

Cybersecurity Simulation Environment for Drones in Naval Applications

Henrique Marques da Silva CATARINO ^a, Luís PINTO ^a, Luís SOUSA ^a,
José CECÍLIO ^a, Pedro M. FERREIRA ^a, and Alan Oliveira de Sá ^{a,1},

^a*LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal*

ORCID ID: Luís Pinto <https://orcid.org/0000-0003-1997-5484>, José Cecílio

<https://orcid.org/0000-0002-5351-5580>, Pedro M. Ferreira

<https://orcid.org/0000-0003-2369-0115>, Alan Oliveira de Sá

<https://orcid.org/0000-0001-6311-9672>

Abstract. The use of drones in naval applications, both military and civilian, has raised concerns about their security. Their protection should not be limited to the physical domain of kinetic warfare or the electromagnetic domain of electronic warfare. It must also take into account cyber threats, given the extensive reliance of drones on computational and communication systems. To support the study of cyber threats and the development of security solutions for drone systems, this work presents a simulation platform (CyDronE) that brings together not only elements of the cyber domain but also properties of the physical domain. CyDronE is implemented on a containerized Robot Operating System 2 (ROS 2) framework, which executes the drones' onboard computational processes. Gazebo is integrated as the physics-based simulation engine, providing high-fidelity modeling of the drones' flight dynamics and environmental interactions during the experiments. This approach enables a consistent assessment of how cyber actions influence the physical behavior of drones. System components are defined through declarative YAML configurations and instantiated into deterministic, per-swarm deployments, allowing multiple isolated swarm instances to execute concurrently on the same host without mutual interference. To evaluate the system, we conduct experiments involving a swarm of drones operating under adversarial conditions. Specifically, we implement a controlled UDP flood attack using dedicated attacker containers that generate traffic directed at the same IP addresses and ports employed by DDS/RTPS communication. The simulations demonstrate that flooding can hinder the timely delivery of legitimate state updates, resulting in degraded vehicle motion, transitioning from continuous trajectories to prolonged hovering with intermittent advancement. These experimental results demonstrate the system's capability to support cyber-physical assessments of cybersecurity activities in drone applications. Additionally, the assessment of the computational resources consumed by CyDronE suggests its good scalability.

Keywords. Cybersecurity, Drone, Swarm, Cyber-Physical Systems, Naval, ROS 2, Gazebo, DoS, Simulation

© Henrique Catarino, Luís Pinto, Luís Sousa, José Cecílio, Pedro Ferreira, and Alan Oliveira 2026. The definitive and edited version of this Article is published in NATO Science for Peace and Security Series - D: Information and Communication Security, Volume XX: XXXXXXXX, Editors: XXXXXXXX, online ISBN XXXXXXXX, pages XXX - XXX, 2026, DOI: XXXXXXXX. These details can be found on <https://ebooks.iospress.nl>.

¹Corresponding Author: Alan Oliveira de Sá, alan@di.fc.ul.pt.

1. Introduction

The increasing use of drones in naval applications, across both military and civilian contexts, has raised significant concerns regarding their security and resilience. Traditionally, the protection of these systems has focused on the physical domain of kinetic warfare and the electromagnetic domain of electronic warfare [Yu et al.(2025)]. However, modern naval drones rely heavily on complex computational platforms, software-defined functionalities, and communication networks for navigation, control, and mission execution. As a result, their security must also explicitly address cyber threats [Oudina et al.(2024)], which introduce new vulnerabilities capable of compromising mission integrity, operational safety, and strategic advantage.

In this context, it becomes important to understand the vulnerabilities and types of attacks that can compromise the proper functioning of drones, not only from the perspective of the cyber domain, but also in terms of the impact these attacks may have on the physical domain. This understanding is relevant not only to assess how far an attack can meet the principle of kinetic effect of cyberwarfare – as proposed by Parks and Duggan, who argue that cyber operations must produce tangible consequences in the physical (kinetic) world to be meaningful – but also to support the development of effective countermeasures. One possible approach to assessing the cybersecurity of drones and the associated physical impacts of attacks is through real-world experiments using physical devices. However, such experiments involve costs and can carry the risk of accidents and material damage. Another possibility is to use simulations that capture the inherent characteristics and properties of both the cyber and physical domains to which drones are subject. The main challenge of the second alternative lies in the lack of simulation environments capable of simultaneously providing integrated realism across both the cyber and physical domains.

This gap is particularly evident in drone swarms, where distributed coordination significantly expands the attack surface and can compromise system-level security and reliability. Swarms depend on the timely dissemination of neighbour-state, formation, and supervisory-control messages. When these communication flows are disrupted – through bandwidth exhaustion, protocol-level interference, or targeted flooding – collective behavior can degrade from cohesive motion to stalled or unstable dynamics. There is a lack of simulators for studying how attacks affect formation stability, state propagation, and control authority. Many existing simulators emphasize physics fidelity and algorithmic evaluation but do not support adversarial traffic generation or the side-by-side, isolated replication of experiments to quantify robustness across parameters, versions, or patches.

To address these needs, we present the CyDronE platform, built upon Robot Operating System 2 (ROS 2) and Gazebo technologies. A containerized ROS 2-based architecture executes the drones’ onboard computational processes, while Gazebo is integrated as the physics-based simulation engine, providing high-fidelity modeling of the drones’ flight dynamics and environmental interactions during the experiments. The platform is specified through declarative YAML configurations that compile into deterministic per-instance deployments.

The environment is designed not only to enable the deployment of representative ROS 2-based drone scenarios, but also to serve as a reusable attack testbed. It supports the execution of adversarial actions, systematic data collection at both network and host levels, and the observation of system behavior during and after attacks. To evaluate Cy-

DronE, we empirically assess the impact of a Denial-of-Service (DoS) attack against a swarm of ROS 2-based drones while flocking. Through controlled experiments, including traffic generation with a common tool – i.e., `hping3` [Sanfilippo(2025)] –, we demonstrate and measure the physical impact of the attacks. Therefore, the main contributions of this work are:

1. The development of a simulation framework to study the cybersecurity of drones, taking into account the effects on both cyber and physical domains;
2. The demonstration and assessment of a DoS attack in a swarm of ROS 2-based drones.

