpose Scenario* to initialize objects that define the environment, the entities, and the simulator engine. To this end, the set of all objects that are used to characterize a scenario can be grouped into three categories:

- *Configuration Objects* – Models that store parameters in a structured way, easily accessible in the C++ language.
- *Configuration Helpers* – Checkers and decoders with the goal to produce a Configuration object or throw an error message.
- *Simulation Helpers* – Objects that help organise pointers to structures commonly found in scenario development. They are used in the protocol stack matrix, shown in Table IX.

Additionally, *Factory Helpers* are defined as weakly-coupled extensions to ns-3 internal data structures to ease their initialisation. They are made to minimise modifications made to the ns-3 core framework, which is used by IoD-Sim. The entire system has been made extensible by design, so that it is possible to support further technologies and configurations with the addition of new configuration objects and helpers as needed. In this way, it is possible to further develop high-level configuration interfaces able to setup scenarios and hence to ease the design activity undertaken by the user.

1) Scenario Configuration Objects and Helpers:

The core of the abstraction layer is the `ns3::ScenarioConfigurationHelper`, a low-level object that directly deals with the JSON configuration file. This helper returns a set of specific data classes that contain exclusively the parameters required to configure IoD-Sim models. Each of them is also loosely coupled with a JSON validator and parser, also known as configuration helpers. The information embedded in these classes is then deserialized and employed by higher-level objects.

- `ns3::ModelConfiguration` describes `ns3::TypeId` objects through key-value pairs that reference the model attributes.
- `ns3::EntityConfiguration` describes an entity, whether it is a *Drone*, a *ZSP*, or a *Remote*. The object retrieves and stores all parameters related to the `ns3::NetDevice` to be installed on the entity, the *Mobility Model* to be applied, and the *Applications*. Optionally, if the entity is a *Drone* there can be defined the mechanics, the battery, and the peripherals. Its parser is called `ns3::EntityConfigurationHelper`.

- `ns3::RemoteConfiguration` denotes key characteristics of *Remotes*. Specifically, a remote needs to know the global network layer ID of reference and the configurations of applications to be installed. Its parser is `ns3::RemoteConfigurationHelper`.
- `ns3::PhyLayerConfiguration` defines the required parameters needed to configure a PHY layer. It is the parent and interface of `ns3::LtePhyLayerConfiguration` and `ns3::WifiPhyLayerConfiguration` data classes. Its parser is `ns3::PhyLayerConfigurationHelper`.
- The `ns3::LtePhyLayerConfiguration` gets all the information needed to set up a PHY layer for LTE, such as its propagation loss model and its spectrum model.
- `ns3::WifiPhyLayerConfiguration` sets up the PHY layer of a Wi-Fi based protocol stack. The PHY layer configuration requires the higher-level Wi-Fi standard to be used, the antenna Rx gain, the data rate, the propagation delay and loss models.
- `ns3::MacLayerConfiguration` collects the required parameters needed to configure a MAC Layer. It is the parent and interface of `ns3::WifiMacLayerConfiguration`. Its parser is `ns3::MacLayerConfigurationHelper`.
- `ns3::WifiMacLayerConfiguration` configures a Wi-Fi Basic Service Set (BSS). The Service Set Identifier (SSID) and access point parameters are defined to create its basic infrastructure.
- `ns3::NetworkLayerConfiguration` defines the required parameters needed to configure the appropriate network layer. It is parent to the `ns3::Ipv4NetworkLayerConfiguration`. Its parser is named `ns3::NetworkConfigurationHelper`.
- `ns3::Ipv4NetworkLayerConfiguration` stores the network address and mask of the configured IPv4 Layer in the configuration file.
- `ns3::LteBearerConfiguration` decodes all the relevant parameters for an LTE bearer, such as its type and the Quality of Service (QoS) defined as a tuple of *Guaranteed Bit Rate* and *Maximum Bit Rate*.
- `ns3::LteNetdeviceConfiguration` collects the information needed by an LTE network device, such as its bearers. The *role* of the network device is then detected, whether it is a User Equipment or an eNB.
- `ns3::NetdeviceConfiguration` defines for a generic network device. The main parameter stored is the global network layer ID, which is used to detect the stack and network to be attached when the network device is created and installed on a *Node*. A specific configuration for Wi-Fi network devices is handled by `ns3::WifiNetdeviceConfiguration` with relevant MAC data to connect to the BSS.

2) *Scenario Simulation Helpers*: To enable complex scenarios that are related to the future IoD communication

paradigms, IoD-Sim enables the simulation of IoD networks in which multiple telecommunication protocols are used at the same time, both for the drones and the ZSPs. Currently, IoD-Sim supports two communication technologies that can be used concurrently: LTE and the IEEE 802.11 family. Each protocol stack must be applied to a dedicated network device, i.e., `ns3::NetDevice`. The architecture of the simulator has been designed so that it eases the configuration phase.

In order to facilitate the implementation and the installation of protocol stacks on IoD entities, additional helpers named *Simulation Helpers* have been developed to arrange the necessary common infrastructure to simulate communications among nodes. Thus, the developed *Simulation Helpers* are:

- `ns3::WifiPhySimulationHelper`, that initializes the PHY layer of a Wi-Fi-based protocol stack.
- `ns3::WifiMacSimulationHelper`, that creates the objects related to IEEE 802.11 MAC.
- `ns3::LtePhySimulationHelper`, that allocates the necessary resources to enable LTE communications.
- `ns3::Ipv4SimulationHelper`, that manages IPv4 networks for each protocol stack.

All the aforementioned can cooperate with the existing helpers in ns-3, such as `ns3::LteHelper`, `ns3::WifiHelper`, `ns3::YansWifiPhyHelper`, and `ns3::WifiMacHelper`.

3) *General Purpose Scenario*: A flexible and highly dynamic *General Purpose Scenario* has been developed in order to setup scenario's entities and, at the same time, to provide abstractions which minimize the effort from a programming perspective. It is fully dependent on a semantic analyzer and allows the entire simulation platform to be compiled beforehand, providing ways to dynamically reconfigure the scenario at run-time. Its development started from the analysis and the detection of a common structure typically followed by the Open Systems Interconnection protocol stack. The entire workflow, depicted in Figure 5, is described hereby.

General Purpose Scenario is composed of two main parts: *configuration* and *run*. Scenario *configuration*, executed through the constructor `Scenario()`, is interleaved with the *Scenario Configuration Interface*. The *run* part is identified by `operator()()` which is characterized by minimal C++ code that starts the ns-3 simulator engine. Moreover, it shows the progress status on the console and, optionally, it saves messages to a log file.

The *General Purpose Scenario* requires the initialization of the *Scenario Configuration Interface* through a JSON configuration file. Once the file is decoded, the number of entities are retrieved to create the initial structures, such as a `ns3::DroneContainer` and four `ns3::NodeContainer` objects. They keep track of ZSPs, *Remotes*, and nodes that participate in the *Backbone Network*.

Once the entities are created, they are registered to their respective global lists, such as `ns3::DroneList`, `ns3::ZspList`, and `ns3::RemoteList`.

After entity creation, the ns-3 static configuration parameters are applied to the simulation. The method is called `ApplyStaticConfig()`. These parameters are a set of key-value pairs that represent certain features of ns-3 models.

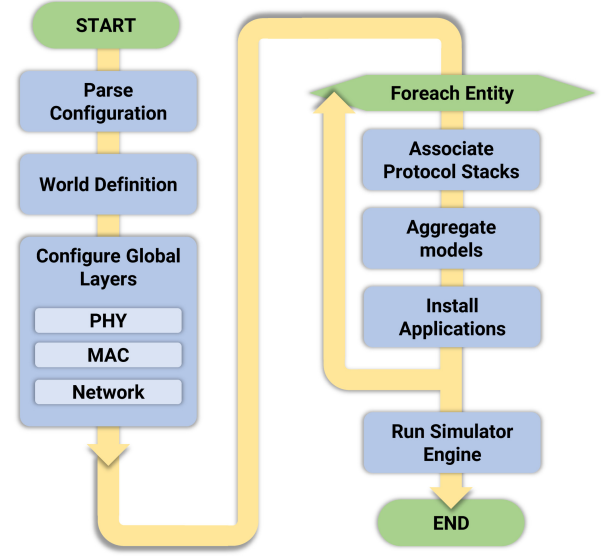


Fig. 5: Logical flow to initialize and configure a scenario in IoD Sim.

World definition is made through `ConfigureWorld()` method. It is related to the configuration of *Buildings* and *RoIs*. The virtual world set up is then followed by the configuration of PHY, MAC, and Network global layers.

As for the PHY layer part, if it is made for a Wi-Fi communication stack, the `ns3::WifiPhySimulationHelper` is employed with the specifications stored in `ns3::WifiPhyLayerConfiguration`. If the PHY layer is for LTE, instead, the `ns3::LtePhySimulationHelper` is set up with `ns3::LtePhyLayerConfiguration` parameters. The same procedure is applied for the global MAC layer configuration. The global *Network* layer is managed by `ns3::Ipv4SimulationHelper` for IPv4 networks with the specifications given by `ns3::NetworkLayerConfiguration`, i.e., network address, mask, and a default route.