Beyond these technical contributions, this study aims to raise awareness of drone cybersecurity and encourage further research into effective countermeasures. By demonstrating how attacks can compromise system availability – and consequently the reliability of robotic functions – we highlight the importance of security-by-design principles, repeatable evaluation methodologies, and rigorous validation of defensive mechanisms before deployment in critical applications.

The remainder of this chapter is organized as follows: Section 2, presents the related works; Section 3, provides the architectural details of the CyDronE platform; Section 4, describes the use case implemented in CyDronE; Section 5, presents the results; and Section 6, provides the conclusions.

2. Related Work

This section does not restrict its scope to aerial robots or drone-specific platforms, but instead considers ROS 2-based cyber-physical systems more broadly. This choice is deliberate, as ROS 2 provides a common middleware and execution model across heterogeneous robotic platforms, including aerial, ground, and stationary systems. Consequently, security assumptions, vulnerabilities, experimental evaluation practices, and simulation platforms identified in one class of ROS 2-enabled cyber-physical system (CPS) often transfer directly to others. By reviewing a broad spectrum of security testbeds and experimental methodologies at the CPS level, this section captures a wider set of design choices and limitations that are equally relevant to drone-based systems operating under ROS 2.

Security research on ROS 2-based robotic systems has expanded as ROS 2 moved from research prototypes toward deployed CPS, and recent surveys catalog ROS 2 work across multiple domains that include security [Al-Batati et al.(2024)]. Prior work includes formal and empirical analyses of ROS 1 and ROS 2 communication mechanisms, demonstrations of vulnerabilities at the DDS/RTPS middleware layer, and the use of experimental testbeds to evaluate intrusion detection approaches [Yang et al.(2024)]. Collectively, these efforts indicate that unprotected ROS 1 and ROS 2 deployments expose a substantial attack surface, particularly at the DDS/RTPS middleware layer, enabling adversaries to disrupt availability and compromise control integrity in practical settings [DiLuoffo et al.(2019)].

Existing studies on ROS 1 and ROS 2 security testing environments can be grouped into three broad categories. First, several studies evaluate vulnerabilities on real or fully instantiated robotic systems, using physical robot platforms or embedded cyber-physical testbeds to demonstrate practical exploits and quantify their impact [Deng et al.(2022)]. Second, a growing body of research focuses on datasets and monitoring infrastructures

for ROS 2, capturing network, middleware, and host-level telemetry during attacks to support intrusion detection and forensic investigation [Puccetti et al.(2024)]. Third, minimal or synthetic testbeds are used to isolate specific vulnerability classes under controlled assumptions, enabling focused analysis of protocol-level behavior in DDS and ROS 2 communication stacks [Yang et al.(2024)].

While these contributions offer useful insights, many rely on small-scale, single-robot scenarios or manually configured laboratory networks, limiting replication, scalability, and coverage of diverse operational contexts. As highlighted in surveys of small-scale testbeds for connected and automated vehicles and robot swarms, most existing platforms are designed for narrowly scoped experiments and face inherent limitations in scalability, generalization, and long-term sustainability, constraining their use for systematic resilience evaluation and comparative studies [Mokhtarian et al.(2024)]. Adversarial experimentation is often constrained to single attackers, fixed network topologies, or narrow parameter spaces, hindering systematic resilience evaluation and cross-study comparison of defensive mechanisms.

In addition, a substantial portion of prior work depends on outdated software stacks. This includes ROS 1, whose centralized architecture and lack of native security mechanisms limit its suitability for security-critical deployments, as well as early ROS 2 deployments that predate mature DDS security configurations and scalability improvements [Di Luoffo et al.; Zhang et al.(2019; 2025)]. Studies on the scalability of cyber-physical systems in ROS 2 show that increasing the number of agents and communication workloads exposes architectural and resource limitations that are rarely addressed in legacy or small-scale experimental setups [Mañas-Álvarez et al.(2023)].

This persistence of legacy stacks is partly explained by the slow and uneven adoption of ROS 2. Portugal et al. report that while awareness of ROS 2 is high (approximately 95% of surveyed respondents), adoption remains limited: about 42% of participants had never worked with ROS 2, and only around 35% were actively using it at the time of the study. The survey identifies key barriers to migration, including the limited availability of migrated ROS 1 packages, insufficient documentation, and the steeper learning curve associated with DDS-based configuration. Adoption is reported to progress more rapidly in industrial contexts than in academia, where projects remain tightly coupled to legacy ROS 1 workflows, and experienced users often perceive migration as non-urgent due to the effort required to port complex systems. As a result, ROS 1-based systems are expected to remain operational for the foreseeable future, prolonging the coexistence of both middleware generations and reinforcing the gap between modern ROS 2 capabilities and the environments typically evaluated in existing research [Portugal et al.(2024)].

Consequently, results obtained in these environments may not accurately reflect the behavior or threat surface of modern ROS 2 systems. These limitations motivate the need for deterministic, scalable, and adversarially instrumented ROS 2-based CPS environments, built on current ROS 2 distributions and integrated within a reproducible CPS framework, to enable repeatable experimentation under realistic multi-robot (such as swarm of drones) and multi-attacker conditions.

Teixeira et al. developed the DoRIS platform, a ROS 1-based autonomous domestic robot designed to operate in typical indoor home environments. It combines a rich sensing and actuation setup with a distributed computing architecture connected via an embedded wireless network. DoRIS integrates multiple sensors and actuators, including an RGB camera, microphones, range sensors, and drive motors, and is interfaced by

two embedded computers running the robot’s perception, decision-making, and control stacks. The embedded computers communicate over an internal IP network provided by an onboard Wi-Fi router physically mounted on the robot. This router also provides wireless access to external machines, including a dedicated server used for computationally intensive tasks and the attacker’s laptop. The system is built on a ROS 1 architecture, decomposed into ROS nodes running on the embedded computers and the external server, which exchange messages over ROS topics following the standard ROS peer-to-peer computation graph.

Due to the lack of authentication and encryption in the ROS 1 system, the authors implemented an MITM attack to eavesdrop on, alter, and block messages between two nodes of the system without detection. The attack is enabled by prior network reconnaissance that identifies the robot and server endpoints. As a result, the robot executes commands injected by the attacker’s machine instead of responding to the user’s voice commands. The authors further observed that improper manipulation of packet payloads, specifically by modifying ROS message serialization patterns, can trigger segmentation faults in subscriber nodes, causing them to crash. Similar fault-inducing techniques later appeared in ROS 2 attack studies targeting the DDS middleware responsible for inter-node communication.

Puccetti et al. present an intrusion dataset for ROSPaCe, a ROS 2-based Smart Passenger Center (SPaCe) system deployed on a tram carriage. The SPaCe system comprises three components: a remote server that orchestrates the fleet and transmits information to relevant agents; an onboard architecture on each carriage that collects and processes data from camera feeds to enhance service quality and fleet optimization; and user devices that consume the processed information. These microservices implement a video analytics pipeline that processes camera streams to extract semantic and behavioral information about passengers and objects. All components are implemented as ROS 2 nodes running on the Foxy distribution, an earlier long-term-support release that provides publish-subscribe and service abstractions, while a contemporaneous DDS implementation handles message distribution. The full SPaCe software stack runs on a single onboard server, which is connected via a local area network to at least one IP camera and to an attacker’s laptop through an Ethernet switch. The attacker machine shares the same Layer-2 network segment and can therefore inject arbitrary traffic towards the server, emulating an external adversary with direct access to the onboard network.

On the server, three monitoring probes observe different architectural layers during normal operation and under attack. At the network layer, a Tshark-based monitor captures all packets traversing the server’s network interface into PCAP traces. At the operating-system layer, a custom OS Monitor periodically samples selected metrics from the virtual filesystem, such as CPU and memory usage, to capture host-level responses to changing load and attack conditions. Lastly, the ROS 2 middleware layer, together with a dedicated ROS 2 Monitor node, scans the ROS 2 graph to discover nodes and topics and records all publisher–subscriber activity.

The ROSPaCe testbed assumes a single external adversary operating an Ubuntu laptop connected to the same Ethernet switch as the onboard server and with prior knowledge of the server’s IP address. The attacker employs standard penetration-testing tools together with custom Python and Scapy scripts to operate within a ROS 2 threat model that treats the external network and the DDS middleware as primary entry points [Mayoral-Vilches(2022)]. In parallel, a ROS 2-specific reconnaissance technique targets

the middleware by sending specially crafted DDS packets. This approach causes the DDS middleware underlying ROS 2 to reveal information about active nodes and services, enabling remote reconstruction of the ROS 2 computation graph. Together, these reconnaissance actions demonstrate that a ROS 2-based CPS, such as SPaCe, can be profiled at both the network stack and the middleware layer, exposing detailed information that can guide subsequent attacks.

The original ROSPaCe study relies on a single physical attack machine, which constrains the evaluation to single-source adversarial scenarios. As a result, prior ROS 2 security evaluations typically remain limited to single-source denial-of-service scenarios and do not explore distributed flooding under controlled conditions.

After completing system reconnaissance, the ROSPaCe evaluation proceeds to exploit known vulnerabilities, primarily through denial-of-service attacks targeting both the generic network stack and DDS internals [Puccetti et al.(2024)]. At the network layer, high-rate flooding traffic is used to saturate the system's TCP/IP stack. Under these volumetric attacks, ROSPaCe primarily experiences increased CPU usage, socket exhaustion, and reduced network service availability; however, the ROS 2 microservices and monitoring components typically remain operational while the flood is active. In contrast, DDS-level exploits manipulate the semantics of ROS 2 and DDS messages, targeting the middleware rather than the underlying network. These attacks either crash subscriber nodes through malformed middleware messages or manipulate multicast control information to redirect communication toward malicious publishers.

From a vulnerability perspective, ROSPaCe illustrates two complementary aspects of ROS 2 system exploitation. First, classical bandwidth-exhaustion floods degrade network-level availability. Second, small and carefully crafted DDS-level attacks can rapidly crash nodes or hijack communication with comparatively modest traffic, undermining both service continuity and system observability.

In contrast to prior work based on real-world scenarios, Patel et al. present a minimalistic ROS 2 testbed that lacks direct real-world application but follows a comparable attack logic. The study was conducted on a single Windows host running two Ubuntu VMs in the same Layer-2 network segment. One VM acts as the victim, running ROS 2 Foxy, while the other acts as the attacker. Both were configured with static IPv4 addresses, resulting in a flat, unrouted LAN with no broker, proxy, or intrusion-detection system in between. The ROS 2 workload on the victim was deliberately minimal, with a single service/server node that logged each request to a local file and to the console. The attacker VM executed a predefined set of attacks, while the victim VM was acquired using Acquired Volatile Memory for Linux (AVML) to generate a RAM image for forensic analysis of network identifiers and active ROS 2 processes. The authors further specify that DDS-Security was disabled, no custom QoS policies were configured, and the topology did not include additional ROS 2 graph elements such as bridges, sensors, or multiple topics. Using a similar attack logic, Patel et al. expose several weaknesses of a default ROS 2 deployment, rooted in the lack of authentication and isolation in their experimental setup.

Kim et al. present a ROS 2/RTPS attack dataset generated using a purpose-built experimental testbed. The dataset was collected in a single broadcast-domain ROS 2 environment explicitly designed to capture RTPS traffic under tightly controlled conditions. The testbed comprised four main components: a victim robot (a Stella N1 UGV running ROS 2 Foxy with Fast DDS on a Raspberry Pi, equipped with onboard sensors such

as LiDAR), a second Raspberry Pi 4 acting as the legitimate operator, an attacker PC running a Kali Linux virtual machine used for spoofing attacks, and a dedicated router. The router was configured without an Internet uplink, ensuring that the LAN carried only experimental traffic and that packet captures remained uncontaminated. All four nodes operated on the same unrouted Ethernet LAN with static IP addresses, and with DDS-Security plugins and ROS 2 graph-level protections disabled, discovery and data exchange followed the default Foxy/Fast DDS behavior.

In a complementary line of work, Soriano-Salvador et al. propose a Robot Intrusion Prevention System (RIPS) for ROS 2 and evaluate it using a physical robotic testbed rather than a purely virtual or container-based environment. Their evaluation is conducted on a real mobile robot executing benchmark-inspired tasks, with ROS 2 nodes deployed directly on the robotic platform and interacting with onboard sensors and actuators under realistic operating conditions. Unlike VM-centric or dataset-oriented testbeds, this setup emphasizes end-to-end execution in a cyber-physical system, where security mechanisms are assessed in the presence of real sensing, actuation, and timing constraints. However, the experimental environment is tailored to validate the effectiveness of the proposed prevention system, and does not aim to provide a reusable or scalable testbed for systematic exploration of ROS 2 communication-layer vulnerabilities.