Global stacks are then linked to the configured entities. Moreover, for LTE devices, the bearer is created to ensure that applications have a logical communication channel with desired properties. When the entity network configuration is done, the mobility model is configured and the applications are installed. Furthermore, if the entity is a *Drone*, its peripherals are installed, together with the associated energy model.

Once all entities are ready, the virtual internet backbone is configured. A CSMA bus is made for the backbone network, identified with address `200.0.0.0/8`. Hosts that can be part of this backbone network are *Remotes*, *Packet Gateways* in the case of an LTE core network, or other routers in the case of the presence of a Wi-Fi BSS.

Finally, in the case of LTE networks, their Radio Environment Maps are set up to generate images that represent the radiation map of the Radio Access Network (RAN).

4) *JSON Configuration Schema*: The entire scenario has been made parametric through the use of a JSON configuration file. Requested at startup, it is decoded and employed to configure and execute the simulation.

In this work, the following configuration schema has been chosen for the General Purpose Scenario:

- **name** – A mandatory string representing the scenario name.
- **dryRun** – An optional boolean to run only the semantic analyser and check that the configuration file and model setup is valid. By default, it is set to false.
- **resultsPath** – A mandatory string representing an existing path to store simulation output files.
- **logOnFile** – A mandatory boolean to output scenario logging information on a file or on standard output.
- **duration** – A mandatory integer that specifies the simulation duration in seconds.
- **staticNs3Config** – A mandatory array of objects, each with **name** and **value** strings, to address ns-3 static configuration parameters. The array can be empty.
- **world** – An optional object containing the description of the simulated space, in particular whether to place buildings or regions of interest.
- **phyLayer** – A mandatory array of objects, each representing a PHY layer configuration to be used in the scenario. Each PHY object declares its **type**, which is a mandatory string. The chosen type must be supported by the semantic analyser. Additional parameters are specific to the kind of PHY layer being configured, most notable are the chosen propagation delay model and the propagation loss model.
- **macLayer** – Its description is similar to **phyLayer**.
- **networkLayer** – Its description is similar to **phyLayer**.
- **drones** – A mandatory array of objects, each representing a drone to be simulated. A drone requires the following properties to be configured: at least one **netDevices** in order to link it to a protocol stack and setup its network address assignment, a **mobilityModel** according to the ones available on IoD-Sim, at least one application that can be installed on a drone, a **mechanics** to define mechanical properties, and a **battery**. Optionally, a **peripherals** array can also be specified in order to equip I/O devices to the drone with a specific **PowerConsumption** indication. They may also be activated by specifying the region of interest through **RoITrigger** parameter.
- **ZSPs** – Its description is similar to **drones**.
- **remotes** – A mandatory array of objects, each representing a remote that is described by its set of applications.
- **logComponents** – A mandatory array of strings to enable log components available in IoD-Sim.

An example of JSON configuration file that realizes a simple scenario is shown in Figure 6.

```
{
  "name": "iod_sim_ftw",
  "resultsPath": "../results/",
  "logOnFile": true,
  "duration": 50,
  "staticNs3Config": [..],
  "world": {
    "buildings": [
      {
        "type": "residential",
        "walls": "concreteWithoutWindows",
        "boundaries": [0.0, 70.0, 0.0, 70.0, 0.0, 20.0],
        "floors": 6,
        "rooms": [2, 1]
      }
    ],
    "regionsOfInterest": [
      {
        "x": 170.0, "y": 340.0, "x2": 180.0, "y2": 250.0, "z": 15.0
      }
    ]
  },
  "phyLayer": [
    {
      "type": "lte",
      "channel": {
        "propagationLossModel": {..},
        "spectrumModel": {..}
      }
    }
  ],
  "macLayer": [..],
  "networkLayer": [
    {
      "type": "ipv4",
      "address": "10.1.0.0",
      "mask": "255.255.255.0",
      "gateway": "10.1.0.1"
    }
  ],
  "drones": [
    {
      "netDevices": [
        {
          "type": "lte",
          "networkLayer": 0,
          "role": "UE",
          "bearers": [..]
        }
      ],
      "mobilityModel": {..},
      "applications": [
        {
          "name": "ns3::DroneClientApplication",
          "attributes": [..]
        }
      ],
      "mechanics": {
        "name": "ns3::Drone",
        "attributes": [..]
      },
      "battery": {
        "name": "ns3::LiIonEnergySource",
        "attributes": [..]
      },
      "peripherals": [
        {
          "name": "ns3::DronePeripheral",
          "attributes": [..]
        }
      ]
    }
  ],
  "ZSPs": [
    {
      "netDevices": [..],
      "mobilityModel": {..},
      "applications": [..]
    }
  ],
  "remotes": [
    {
      "networkLayer": 0,
      "applications": [
        {
          "name": "ns3::DroneServerApplication",
          "attributes": [..]
        }
      ]
    }
  ],
  "logComponents": [..]
}
```

Fig. 6: An excerpt of scenario configuration with an overlay of the models associated to the analyzed parts.

VI. SIMULATION DEVELOPMENT PLATFORM

The *Scenario Configuration Interface*, discussed in the previous Section, eases the design and configuration of complex scenarios from a high-level perspective. Indeed, JSON greatly facilitates management and maintainability thanks to its dry and human-readable syntax. However, the user experience is still hindered by the following:

- As IoD-Sim grows in size and introduces more complex and powerful models over time, the learning curve to effectively use this simulator steepens.
- This project is continuously developed and upgraded with new features, technologies, and standards. A high-level abstraction helps reduce the barrier for scenario developers in approaching new features and the required effort to implement a scenario.
- A general purpose configuration interface, provided in the form of JSON-encoded files, does not give any visual clue on scenario design. Indeed, plain text files alone require low-level knowledge of the simulator, thus implying that the users have to rely on their experience and imagination to effectively know all the aspects related to a complex scenario configuration, such as the number of drones, their trajectories, their purpose, their equipment, the topology of the ground infrastructure, and the services exposed by remote nodes.
- Error reporting messages cannot be easily understood by end-users, forcing the use of a debugger to isolate the problem. Therefore, a semantic analysis would be beneficial to detect problems at scenario configuration.

To address all the points above, the IoD-Sim *Simulation Development Platform* provides a set of extensions, i.e., the *Report Module*, output files for data analysis, and standalone applications for scenario design, such as *Airflow*. These tools ease scenario design and analysis, thus ensuring that IoD-Sim can be easily introduced to newcomers, especially university students and researchers.

A. Report Module

The *Report Module*, illustrated in Figure 7, is an extension of IoD-Sim which stores data at run-time and elaborates, at the end of simulation, a comprehensive summary. The aim of the extension is to introspect simulator's data structures to gather relevant data to be reported (e.g., data traffic, trajectory, and telemetry). To provide a final report that is both human and machine readable, the XML format has been chosen. Therefore, a schema is defined to describe the expected structure of the produced file.

More insights about the structure of the proposed extension are provided hereby. The root XML element, i.e., *Simulation*, represents the summary of a scenario previously executed. The attributes that characterize the simulation are *scenario*, which is a string that carries the name of the scenario that was executed, and *executedAt*, which reports the date and time of execution of this simulation. Moreover, *Simulation* presents further information about simulation results, such as its duration, which is reported in *real* and

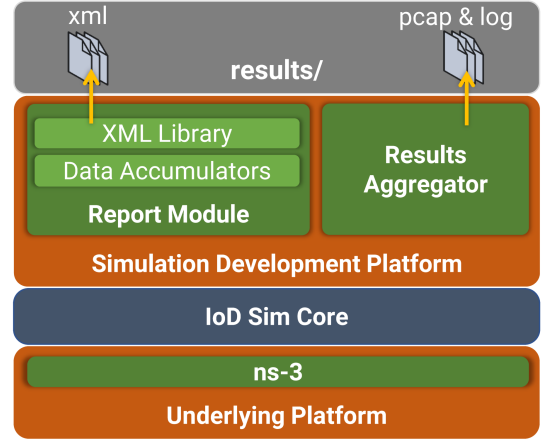


Fig. 7: Block diagram of the Report Module.

virtual time, *World*, which contains the *Buildings* and *InterestRegions*, and entities containers.

The first of these containers is *Zsps*, which is a complex XML type that summarizes each ZSP through position described by the 3D coordinates, and *NetDevices*, which is a list of configured network devices. Each of them is described by structures that represent the configuration of the PHY, MAC, and network layers, together with the data traffic. Each captured packet is expressed by direction, length in bytes, timestamp, and textual representation of the payload.

Similarly, *Drones* summarizes the state of each Drone. This structure maintains the *NetDevices* already discussed for *Zsps*. Additionally, particular characteristics of drones are reported, such as *trajectory* and the set of onboard *Peripherals*. The former is defined by a list of points, each of them with its own timestamp. The latter reports the characteristics of the used peripherals type.

Finally, *Remotes* are described only by their *NetDevices*.

This output XML file is put together with other files relevant to the simulation in the *results* directory.

B. Results Aggregator

Log Files gather all the relevant information and debug messages about the internal components of the simulation. Primarily, the *General Purpose Scenario* emits *progress.log* and *IoD Sim.log* files. The former is the output of the progress information messages that are also delivered on the standard output during scenario execution. The latter contains all debug messages coming from the different internal components of IoD-Sim. The log components can be enabled by specifying them in the *logComponents* field of the scenario configuration JSON file.

progress.log file starts by determining the current date and time of the start of the simulation. For each second, it prints a status report on a single line. The status report presents the following fields:

- The simulation time instant at which the report is referring to.