While the works discussed above primarily focus on protocol-level security mechanisms and high-level frameworks for protecting ROS 2 and DDS-based systems, they generally assume that the underlying operating system and cryptographic libraries are trustworthy and uncompromised. However, in realistic robotic deployments, adversaries may obtain elevated privileges on the host or physical access to the platform, and thus be able to tamper with user-space components that current designs implicitly trust. Recent work has begun to examine these lower layers of the stack, demonstrating that even correctly configured ‘secure’ ROS 2 deployments can be subverted when the integrity of cryptographic and configuration dependencies is not enforced.

Di Luoffo et al. built two separate experimental systems to study the impact of a compromised cryptographic library. The first system is a pure DDS testbed based on RTI Connex DDS with security enabled and linked against a modified OpenSSL library, with RTPS traffic captured using Wireshark. The second system is a ROS 2 testbed running on Ubuntu, using RTI DDS with security enabled, the ROS 2 Bouncy distribution, and SROS 2 for ROS 2 graph-level security. In this setup, the same instrumented OpenSSL library is installed such that both the DDS security plugins and ROS 2/SROS 2 dynamically link against it. The authors run a standard ROS 2 ‘Hello World’ talker–listener example with SROS 2 enabled, thereby exercising DDS Security under nominal secure communication settings.

Moving beyond network-level attacks, Di Luoffo et al. demonstrate an exploitation scenario in which DDS Security and SROS 2 appear correctly configured, yet the confidentiality of ROS 2 traffic is compromised through tampering with a user-space cryptographic library. Starting from the assumption that an adversary can obtain elevated privileges on the robot’s Linux system, the authors argue that the attacker can replace the OpenSSL libraries with instrumented versions and adjust the corresponding symbolic links. In their validation setup, the original OpenSSL files are overwritten with the modified libraries, and symbolic links - including those used by the RTI DDS security plugin within ROS 2/SROS 2 - are updated to load the altered OpenSSL transparently. Once this substitution is in place, executing the ROS 2 Bouncy ‘Hello World’ example with

SROS 2 enabled causes all DDS Security cryptographic operations to pass through the instrumented library, allowing the modified OpenSSL to record both key material and application data and thereby expose the cleartext messages exchanged between nodes.

Across the surveyed literature, existing works primarily focus on identifying and exploiting vulnerabilities in ROS and DDS-based communication stacks, or on validating specific security mechanisms using either simplified virtual setups or tightly scoped physical platforms. While several studies evaluate attacks in drone-based or robotic scenarios, and others propose simulation environments or cybersecurity-focused datasets, we do not find a testbed that jointly supports large-scale robotic simulation, cybersecurity experimentation, and drone systems within a unified, reusable, and reconfigurable framework. In particular, the combination of scalable simulation, realistic ROS 2 communication behavior, and systematic cyber attack evaluation for drone-centric CPS remains insufficiently addressed, leaving a gap between isolated vulnerability demonstrations and repeatable, security-oriented experimentation at scale. In this work, we aim to address this gap.

3. CyDronE simulation platform

The CyDronE simulation platform integrates Gazebo [Gazebo(2025)], a high-fidelity robotics simulator, with a ROS 2-based control stack, combining them into a realistic, scalable and automated simulated CPS environment that supports multiple functionalities. Its architecture, depicted in Figure 1, is composed of: an *Architecture Coordinator* module, which instantiates the simulation environment based on a set of specifications provided through a YAML configuration file; the simulation environment, which includes devices such as drones, mission controllers, human-machine interfaces (HMI), and instance(s) of Gazebo to simulate the physical behavior of drones and their interaction with the environment; and a *Data Collection* module, to collect data from the simulated physical and cyber domains (e.g., drone positions, velocities, network traffic, CPU, RAM usage, etc.).

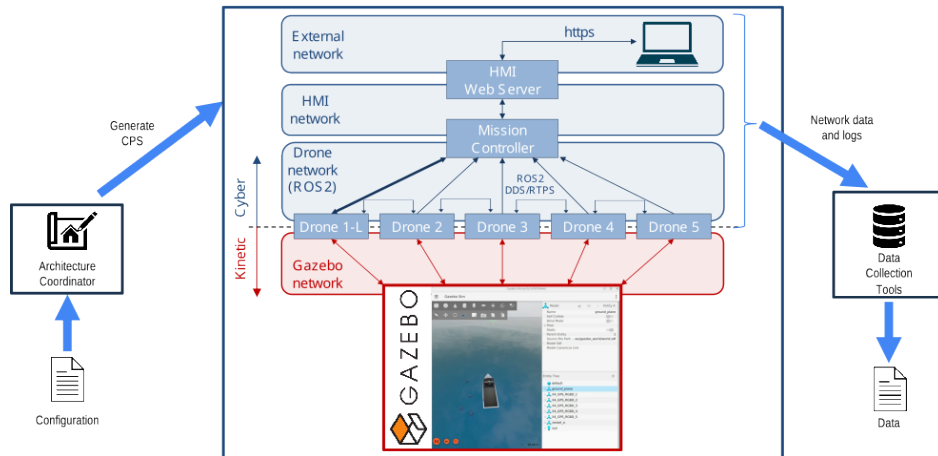


Figure 1. CyDronE architecture.

In this simulation platform, each device shown in Figure 1 runs in a different container, and communicate to each other through virtual communication networks. At the bottom of the simulation environment, it is possible to see Gazebo which runs the simulated world and manages the physics of multiple drones while interfacing with multiple ROS 2 nodes (each one performing the computational process of a drone). All ROS 2 nodes (representing the drones) communicate with Gazebo through the Gazebo Network (highlighted in red), which is inaccessible to the attacker, since it is used solely to enable the exchange of physical-state information with Gazebo. In the real world, this Gazebo Network would not exist, as physical information would be obtained directly through measurements in the real physical environment. We use a ROS 2-Gazebo bridge to translate the Gazebo inputs and outputs into standard ROS 2 topics.