- The speed up in simulating the scenario with respect to real time. This is dependent on the simulator performance and how many events are elaborated.
- The number of events processed in the time interval relative to the previous status report.

The file then ends with the current date and time and the duration of the simulation as *Elapsed wall clock*.

Trace Files are ASCII-encoded text files that record all the activities regarding a specific Network Device. All the traces are bounded by what is sent or received at the MAC layer. A *Trace File* name is composed of three fields, separated by a hyphen: (i) the global layer name, (ii) the unique identifier of the host in the network, and (iii) the unique identifier of the host network device. For instance, `internet-2-1.tr` indicates that the trace has been done on the first network device of the second host in the virtual Internet network.

LTE Log Files are ASCII-encoded text files that represent a series of statistics on relevant Key Performance Indexes (KPIs). These log files are focused on specific low-level layers of the LTE stack, particularly PHY, MAC, Radio Link Control, and Packet Data Convergence Protocol. For each layer, there are two separate trace files: one for *downlink* and one for *uplink* communications. As part of the *LTE Log Files*, there are also PCAP traces of the *SI-U* interface that links the RAN with the Evolved Packet Core.

PCAP Files are well-known files that record network activity in the PCAP format and contain the traffic that occurred on a certain network device of a host. The filename format is similar to *Trace Files*. Due to the fact that these files are binary, a suitable decoder should be used to explore the data structure. A popular decoder is the `libpcap` open-source project, used by frameworks for PCAP data analysis, e.g., *Scapy*, and Graphical User Interface (GUI) programs such as *Wireshark*. As these *PCAP Files* are generated by a simulation, each captured frame is marked with the relative timestamp of the simulation. Therefore, each *PCAP File* starts with the transmission/reception of captured frames at 0 seconds.

C. Airflow

Airflow is a high-level abstraction tool that gives visual clues during simulation design, thus enriching the user experience, especially for newcomers. Airflow has been developed on top of Splash, a specialized transpiler for IoD-Sim. It scans the source code of the simulator and outputs visual blocks that can be referenced in the *Core Editor* to configure a scenario. Thanks to the GUI editor, a scenario can be exported into a JSON file that can be interpreted by IoD-Sim *Scenario Configuration Interface*. From a software design standpoint, as illustrated in Figure 8, the Airflow project is entirely decoupled from IoD-Sim. Its integration with the simulator relies on interfaces that enable bidirectional communications.

1) *Splash*: Splash is a middleware that analyzes IoD-Sim source code and translates ns-3 models into visual block code used by Airflow. These blocks can be added to the editor as external packages. Splash enables the decoupling mechanism, able to ensure that Airflow and IoD-Sim can be developed asynchronously and updated when needed.

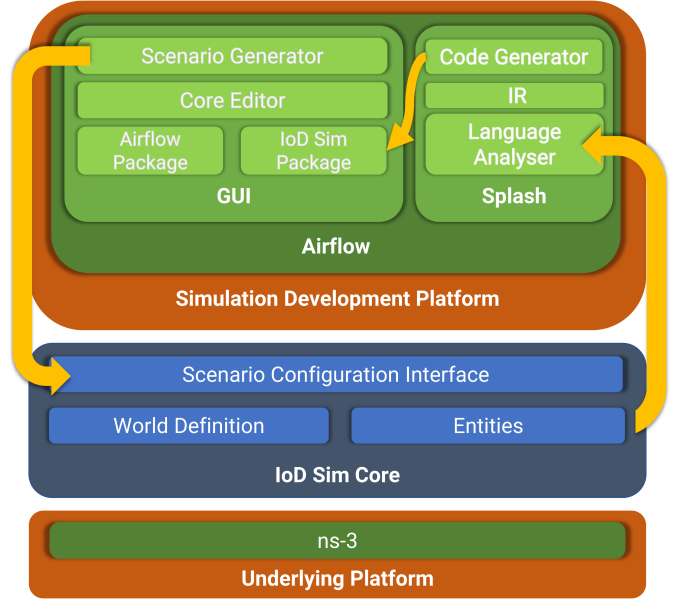


Fig. 8: Airflow Architectural Design.

In particular, it accomplishes the following tasks:

- 1) Parses the source code of IoD-Sim by relying on Clang lexical and syntax analyzers, producing the Abstract Syntax Tree (AST) that is stored into a binary file.
- 2) Scans the AST to find relevant simulation models, excluding internal structures and routines that are not relevant for the design of a scenario. This information is then encoded in an Intermediate Representation (IR).
- 3) Optimizes the IR by solving model hierarchies and removing redundancies.
- 4) Generates Python code that describes the models as Airflow visual blocks. This output can then be moved to the Airflow project folder for integration.

Concretely, this pipeline works as follows. The script

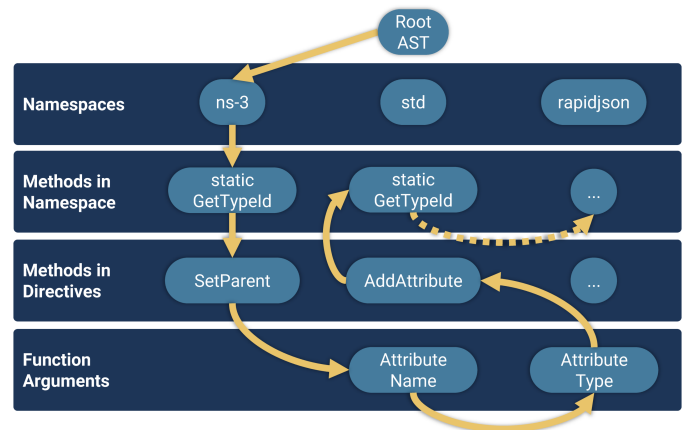


Fig. 9: The tree-traversal search algorithm employed by Splash to extract the models from the IoD-Sim source code. The numerical ordering given on the edges reflects the algorithm logic used to extract model information.

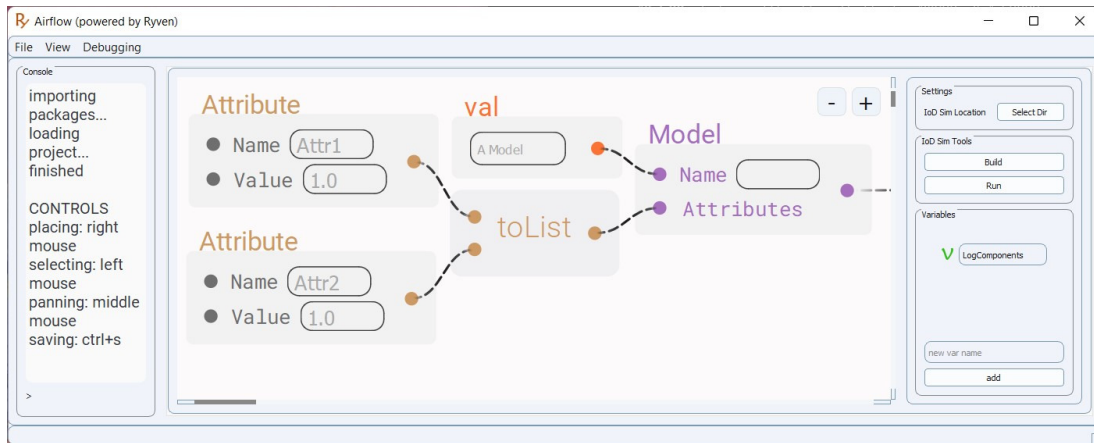


Fig. 10: An overview of the configuration of a generic model in Airflow.

`splash.sh` can be executed by passing the IoD-Sim project directory as an argument. The program then searches for any relevant C++ source code files in it. This process is eased by the ns-3 convention: models have the suffix `-model.cc`, `-manager.cc`, `-mac.cc`, and `-application.cc` in their filenames. To this end, other files are filtered out to optimize parsing operations and to prevent the exposure of the simulator's internal structures.

For each file found, the `clang` command is used to analyze the source code and solve any include directives needed by the preprocessor. Finally, the output is an AST, which is encoded in an optimized binary file readable only through `clang`'s APIs. The file extension is named Precompiled Header (PCH).

The PCH file is then passed into `splash` core executable. This program relies on `cxxopts` library to behave like an interactive command-line application, on `boost-json` to serialize C++ data structures in JSON, and on `libclang` to read the AST. The application requires the PCH file path as input with the output directory path in order to store the IRs. These IR files are encoded into JSON to ensure software interoperability and readability.

Once the command-line program is executed, the entire translation unit of the AST is scanned in order to lookup for any model used in the simulator. A custom tree-traversal algorithm is used to optimize the parse time. It works as a hybrid implementation of the classical *Breadth-first* and *Depth-first* search algorithms. A high-level representation of the translation unit is given in Figure 9. The key feature of this approach is the speed up introduced by the algorithm. In fact, it first traverses the tree using *Depth-first* to find the depth at which one or more `ns3::TypeId` can be found, and then uses *Breadth-first* to analyze each model at the same depth. The same strategy is applied to extract all the attributes relevant to the simulator model. Each model is represented and exported into a JSON file having the following structure: the name of the parent model, the model name, and a list of attributes, each one described by a name, an optional description, and the ns-3 data type that characterizes it. Once the entire model hierarchy is solved and optimized, the attributes are copied from parent to children, if any. Then,

a code generator is executed to create the visual blocks for the editor GUI. Each block name reflects the model's one and the attributes are considered as block input parameters. The generated Python code is interpreted by the GUI to display a visual block with the model name as its title and model attributes as its inputs, as illustrated in Figure 10.

2) *Graphical User Interface*: The Airflow GUI, shown in Figure 10, is based on the open-source engine named Ryven[†], which is a dynamic runtime, *flow-based* visual programming environment for Python scripts. It offers: (i) a central rendering view to place blocks and link them together, (ii) a settings area to customize options, (iii) a variable management section to include and store data that can be integrated with the flow, and (iv) a console to report errors. Ryven includes additional features to optionally debug internal routines with the help of console messages. Moreover, thanks to its modular design, it allows blocks generated by Splash to be aggregated into *packages*. Ryven has been deeply extended to inter-operate with IoD-Sim, especially for its compatibility with the *Scenario Configuration Interface*.

The user interface is organized into the following components:

- 1) A menu bar at the top of the GUI window.
- 2) A *Console* on the left in order to monitor errors and messages coming from Airflow or IoD-Sim. Informative messages are reported in blue, while errors are displayed in red.
- 3) A central workspace to design the scenario by placing blocks and connecting them together.
- 4) A settings panel on the right.

The menu bar is divided into three categories: with *File* it is possible to import Airflow packages to extend the user experience with third-party visual blocks. Moreover, it provides features to save the project or export it as an IoD-Sim configuration file. *View* offers graphical options, such as changing the theme, making a screenshot of the project, and tuning performance parameters. Finally, *Debugging* enables technical features to ease troubleshooting of the program, such as increasing verbosity level on the *Console*.

[†]<https://ryven.org/>

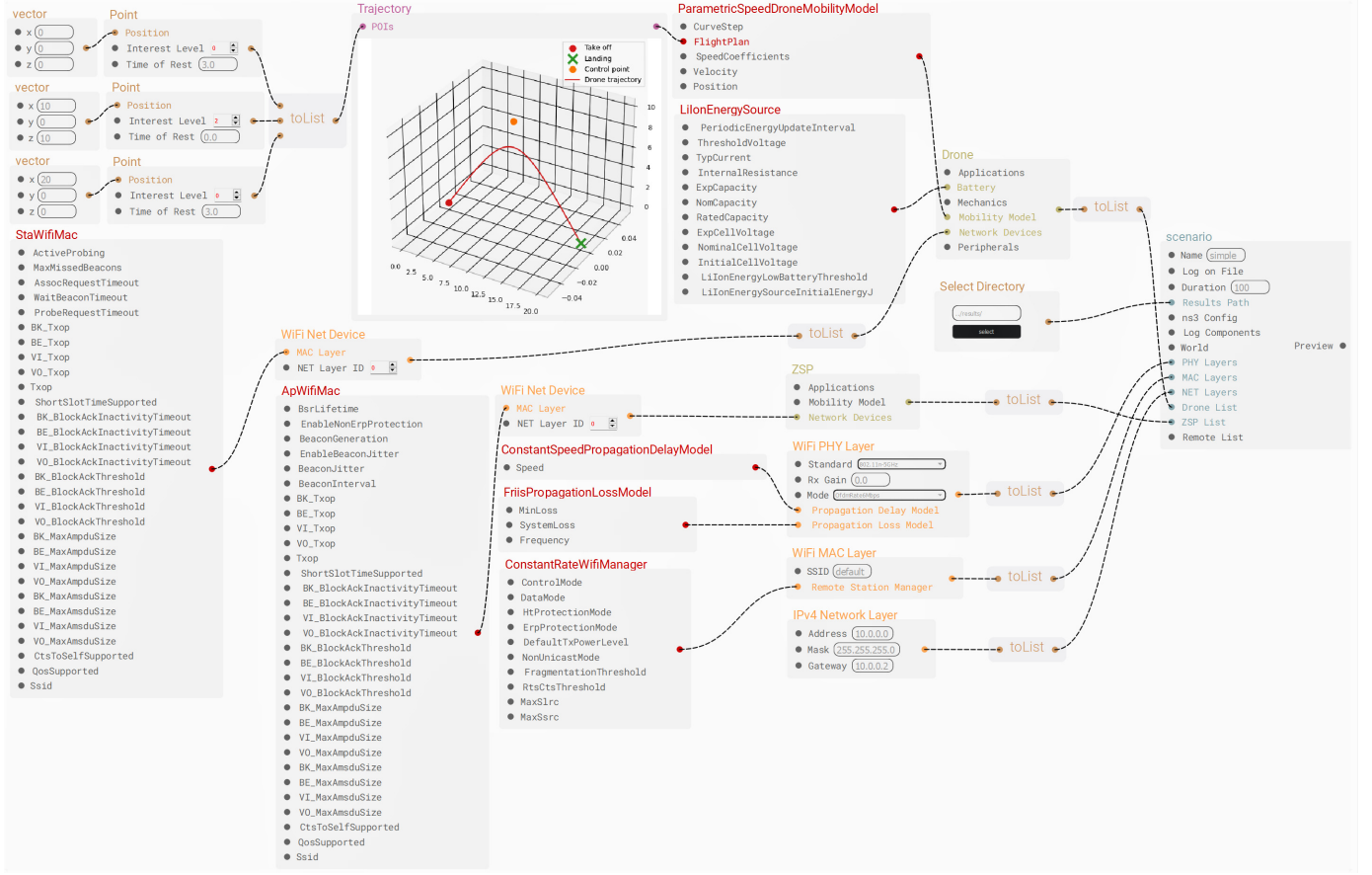


Fig. 11: A simple scenario with one drone and ZSP designed from scratch in Airflow.

The central workspace is the canvas where blocks and links are placed by the user to design a scenario. A block, as depicted in Figure 10, consists of a set of inputs and outputs. Each input and output can be connected to other outputs and inputs of other blocks, in order to create a tree. The root block is named *Scenario*. Each block has a different meaning and function. As a general overview, blocks can be divided into the following categories: operators, helpers, and IoD-Sim models. Operators are built-in blocks that can be used to work with values, constants, and data structures. Instead, helpers are special blocks that ease the configuration of a scenario, i.e., entities, Wi-Fi, and LTE configuration blocks. Usually, blocks provide a single output without a label. This output delivers the information of the block, along with all its inputs, to the next connected block. Blocks can be added to the workspace by a specific menu that is shown by clicking with the right mouse button. Moreover, each block can be right-clicked to show its contextual menu that can be used (i) to remove it, (ii) to refresh it (and hence to read all its inputs again), and (iii) to use some particular features available in certain blocks. For instance, *toList* offers some additional controls to add or remove inputs.

In the settings panel, it is possible to set the IoD-Sim path in order to enable interoperability features, such as checking the scenario configuration for errors, or running the scenario and reporting the status on the *Console*. These features can be

used by clicking on the *Build* and *Run* buttons, respectively. Finally, a variable manager can be used to create, store, and reference values by their respective labels on the workspace. This allows to reduce redundancy and to make the block tree more compact.

VII. SIMULATION CAMPAIGN

This Section demonstrates the huge potential of IoD-Sim by means of an extensive simulation campaign which investigates the many facets of IoD scenarios. Firstly, the discussion explains how the simulation can be designed. Secondly, three different scenarios with increasing complexity are presented.

In particular, the first scenario discusses the use-case of telemetry with a few drones flying in a RoI, which follow customized trajectories while gathering data. The purpose of this scenario is to demonstrate that it is possible to monitor one or more variables with on-board sensors, while estimating the energy consumption associated with flight dynamics.

The second scenario has a wider perspective since it focuses on surveying and monitoring activities, further completed with the acquisition of multimedia signals by each drone. The possible applications include several real-world use cases in the fields of civil engineering, smart agriculture, or environmental monitoring, e.g., coastal erosion and other slow phenomena. In fact, in this scenario, drones are on a mission in neighboring areas since it is assumed that the information of

interest needs to be contextualized, i.e., must be gathered at the same time. Furthermore, this case investigates the possibilities enabled by different data storage capabilities of drones. Also, the offloading functionality of the acquired data avoids the overload/saturation of onboard available resources. Once data is gathered, they can be involved in offline post-processing, evaluation, and analysis.

The third scenario has been specifically designed to be the reference benchmark for IoD applications. It is set in the context of smart cities and involves clusters of low-power IoT sensors. This scenario models real-world applications and, hence, shadowing and pathloss phenomena are included, thanks to the adoption of propagation models that are influenced by the presence of buildings. In order to guarantee a reliable communication, drones are in charge of relaying traffic to ensure coverage to all sensors in the city.