In the upper side of the simulation environment (in blue), we have the simulation of the cyber domain of the drone application scenario. It contains four main components: the drones, communicating with each other and with the Mission Controller using ROS 2 DDS/RTSPS; a Mission Controller that can send/receive commands/data to/from selected drones according to the application characteristics; an HMI Web Server that receives data from the mission controller and makes it available to virtual clients (system operators) through HTTPS; and lastly the client that accesses the HMI Web Server. The drones and the Mission Controller can run on both ROS 2 Humble and Jazzy, though both devices and their interconnecting networks remain accessible to potential attackers.

Notably, the CyDronE platform automates everything end-to-end, from the initial instantiation of the environment to the orchestration of individual components used as modular building blocks. Each component is described by a configuration file that specifies the number of instances, how they interconnect, and in what order to start including other dependencies. From these declarative files the Architecture Coordinator generates runnable stacks that can be launched in parallel, each with its own isolated processes and message spaces so multiple swarms can run side-by-side without interference. Built-in functionalities include coordinated startup/teardown, health checks, telemetry streaming to an HMI and system-metrics capture. This approach allows reproducibility, producing identical isolated environments suitable for control experiments, performance benchmarking, and (our main focus) cybersecurity research. The remainder of this section provide more details on the CyDronE platform.

3.1. Gazebo Simulator

At the lowest level of the system, the Gazebo simulation layer, we run a containerized instance of Gazebo Harmonic. This simulator provides realistic physics, rendering, and sensor emulation—such as gravity, lighting, GPS, and odometry for the drones, as well as the input/output interfaces used by ROS 2 nodes for control and state estimation. The simulation world and all required assets are packaged within the Gazebo container image, ensuring a controlled, reproducible virtual environment. Although the computational process for each drone is executed independently on distinct ROS 2 nodes, as shown in Figure 1, the physics of all drones is simulated on the same Gazebo instance, since these drones would share the same physical world. This yields an environment in which all drones can physically interact steadily with one another and with environmental elements (e.g., water and a small boat).

Because Gazebo is unable to natively speak ROS 2, the `ROS_2_gz_bridge` package provides a bidirectional bridge interface between each ROS 2 node and the simula-

tor. In the upstream communication (Gazebo $\rightarrow$ ROS 2 nodes), Gazebo publishes model ground-truth state, which the bridge exposes as standard ROS 2 topics, consumed by the ROS 2 nodes to obtain pose/velocity measurements. Meanwhile the nodes publish Downstream (ROS 2 $\rightarrow$ Gazebo) velocity commands. The bridge injects these into the corresponding Gazebo models, and the simulator applies them on simulated physical drones.

To mirror real world systems where the drone–environment interface is not subject to application-level interference, we focused during development on maximizing the isolation, as much as possible, between the Gazebo network and the attacker-accessible cyber domain. To achieve this, all Gazebo-related traffic is confined to a dedicated Docker network (`gazebo_net`) and set a `GZ_PARTITION` transport variable that isolates Gazebo’s internal transport space from any other Gazebo instances that might be present on the host. The ROS 2 middleware traffic among the nodes is carried on a separate network, keeping Gazebo communication and ROS 2 communications strictly segregated. Multicast is disabled on `gazebo_net` to reduce unsolicited discovery noise and further minimize interference.

Finally, each drone/ROS 2 container has two Network Interface Cards (NICs) one attached to `gazebo_net` for simulator adjacency while the other one to the ROS 2 network for all ROS 2/DDS communications. By assigning each containerized node dual IP addresses, one for the ROS 2 network and another dedicated to Gazebo Transport, we ensure that traffic remains cleanly separated even though processes coexist within the same containers.

3.2. ROS 2-nodes

In our simulation environment, all drones and the Mission Controller run in containerized ROS 2 nodes. These nodes communicate with each other using the eProsima FastDDS [eProsima(2025)] communication middleware, either to establish inter-drone communication (e.g., for exchanging flocking messages) or to exchange data between drones and the Mission Controller. The ROS 2-based coordination approach is explained in Section 3.2.1, while the details on the use of eProsima FastDDS is provided in Section 3.2.2.

3.2.1. Drone coordination

This project’s ROS 2 layer is organized as a small, well-defined graph of nodes using two supported distributions (Humble & Jazzy). The generic architecture of this ROS 2 layer is composed of a Mission Controller and a set of drones, each of these devices is executed in a different container running a separate ROS 2 process. These processes join a shared DDS domain, and all inter-process communication happens through standard ROS 2 topics and one custom message interface, the `DroneState` message object.

All drones publish their own `DroneState` on a shared topic and subscribe to that same topic to maintain a local view of their neighbors. Each one of these messages comprises three variables: the drone ID; the current position vector of the drone; and finally the velocity vector of the drone. In addition to helping the other drones in the swarm to coordinate their flights, the content of these messages also serves for the Mission Controller to track the swarm and determine actions for selected drones.

The drones’ ROS 2 processes not only establish communication with other drones, but also locally execute the drone navigation algorithms (such as flocking, follow-the-leader, etc.) within the same control loop: the drone ingests its own odometry and the

shared `/drones_states`, runs the navigation algorithm to compute its velocity vector, and then publishes a `Twist` message (i.e., the velocity vector) to its command topic.

3.2.2. *eProsima FastDDS*

At the core of the ROS 2 communication layer used sits eProsima FastDDS [eProsima(2025)], the Data Distribution Service (DDS) implementation middleware defined by the Object Management Group (OMG). It consists of a Data-Centric Publish-Subscribe (DCPS) model for distributed applications with refined Quality of Service (QoS) parameters that regulate the information flow of different aspects of distributed communication.

Every ROS 2 process (i.e., a node) that participates in DDS creates a Domain Participant, which serves as the entry point to a logical boundary known as a domain. This domain blocks all inter-domain communication, and each Domain Participant is associated with exactly one domain from the moment it is created. Each Domain Participant owns a set of DataWriters and DataReaders responsible for sending and receiving data, which are respectively encapsulated by the ROS 2 Publishers and Subscribers within the node. DataWriters and DataReaders under the same ROS 2 Publisher or Subscriber operate independently, sharing no relationship with one another beyond sharing the Publisher/Subscriber's Quality of Service (QoS) settings. Each DataWriter and DataReader is mapped to a specific DDS Topic (named channel with a strictly defined message type) enabling writers to publish data and readers to subscribe accordingly.