A. Scenario Design

Airflow represents the foremost application for visual scenario development. To better understand how to design simulations, a simple configuration setup is provided hereby. The envisioned scenario considers a drone that follows an arc-like trajectory and communicates telemetry to a ZSP by means of Wi-Fi. Specifically, the drone acts as a *station* and the ZSP as an *access point*. The entire configuration is depicted in Figure 11, where all the visual components, encompassed in the Airflow workspace, are properly set up and linked together. Starting from the right, the block *Scenario* glues some configuration input values, e.g., Name and Duration, with more complex components, such as (i) PHY/MAC/NET Layers, (ii) Drone List, and (iii) ZSP List.

In particular, the communication layers are configured to implement the Wi-Fi stack. The *WiFi PHY Layer* object defines the PHY layer to be used with particular propagation and loss models. The *WiFi MAC Layer*, instead, specifies the SSID of the network and the *Wi-Fi Manager* object that handles the MAC control plane. Further, the *IPv4 Network Layer* determines the address and mask of the overlying network.

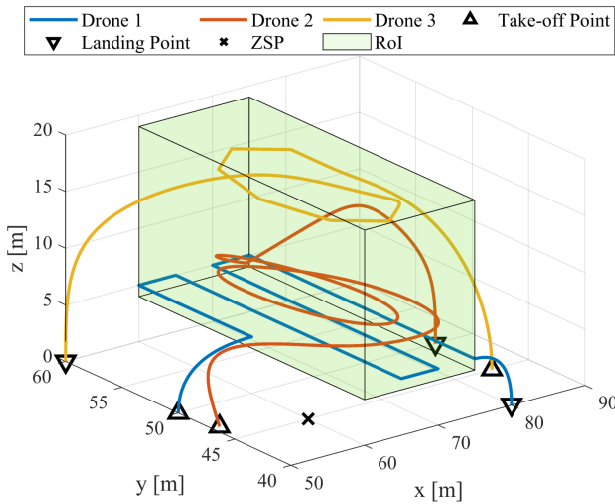


Fig. 12: Scenario #1.

Both *Drone List* and *ZSP List* properties are connected to the simulated entities, namely *Drone* and *ZSP*. These two components share different properties such as *Applications*, *Mobility Model* and *Network Devices*. However, the *Drone* block is also characterized by its unique features, i.e., *Peripherals*, *Mechanics*, and *Battery*. In this configuration, the *ConstantPositionMobilityModel* allows placing the ZSP at a fixed location, while the *ParametricSpeedMobilityModel* is employed to define the drone trajectory. In this regard, the *Trajectory* component, linked to the *FlightPlan* property of the mobility model, facilitates the design of the desired path.

The *Network Devices* property of both drone and ZSP is linked to a *WiFi Net Device* block. While *StaWifiMac* characterizes the device of the former, *ApWifiMac* is associated with the latter. Finally, a *LiIonEnergySource* defines the power supply of the drone.

The development strategy discussed above represents the common ground for the design of the following three scenarios.

B. Scenario #1 - Telemetry

The first scenario, as depicted in Figure 12, envisions three drones with the same mechanical characteristics, all equipped with an Inertial Measurement Unit (IMU). In this scenario, drones are flying in the same RoI at a constant speed, following different trajectories. Moreover, a ZSP is deployed on the ground. The latter is released in $[60 \ 45]^T$, which continuously monitors drones' operations by acquiring telemetry through Wi-Fi.

UAVs' trajectories are based on the *ParametricSpeed-DroneMobilityModel*, which is configured to guarantee a constant speed of 5 m/s, 3 m/s, and 4 m/s, respectively. They are also equipped with IMUs, which are generic drone peripherals that provide basic telemetry data to the ZSP thanks to a dedicated application, as mentioned in Section V-E1. It is worth specifying that drones' IMUs have different power consumption, i.e., 12 W, 5 W, and 6 W.

The outcome of the simulation is hereby discussed. Figures 13 and 14 depict the power consumption trend with respect to time and trajectories. In the former, the three curves share an

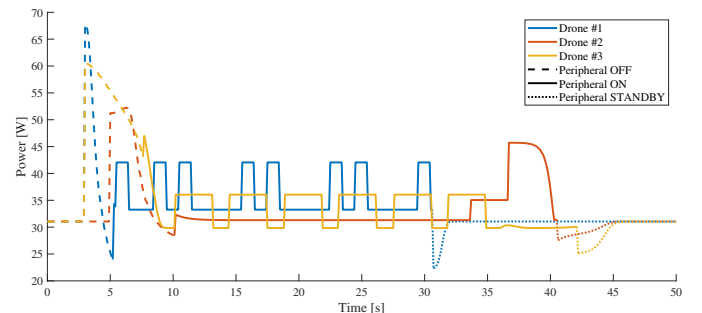


Fig. 13: Power consumption and peripheral state for each drone, in the first scenario.

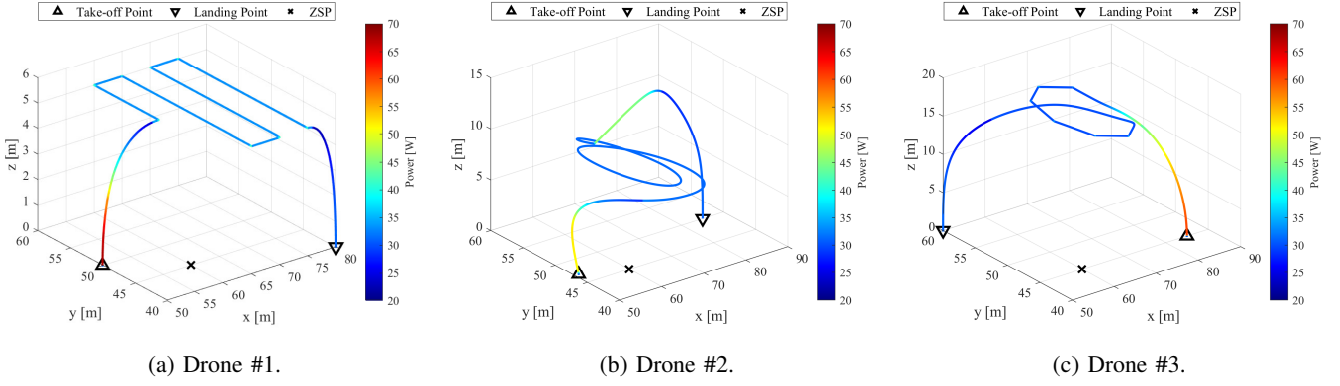


Fig. 14: Drones' trajectories with their power consumption, in the first scenario.

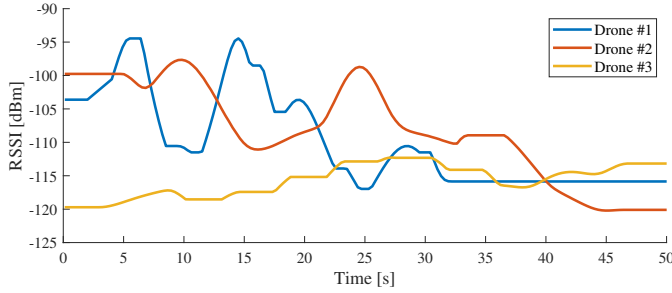


Fig. 15: Measurement of the Received Signal Strength Indicator (RSSI) of each drone by the ZSP in the first scenario.

initial peak which corresponds to the energy required to take off. Indeed, acquiring altitude requires more power than flying along the xy plane, as highlighted. This phenomenon is further remarked in Drone #2 landing maneuver. It includes a little parabola that yields a peak in the last part of the associated curve of Figure 13, which is also present in Figure 14b. After ~ 10 s, the drones reach and almost maintain a target altitude. The corresponding power consumption, for Drones #1 and #3, is characterized by peaks due to hovering over the interest points for 1 s and 3 s, respectively. These points are identified by the vertices of the snake-like and octagon-shaped trajectories. Instead, this phenomenon is not present on Drone #2, since its trajectory describes a continuous curve. When the drones enter the RoI, the peripherals become active, and hence the IMUs power contribution is non-zero. Spikes can be noticed in the curves of Figure 13, especially in Drones #1 and #2, since they are equipped with two more energy-demanding peripherals. As soon as drones exit such a region, the peripherals go into standby mode, which preserves energy.

Figure 15 illustrates the measured RSSI of each drone during the mission. Measurements are carried out by the ZSP. In general, such values can be conceived as an assessment of ranging operations carried out by a single node when its position is fixed. From this Figure, it clearly emerges that, on average, Drones #1 and #2 maintain a better signal quality with respect to the UAV #3. Obviously, the higher altitude, and hence the greater distance from the ZSP, worsens the communication quality due to the Friis propagation loss model employed to model the fading effects in this scenario.

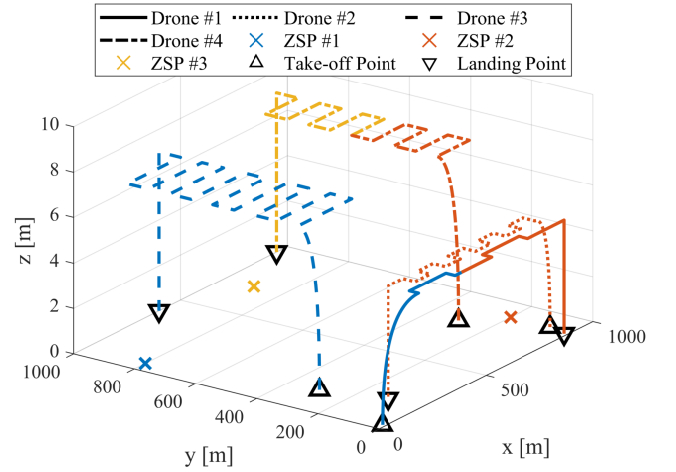


Fig. 16: Trajectory design and eNB attachment for each drone, in the second scenario.