However, before any data can be exchanged, the system must first perform self-discovery using the decentralized discovery procedure defined by the RTPS wire protocol. The process begins with participant discovery, in which each participant periodically advertises itself via multicast and shares the network addresses (locators) it listens on. Once two participants detect each other, endpoint discovery starts. During this phase, the matched participants exchange endpoint information—namely, lists of their writers and readers, each associated with a topic and corresponding QoS policies. After discovery is complete, user data is exchanged as RTPS submessages over UDP. This baseline communication includes sequence numbers and heartbeats, with per-writer history caches managed by Fast DDS.

Since all containers bind ROS 2 to the `drone_net` network interface and use the same `ROS_2_DOMAIN_ID`, all ROS 2 discovery and data traffic is confined to that network (i.e., the Drone network shown in Figure 1). In contrast, Gazebo's transport operates on a separate network (the Gazebo network shown in red in Figure 1) and uses a different stack, keeping the traffic isolated and clean.

3.3. *Mission Controller*

The Mission Controller node is the only ROS 2 node not connected to the Gazebo network and, therefore, only has one NIC. The controller's deployment preserves a clean isolation as the controller neither advertises nor consumes Gazebo Transport messages. The Mission Controller ingests the swarm's state exclusively through the ROS 2 topic `drones_states`, maintains a real-time registry of the most recent measurements for all the drones. It leverages this information to manage direct control over the posture of selected drones (e.g., a leader drone, depending on the application), publishing `Twist` messages (i.e., velocity commands) to them.

In parallel, the Mission Controller streams telemetry to a Human–Machine Interface (HMI) server, via a dedicated worker thread, for each `DroneState` received by the Swarm. It enqueues a compact JSON payload, performs the HTTP posts to the configured HMI endpoint without blocking the underlying control loop of the drone application.

3.4. Instantiation and Monitoring

The aforementioned drone environment is implemented on a scalable platform that can efficiently and dynamically deploys multiple instances of the same project in parallel, thanks to the Architecture Coordination module.

To create a concrete experiment, a YAML file containing a high-level specification of the environment is provided to the Architecture Coordinator. This file defines the system topology in a declarative way, specifying which components are used, how many instances of each are required, the logical networks they connect to, and the dependencies between components. Based on this document, the launcher generates a complete and deterministic Docker Compose stack for each instance. It instantiates services from predefined component templates, creates the required dedicated network bridges, and enforces the correct startup order. Because each instance is assigned its own isolated set of networks and service names, multiple swarms can be deployed concurrently on the same host with no risk of cross-talk or resource collision.

The CyDronE platform includes an integrated data-collection component that can be configured and attached to any simulated device. This component captures network traffic (e.g., RTPS flows on `drone_net`) as well as system-level metrics, including per-container CPU and memory utilization and execution logs. A lightweight labeling pipeline hashes and tags captured flows, ensuring that the collected data can be systematically transformed into cybersecurity datasets for drone applications.

4. Use Case

The use case adopted to evaluate CyDronE consists of a scenario where five drones are flocking, guided by a leader drone. During the flight, a drone is targeted by a Distributed DoS (DDoS) attack. The details of the flocking application scenario are provided in Section 4.1, while the implemented DDoS attack is explained in Section 4.2.

4.1. The flocking application scenario

In this use case, a drone is selected to act as the leader of the swarm. As leader, it constantly receives `Twist` messages (i.e., velocity commands) from the Mission Controller and moves according to these messages. The other four drones flock together with the leader using the Boids model proposed by Craig Reynolds [Reynolds(1987)]. Each non-leader drone in the swarm computes its velocity based on three fundamental rules:

- *Separation*: To avoid crowding, drone d_i computes a separation vector s_i according to Equation 1:

$$\mathbf{s}_i = - \sum_{j \in \mathcal{N}_i^s} (\mathbf{p}_j - \mathbf{p}_i). \quad (1)$$

where $\mathbf{p}_i$ and $\mathbf{p}_j$ denote the positions of drone d_i and its neighbor d_j , respectively, and $\mathcal{N}_i^s$ is the set of neighbors within separation radius r_s .

- *Alignment*: To match its velocity with nearby drones, drone d_i computes the alignment vector a_i according to Equation 2:

$$\mathbf{a}_i = \begin{cases} \frac{1}{|\mathcal{N}_i^a|} \sum_{j \in \mathcal{N}_i^a} \mathbf{v}_j, & \text{if } |\mathcal{N}_i^a| > 0; \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

where $\mathbf{v}_j$ is the velocity of neighbor d_j , and $\mathcal{N}_i^a$ includes neighbors within alignment radius r_a .

- *Cohesion*: To remain close to the swarm, drone d_i computes the cohesion vector c_i according to Equation 3:

$$\mathbf{c}_i = \begin{cases} \frac{1}{|\mathcal{N}_i^c|} \sum_{j \in \mathcal{N}_i^c} \mathbf{p}_j - \mathbf{p}_i, & \text{if } |\mathcal{N}_i^c| > 0; \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

where $\mathcal{N}_i^c$ denotes neighbors within cohesion radius r_c .

The velocity of each drone is then computed as a weighted sum:

$$\mathbf{v}_i = w_s \mathbf{s}_i + w_a \mathbf{a}_i + w_c \mathbf{c}_i, \quad (4)$$

where w_s , w_a , and w_c are the separation, alignment, and cohesion weights, respectively. If $|\mathbf{v}_i| > v_{\max}$, the velocity is capped such that $|\mathbf{v}_i| = v_{\max}$. Each drone periodically shares its state $S_i = \{\mathbf{p}_i, \mathbf{v}_i\}$ with neighboring drones every τ seconds through the ROS 2 network shown in Figure 1.

4.2. DDoS attack

In this use case we chose a DDoS attack to demonstrate the CyDronE capability in simulating cyberattacks to drones with cyber and physical realism. We observe the system's response when drones are suffering from an UDP flood attack, leveraging the fact that the RTPS protocol used by DDS typically operates over unreliable UDP/IP transports. To implement this, we use the `hping3` traffic generator [Sanfilippo(2025)], released as free/open-source software (GPL).

The `hping3` can work as a packet generator or network tester/probe engine that talks directly to the IP stack via raw sockets. Typical legitimate uses include testing firewall rules, experimenting with network paths and MTU settings, validating service reachability, studying TCP/IP stack behaviour, and measuring performance and latency in controlled laboratory environments.