C. Scenario #2 - Multimedia Signals Acquisition

The second scenario is depicted in Figure 16. A swarm of four drones is in charge of acquiring multimedia signals in an operating area that is 10^6 m^2 wide. Acquired data are stored on-board and off-loaded to a remote server as soon as the drone is able to communicate with a ground infrastructure. The latter, which allows data upload, is composed of three ZSPs, also referred to as eNBs, that are deployed on the ground in three different locations: $[50 \ 800]^T$, $[900 \ 200]^T$, and $[700 \ 900]^T$, respectively. All the entities involved in the mission, which lasts 250 s, are equipped with LTE interfaces, where the Okumura-Hata propagation loss model has been employed. Drones follow snake-like trajectories, each different from the other in terms of amplitude and frequency. Nevertheless, they adopt the same mobility model with a constant acceleration of 4 m/s^2 and a maximum velocity between 15 and 20 m/s. Each drone is equipped with cameras that operate at different data rates, 2 Mbps, 1.6 Mbps, 1.3 Mbps, and 1 Mbps, respectively. The communication between each UAV and the remote server is handled by *Generic Traffic Applications* (see Section V-E2), with a payload size of 1024 bytes and a TCP Max Segment Size of 1380 bytes.

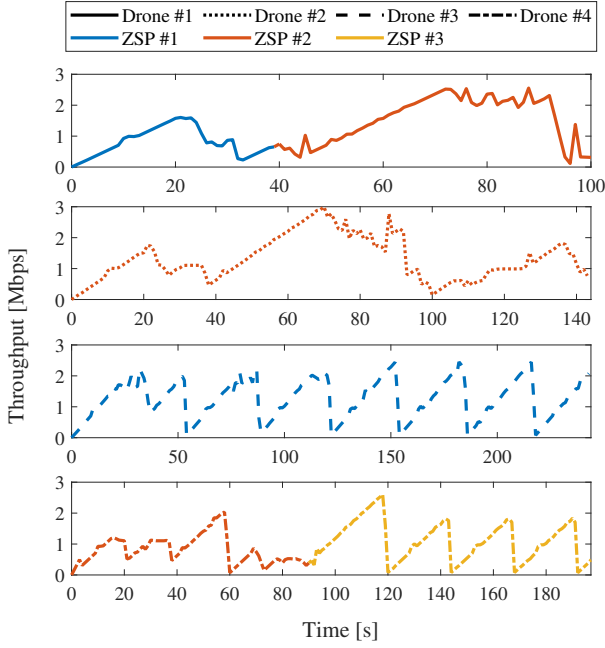


Fig. 17: Drones' throughput, in the second scenario.

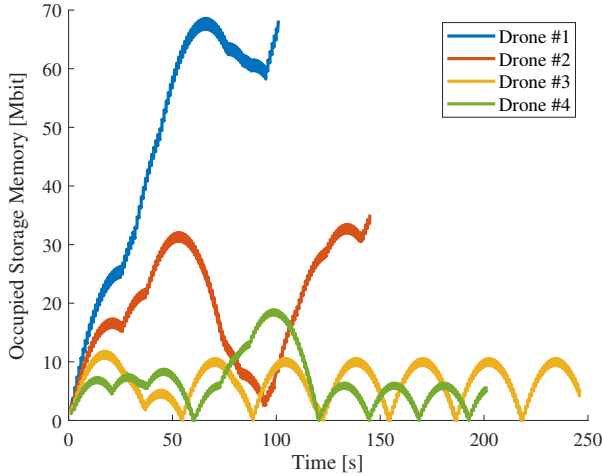


Fig. 18: Memory occupancy for each drone, in the second scenario.

In the same figure, it can be further observed the attachment of the drones to the ZSPs. Throughout the mission, Drones #2 and #3 remain linked to the same eNB, i.e., ZSP #2 and #1. On the other hand, UAV #1 and #4 perform a handover procedure which changes the reference ZSP from #1 to #2 and from #2 to #3, respectively. It is worth noting that, despite Drone #1 takes off in the same area where Drone #2 lands, they are not attached to the same ZSP. Indeed, even if the two trajectories share the same direction, they have opposite verse: while one approaches an eNB, as the mission goes by, the other flies away from the ZSP without really getting closer to another one. Figure 17 shows the throughput for each drone on the associated ZSP over time. It is shown

that UAV #1 experiences an average data rate of ~ 1 Mbps until the handover procedure takes place, which increases this value by $\sim 50\%$. Similarly, the average throughput of Drone #4 is also ameliorated since it increases from ~ 800 kbps to ~ 1.1 Mbps. It is worth noting that there exists a pattern correspondence between the throughput and the occupied storage curves (see Figure 18). This is particularly evident for Drones #3 and #4: when the occupied memory lowers and goes to zero, the data rate decreases as well, and tends to zero. Indeed, for the information causality principle, it is not possible that a larger amount of information is transmitted with respect to the stored one. Notice that this happens as long as the acquisition rate remains lower or equal to the channel capacity which, for instance, is not the case of Drone #1.

D. Scenario #3 - Smart Cities

The third scenario reproduces a smart city context, in which drones are in charge of relaying traffic coming from clusters of Ground Users (GUs), using Wi-Fi technology, to a remote server over the Internet, through LTE. In this regard, the presence of buildings plays an important role both in trajectory design and in fading phenomena. The envisioned scenario is designed starting from the map of an urban area in the neighborhood of the Central Station of Bari, Puglia, Italy.

The xy coordinates (i) are extracted from OpenStreetMap with the aid of OpenCV [29], (ii) rescaled according to their real profile, and (iii) transposed into the spatial reference system of the simulator. Finally, the buildings' heights are generated using a random variable uniformly distributed in [24, 30], which corresponds to the characteristic height (in meters) of the buildings in that area. As shown in Figure 19, four GUs clusters of different size are present on the ground. Each of them is served by a drone, which relays the

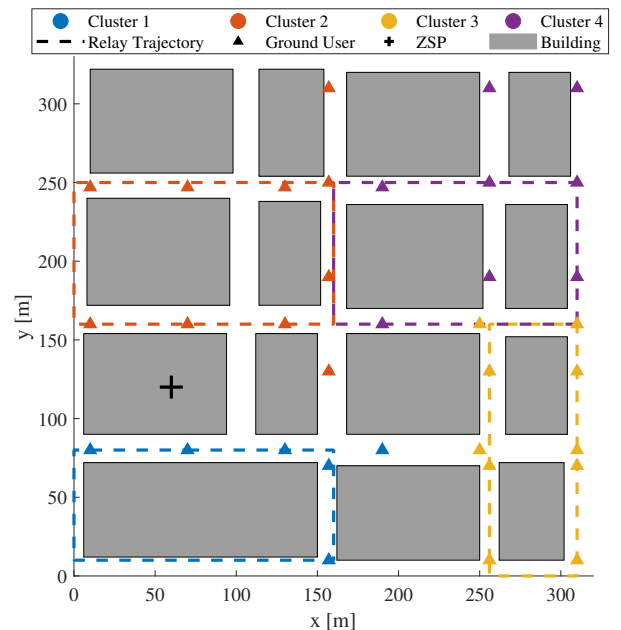


Fig. 19: Scenario #3 simulation environment.

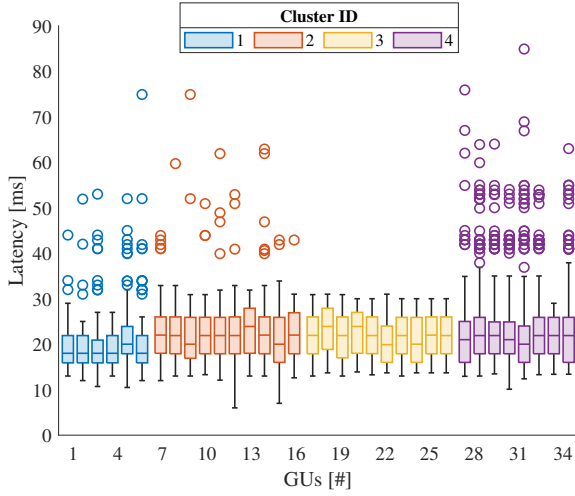


Fig. 20: GUs application latency of link combined by Wi-Fi, relay drone, and LTE.

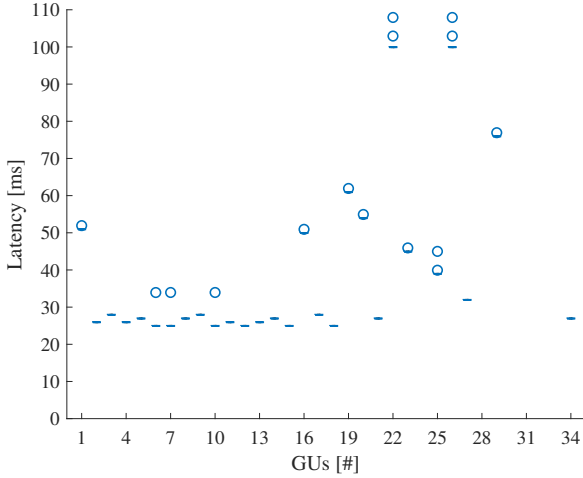


Fig. 21: GUs application latency over LTE-only link.

traffic by means of the NAT application discussed in Section V-E3. The entire simulation lasts 180s and employs the `ns3::HybridBuildingsPropagationLossModel` to take into account the fading caused by the presence of buildings. It includes a combination of Okumura-Hata model and COST231 for long-range communications, ITU-R P.1411 for short-range communications, and ITU-R P.1238 for indoor ones. This allows to support a wide range of frequencies spanning from 200 MHz up to 2600 MHz. Moreover, each building is characterized by a window per room and is assumed to be built with concrete walls. The Wi-Fi stack has been configured based on the 802.11ax standard operating at 2.4 GHz and is controlled by the `ns3::IdealWifiManager`, which allows to keep track of the SINR. Thanks to this mechanism, it is possible to always choose the best transmission mode to be used, i.e., a combination of modulation, coding scheme, and data rate.

As for the network level, each cluster is connected to its relay according to the 10.[1 – 4].0.0/24 network address

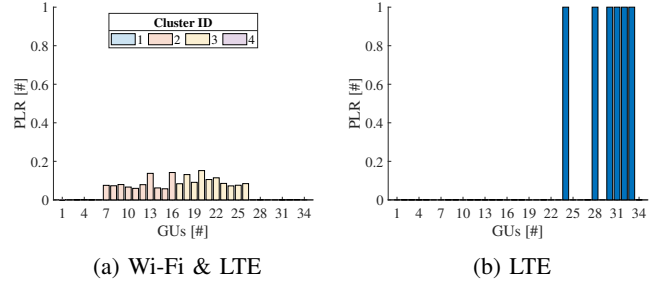


Fig. 22: GUs application PLR for Scenario #3.

Scenario #	Events [#]	real time [s]	Sim. Time [s]
1	57,437	9	50
2	18,226,323	761	250
3 LTE	37,178,812	4,620	180
3 Wi-Fi & LTE	28,903,306	2,858	180

TABLE X: Comparison of the total number of events, the real time taken to execute, and the simulated time of each scenario.

range, while LTE uses 7.0.0.0/8. Drones' trajectories are designed to the layout of the streets in order to minimize the shadowing effects and maximize the Line of Sight with the GUs. Furthermore, the path also maximizes energy efficiency as the translation in the xy plane is less costly when compared to changes of altitude. At each angle of the trajectory, the drones pause for 1 s in order to simulate an accurate 90 degrees yaw.

Accordingly, each relay drone flies at a constant altitude of 50 m at 5 m/s. Drones are equipped with the `ns3::NatApplication`, which implements a simple Port-based NAT strategy for UDP communications. Each GU has a constant position and is equipped with a simple `ns3::UdpEchoClientApplication`, which periodically sends a packet of 1024 bytes to the remote address 200.0.0.1:1337 with a frequency of 10 Hz. Each packet is equipped with an application header that reports an incremental sequence number and the time of creation. Finally, the remote has a `ns3::DroneServerApplication`, which records via log messages the received packets.

The only ZSP, located at [60, 120, 40]^T, provides LTE access to the drones, thus allowing the communication with the remote host. Figures 20 and 21 clearly show the advantage brought by the relay activity by the drones. In the relay case (Figure 20), all the GUs experience an average latency of ~ 25 ms, a result that is achieved also thanks to the proposed trajectory design.

On the contrary, in absence of relay drones (see Figure 21), while the GUs that are closer to the ZSP are affected by a latency similar to the previous case, the farther ones register a significant delay, which inevitably compromises the reliability of the link and, hence, the QoS. Nevertheless, this comes with a trade-off as highlighted in Figure 22, which shows the PLR in both cases. In the former, all nodes are able to transmit data to the remote, but with a loss ratio of $\sim 10\%$ for the cluster #2 and #3. It is worth noting that this result can be further improved by properly optimizing the trajectory design

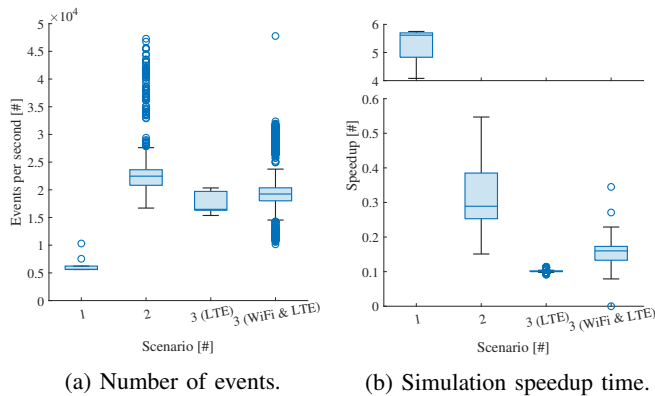


Fig. 23: Performance evaluation of the different simulated scenarios.

to target the desired trade-off. In the latter, instead, six nodes have 100% PLR, which means that there is no exchange of data.

E. Performance Evaluations

To evaluate the performance of the simulator, and hence its scalability, the performance metrics of the simulated scenarios are analyzed and compared hereby. The runtime environment is characterized by the following hardware and software specifications: (i) Intel (R) Xeon (R) Bronze 3106 at 1.70 GHz with 16 cores and no hyper-threading, (ii) RAM 92 GB DDR4 at 2666 MHz, (iii) 7200 RPM hard drives and (iv) OS Fedora 35 on LXD container [30]. It is worth specifying that the present assessment is made leveraging a single-core configuration, although multi-processing support is available. To fairly compare the simulations, two metrics are selected. The former takes into account the number of events processed per second for each simulation, thus providing an insight related to the scenario complexity. The latter considers the ratio between the simulated time and the real time, thus further addressing the complexity of the designed missions. Moreover, Table X summarizes the total number of events, the time taken to simulate (real time), and the simulated time of each scenario. It is worth noting that all scenarios are constructed differently and hence are difficult to compare. However, some clear indications can be derived from the following analysis. Indeed, Figure 23 shows that in Scenario #1 the employment of Wi-Fi technology slows the number of events processed per second, which means that the complexity is higher. On the contrary, the adoption of LTE (either mixed with Wi-Fi) reduces the overall computational complexity. However, in the first case (Scenario #1) the speedup is greater with respect to the second case (remaining scenarios): this is due to the fact that the number of generated events is way lower. This is particularly evident in Scenario #3, where the simulation time and the number of GUs are the same, as shown in Table X. Overall, even if the number of actors increases when drone relays are employed (LTE & Wi-Fi), the lower number of events generated guarantees better performance.

Moreover, in order to further investigate the simulator performance and derive more insights regarding the required

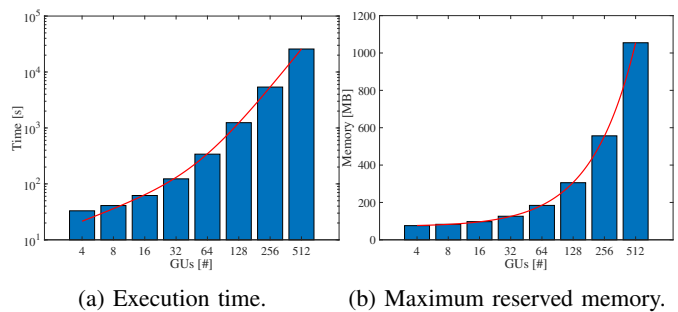


Fig. 24: Performance evaluation of the simulator with respect to the number of GUs.

Metric	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
Time	117.2	1.197	87.45	3.961
Memory	64.35	-0.05182	85.62	1.717

TABLE XI: Coefficients of Equation 7.

resources to run a computationally complex scenario, the following final evaluation is provided. A square area is partitioned into four quadrants, each one with a central drone relay. According to a uniform random distribution, a set of GUs is generated and symmetrically placed into the four regions with respect to the BS, which is placed in the center of the area. The number of considered GUs is then increased in accordance with the power of 2. Given this scenario, execution time and the maximum reserved memory are considered reference metrics and are reported in Figure 24. As it can be deduced, both exponentially grow with an increasing number of GUs, as confirmed by the regression performed on the obtained data. Indeed, it is possible to predict the time and memory required for a specific simulation as

$$ae^{bx} + ce^{dx}, \quad (7)$$

where x is the number of GUs, and a, b, c, d are the fitted coefficients provided in Table XI.

VIII. CONCLUSION

The IoD paradigm enables trailblazing applications, as the flexibility proposed by drones may significantly boost the effectiveness of existing activities, e.g., first response, monitoring, delivery, and surveillance. Moreover, drones are already becoming pervasive in several industrial sectors, such as smart agriculture, proactive maintenance, civil engineering, and many more.

As a matter of fact, the large-scale adoption should be evaluated after a prototyping phase that can be time-consuming and may require unfeasible costs. To tackle this problem, simulators are an essential tool to facilitate the testing phase and state the readiness for real-world exploitation. At the same time, simulators can be a learning tool for young professionals, engineering students, and researchers to improve their knowledge and explore scenarios never considered before.