The application builds the entire frame payload itself and asks the kernel to inject those bytes on a chosen interface. It operates at OSI layers three and four giving `hping3` bit-level control over fields such as IP, TCP, UDP and ICMP along with payload size and contents. It can craft arbitrary packets with precise control of header fields and then measure the replies it elicits. "By manually constructing packet headers, the tool can implement fragmentation at will and craft rare or malformed combinations that bypass the inherent restrictions of standard socket APIs. On multi-homed hosts (i.e., systems with multiple network interfaces or IP addresses), it binds to a specific interface or source address. This is essential in testbeds like ours that use multiple network bridges, as it ensures deterministic traffic routing and prevents ambiguity in interface selection.

Besides crafting and sending packets, `hping3` listens for any responses the network or endpoint generates and attributes them to the probes it sent. For the TCP mode, typical replies are SYN/ACK (port open) or RST (closed/filtered), for UDP mode (in which the attack will be focused) replies are often ICMP Port Unreachable when a packet hits a closed port, or no reply if the traffic is silently dropped, while for ICMP mode, it expects Echo Replies or diagnostic errors. When explicitly run in traceroute mode, `hping3` performs TTL-limited probing using TCP, UDP, or ICMP packets to infer hop-by-hop paths and interactions with middleboxes.

A core capability of `hping3` is its granular timing control, which facilitate accurate inter-packet spacing. This precision enables constant-rate transmissions, burst firing, gradual rate ramp-up, and flood modes that minimize per-packet overhead. In practice, however, the achievable throughput is limited by operating system scheduling, NIC driver queues, and the CPU time available for user-space packet crafting.

The DDoS attack is instantiated and launched from dedicated attack containers attached to the swarm's ROS 2-facing bridge (`drone_net`). The Docker Compose configuration enables raw sockets and other low-level network operations required by `hping3`. When each container starts, it runs:

```
hping3 -2 <victim-ip> -p <victim-port> --flood -d 1412 --rand-source
```

This command configures `hping3` in UDP mode (`-2`) to transmit a continuous stream (`-flood`) of user-crafted UDP datagrams to the destination port of a specific container IP on `drone_net`. On the specified destination port (`<victim-port>`), the ROS 2 node normally receives RTPS messages containing the positions of the other drones. However, when `hping3` is executed, the node also receives `hping3` packets with a payload size of 1412 bytes (`-d 1412`), which is close to the RTPS message fragmentation limit. The `-flood` option eliminates inter-packet delays, so transmission pacing is determined by the sender's user-space loop, kernel queueing, and network interface scheduling. When combined with the `-rand-source` option, successive datagrams differ at least in the IP source address (and corresponding checksums), and may also vary in the ephemeral UDP source port if this is enabled.

Internally, `hping3` constructs the OSI layer-3 and layer-4 headers in user space, computes the required checksums, and injects the packets through the container's raw socket. Because the container and the target share the same Docker bridge, packet addressing and delivery take place entirely within the same OSI layer-2 broadcast domain.

Therefore, in this use case, each attack container generates one continuous packet stream directed at the selected `<victim-ip>`. Running N attack containers in parallel generates multiple independent packet streams whose aggregate rate at the destination simulates a DDoS attack.

5. Results

This section presents the experimental results obtained from executing the use case described in Section 4 on the CyDronE platform. It also evaluates the platform's performance and scalability when multiple instances of this use case are executed concurrently. The results and analyses are detailed in Sections 5.1 and 5.2, respectively.

All experiments were executed on a machine with the following specifications: an Intel Core i7-13620H (10 cores), an NVIDIA GeForce RTX 4060 GPU with 8 GB of

VRAM, 32 GB of RAM, and a 1 TB SSD. Each drone container (ROS 2 node) was limited to 0.15 CPU and 150 MiB of RAM. In the Gazebo simulation, each airframe was equipped with an HD camera, IMU, pressure sensor, magnetometer, GPS, LiDAR, and an RGB-D sensor. The parameters of the flocking algorithm described in Section 4.1 are: $r_s = 2.50m$; $r_a = 100.0m$; $r_c = 30.0m$; $v_{\max} = 1.0m/s$; $w_s = 1.0$; $w_a = 3.0$; $w_c = 0.5$.

5.1. Attack Effects

In this experiment, five drones operate in a maritime environment using a flocking behavior. The leader drone follows the direction commanded by the Mission Controller via Twist messages, while the remaining four drones maintain formation by flocking with the leader. The Mission Controller commands the leader’s velocity to establish a circular trajectory around a ship.

The simulation’s duration corresponds to approximately 10 minutes, during which the attack is executed twice. To evaluate the impact of the DDoS attack on flocking behavior, we compute, for each drone (excluding the leader), the sum of its distances to all other drones in the swarm. The resulting physical effects are shown in Figure 2, along with indications of when each attack began and ended.

The attack targets Drone 3, overwhelming its communications and causing it to briefly stop in place, as shown in Figure 2 by the first spike in the sum of distances between this drone and the others. The distance summation for the other drones also increases, but to a lesser extent, because they move away from Drone 3 (which stopped due to the DDoS attack) while the distances among themselves remain nearly unchanged. As previously noted, the leader drone follows a circular trajectory, causing the remaining drones in the flock to periodically pass near Drone 3. Consequently, the aggregated distance metric returns to nominal values before exhibiting subsequent sharp increases. This cyclical behavior is observed three times during the first DDoS attack. After waiting approximately two minutes following the end of the first attack, a second attack is launched against the same drone for a shorter duration – roughly the time required for the leader drone to complete one full lap – and it ends as soon as the drones are again close to the targeted drone. Note, especially in the peaks, that sudden drops and increases in the total distance can be observed. This occurs because some legitimate messages may occasionally reach the attacked drone, causing it to briefly resume its trajectory before returning to a static state.

At a technical level, these physical symptoms result directly from a high-rate UDP stream interfering with DDS’s UDP transport and the host’s receive path. Note that each ROS 2 process binds to a small, fixed set of UDP sockets. If an attack floods the victim’s IP and port, the malicious packets follow the same kernel path as legitimate RTPS traffic. As a result, every incoming packet—whether valid or not—must pass through the network interface’s input queue, the operating system’s networking stack, and the process’s message buffers before the middleware can even inspect it.