In these regards, IoD-Sim is a thorough and user-welcoming tool that can be used to evaluate the many facets of IoD scenarios, including trajectory design, networking functionalities,

mechanical characteristics, and data analytics. Nevertheless, IoD-Sim has been created as a modular tool that can be updated and upgraded as needed. A Visual Programming Editor for IoD-Sim has also been developed, relying on compilers' theory and tools to dynamically update its contents based on the main simulator platform, ensuring that such project can be maintained with ease in the long term. Moreover, a predictable build environment is used to ease the installation due to its dependencies that require careful setup and knowledge about the underlying simulator, libraries, and compilers.

Even though IoD-Sim is a reliable solution, in the future more efforts will be focused on the improvement of the entire project, especially along the following research and development lines:

- Extend the support to design scenarios using technologies such as MAVlink, satellite communications, and 5G-New Radio.
- Speedup in Splash compilation with the use of parallel multiprocessing and optimized algorithms.
- Develop interactive visual blocks to preview or design more accurate simulations in less time.
- Improve the overall User Experience of the visual editor.
- Allow the employment of multi-processing systems and clusters.
- Directly compare the performance and the features with other IoD simulation platforms.

Finally, the birth of a thriving and empowering community on open-source collaboration platforms will be crucial in assessing the future development efforts of this work.

REFERENCES

- [1] M. Gharibi, R. Boutaba, and S. L. Waslander, "Internet of drones," *IEEE Access*, vol. 4, pp. 1148–1162, 2016.
- [2] P. Boccadoro, D. Striccoli, and L. A. Grieco, "An extensive survey on the internet of drones," *Ad Hoc Networks*, vol. 122, p. 102600, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1570870521001335>
- [3] M. A. Hoque, M. Hossain, S. Noor, S. M. R. Islam, and R. Hasan, "IoTaaS: Drone-Based Internet of Things as a Service Framework for Smart Cities," *IEEE Internet of Things Journal*, vol. 9, no. 14, pp. 12425–12439, 2022.
- [4] S. Horsmanheimo, L. Tuomimäki, V. Semkin, S. Mehnert, T. Chen, M. Ojennus, and L. Nykänen, "5G Communication QoS Measurements for Smart City UAV Services," in *2022 16th European Conference on Antennas and Propagation (EuCAP)*, 2022, pp. 1–5.
- [5] S. H. Alsamhi, O. Ma, M. S. Ansari, and S. K. Gupta, "Collaboration of drone and internet of public safety things in smart cities: An overview of qos and network performance optimization," *Drones*, vol. 3, no. 1, p. 13, 2019.
- [6] N. S. Labib, M. R. Brust, G. Danoy, and P. Bouvry, "The Rise of Drones in Internet of Things: A Survey on the Evolution, Prospects and Challenges of Unmanned Aerial Vehicles," *IEEE Access*, vol. 9, pp. 115466–115487, 2021.
- [7] —, "The rise of drones in internet of things: A survey on the evolution, prospects and challenges of unmanned aerial vehicles," *IEEE Access*, vol. 9, pp. 115466–115487, 2021.
- [8] S. H. Alsamhi, O. Ma, M. S. Ansari, and F. A. Almalki, "Survey on collaborative smart drones and internet of things for improving smartness of smart cities," *IEEE Access*, vol. 7, pp. 128125–128152, 2019.
- [9] S. H. Alsamhi, F. A. Almalki, H. Al-Dois, S. Ben Othman, J. Hassan, A. Hawbani, R. Sahal, B. Lee, and H. Saleh, "Machine learning for smart environments in B5G networks: connectivity and QoS," *Computational Intelligence and Neuroscience*, vol. 2021, 2021.
- [10] S. Liao, J. Wu, J. Li, A. K. Bashir, and W. Yang, "Securing collaborative environment monitoring in smart cities using blockchain enabled software-defined internet of drones," *IEEE Internet of Things Magazine*, vol. 4, no. 1, pp. 12–18, 2021.
- [11] Y. Zeng, I. Guvenc, R. Zhang, G. Geraci, and D. W. Matolak, *UAV Communications for 5G and Beyond*. John Wiley & Sons, 2020.
- [12] S. Baidya, Z. Shaikh, and M. Levorato, "Flynetsim: An open source synchronized uav network simulator based on ns-3 and ardupilot," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWIM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 37–45. [Online]. Available: <https://doi.org/10.1145/3242102.3242118>
- [13] N. R. Zema, A. Trotta, G. Sanahuja, E. Natalizio, M. Di Felice, and L. Bononi, "Cuscut: An integrated simulation architecture for distributed networked control systems," in *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2017, pp. 287–292.
- [14] E. A. Marconato, M. Rodrigues, R. d. M. Pires, D. F. Pigatto, A. R. Pinto, K. R. Branco *et al.*, "Avens-a novel flying ad hoc network simulator with automatic code generation for unmanned aircraft system," in *Proceedings of the 50th Hawaii international conference on system sciences*, 2017.
- [15] J. A. Millan-Romera, J. J. Acevedo, A. R. Castaño, H. Perez-Leon, C. Capitán, and A. Ollero, "A utm simulator based on ros and gazebo," in *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*, 2019, pp. 132–141.
- [16] M. Tropea, P. Fazio, F. De Rango, and N. Cordeschi, "A new fanet simulator for managing drone networks and providing dynamic connectivity," *Electronics*, vol. 9, no. 4, 2020. [Online]. Available: <https://www.mdpi.com/2079-9292/9/4/543>
- [17] S. Acharya, B. Amrutur, M. Bharatheesha, and Y. Simmhan, "Cornet 2.0: A co-simulation middleware for robot networks," 2021.
- [18] S. Park, W. G. La, W. Lee, and H. Kim, "Devising a distributed co-simulator for a multi-uav network," *Sensors*, vol. 20, no. 21, p. 6196, 2020.
- [19] G. Grieco, R. Artuso, P. Boccadoro, G. Piro, and L. Grieco, "An open source and system-level simulator for the internet of drones," in *Proc. of IEEE International Workshop on Internet of Mobile Things (IoMT), in conjunction with PIMRC 2019*, Istanbul, Turkey, Sep. 2019.
- [20] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.
- [21] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No. 04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.
- [22] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_2
- [23] A. Varga, *OMNeT++*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 35–59. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_3
- [24] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, and R. Ulerich, *GNU scientific library*. Network Theory Limited, 2002.
- [25] M. Yip, "Rapidjson—a fast json parser/generator for c++ with both sax/dom style api," *THL A29*. [Online]. Available: <https://github.com/miloyip/rapidjson>, 2015.
- [26] C. Shepherd, "Theoretical design of primary and secondary cells. part 3. battery discharge equation," *NAVAL RESEARCH LAB WASHINGTON DC*, Tech. Rep., 1963.
- [27] O. Tremblay, L.-A. Dessaint, and A.-I. Dekkiche, "A generic battery model for the dynamic simulation of hybrid electric vehicles," in *2007 IEEE Vehicle Power and Propulsion Conference*. Ieee, 2007, pp. 284–289.
- [28] Sun, Y. and Xu, D. and Ng, D. W. K. and Dai, L. and Schober, R., "Optimal 3d-trajectory design and resource allocation for solar-powered uav communication systems," *IEEE Transactions on Communications*, vol. 67(6), pp. 4281–4298, 2019.
- [29] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, "A brief introduction to opencv," in *2012 Proceedings of the 35th International Convention MIPRO*, 2012, pp. 1725–1730.
- [30] S. Senthil Kumaran, *Practical LXC and LXD: linux containers for virtualization and orchestration*. Springer, 2017.



Giovanni Grieco received the Dr. Eng. degree (with honors) in Telecommunications Engineering from Politecnico di Bari, Bari, Italy in October 2021. His research interests include Internet of Drones, Cybersecurity, and Future Networking Architectures. He is the principal maintainer of IoD_Sim. Since 2021, he has been a Ph.D. student at the Department of Electrical and Information Engineering at Politecnico di Bari.



Giovanni Iacovelli received the Dr. Eng. degree (with honors) in information engineering from Politecnico di Bari, Bari, Italy, in July 2019. His research interests include Internet of Drones, Machine Learning, Optimization and Telecommunications. Since November 2019 he is a Ph.D. Student at the Department of Electrical and Information Engineering, Politecnico di Bari.



Pietro Boccadoro received the Dr. Eng. degree (with honors) in electronic engineering from Politecnico di Bari, Bari, Italy, in July 2015. From Nov. 2015 to Oct. 2020, he collaborated as a researcher at Politecnico di Bari. In 2021, he finished his Ph.D. Course. He is currently R&D Software engineer at Nextome srl. His research interests include Internet of Drones, Robotic-aided IoT and Future Internet Architectures.



L. Alfredo Grieco is a full professor in telecommunications at Politecnico di Bari. His research interests include Internet of Things, Future Internet Architectures, and Nano-communications. He serves as Founder Editor in Chief of the Internet Technology Letters journal (Wiley) and as Associate Editor of the IEEE Transactions on Vehicular Technology journal (for which he has been awarded as top editor in 2012, 2017, and 2020).