When the arrival rate exceeds the system’s service capacity – constrained by CPU time, queue size, and the container’s resource limits – queues and buffers begin to accumulate and eventually drop data. Since DDS operates in a brokerless architecture and multiplexes multiple topics over the same UDP endpoints, congestion on this shared transport path can introduce head-of-line blocking. A saturated endpoint carrying nonessential traffic can delay the delivery of all topics within the process, including time-critical discovery beacons.

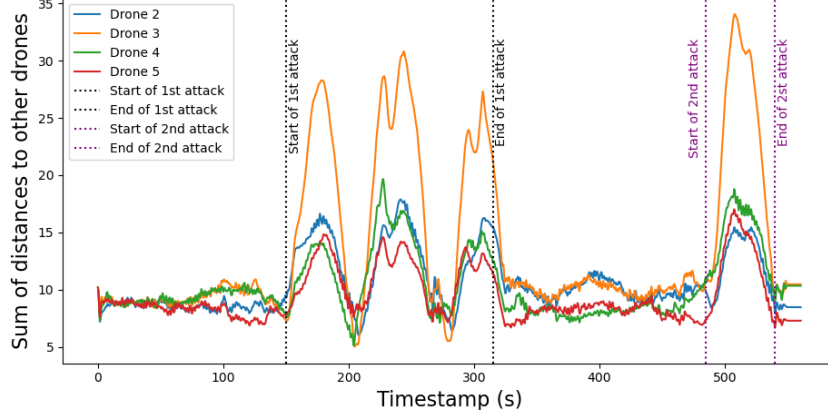


Figure 2. DDoS effect on the physical cohesion of the swarm.

In general, these results show and quantify the impact of the DDoS attack on the swarm’s physical dynamics. During the attack phase, the swarm exhibits a relevant degradation of cohesion. This cyber-induced physical effect is effectively captured by Cy-DronE, highlighting its capability to enable a comprehensive evaluation of cybersecurity experiments in drone swarm applications.

5.2. Performance and Scalability

In this section we evaluate how the platform performs in terms of resource usage when one or more swarm coordination scenarios run in parallel in the same simulation scenario. For this assessment, we begin with a single swarm coordination scenario (i.e., five drones and one Mission Controller) and then progressively increase the number of simultaneous swarm coordination scenarios to two, four, and eight. Since resource consumption may vary depending on the type of attack, this section presents only the resource usage profile under no-attack conditions. These profiles, therefore, serve as baselines onto which the additional resource consumption caused by attacks would be added. In all experiments, we measure CPU, RAM, and GPU usage throughout the simulation. The corresponding plots are shown in Figures 3, 4, and 5, respectively.

Across the three plots, all runs exhibit the same three-act pattern: first, a short launch surge while the world and sensors initialize and DDS discovery converges; second, a long steady-state plateau takes place, where physics, sensors, and ROS 2 timers tick at fixed rate; and third, a clean teardown drop back to baseline as containers exit and drivers free resources. This shape is visible and consistent for every concurrency level (1, 2, 4, and 8 parallel swarm coordination scenarios).

It can be observed in Figure 3 that CPU usage scales almost in a logarithmic growth with the number of swarm coordination scenarios – i.e., the CPU usage approximately increases by a constant amount as the number of swarm coordination scenarios doubles. The load is largely dominated by Gazebo’s physics and sensor-processing threads, rather than by the relatively lightweight ROS 2 control loops. A single simulation stabilizes at around 35% CPU usage; two simulations at approximately 50%; and four simulations at about 70%, with periodic ripples caused by synchronized sensor bursts and visualization.

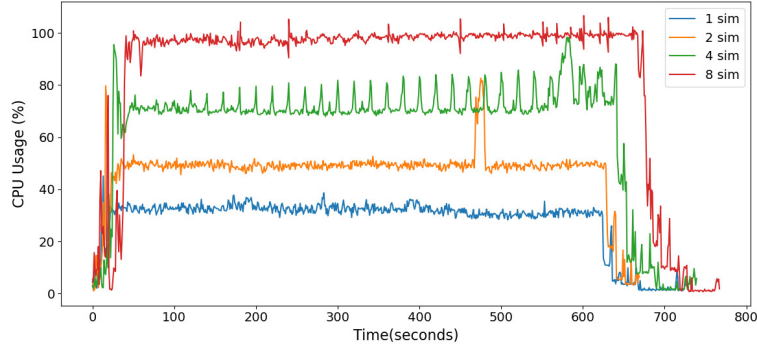


Figure 3. CPU usage for different numbers of concurrent swarm coordination scenarios.

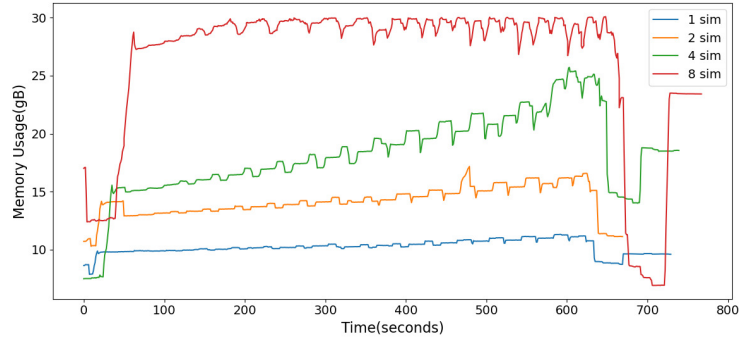


Figure 4. RAM usage for different numbers of concurrent swarm coordination scenarios.

Running eight simulations pushes CPU usage to 95–100%, making the CPU the first hard bottleneck, despite a per-drone CPU cap of 0.15 each.

As shown in Figure 4, RAM usage rises predictably in a staircase pattern as assets and sensor buffers warm up. The growth across different swarm coordination scenarios also follows an approximately logarithmic pattern, similar to the CPU usage case, nearly reaching saturation with 8 swarm coordination scenarios.

The GPU VRAM usage, in turn, increases almost linearly, approximately doubling when the number of swarm coordination scenarios doubles, as shown in Figure 5. This occurs because the physics of each swarm coordination scenario is simulated in a separate Gazebo instance, as the swarms do not share the same physical environment. As a result, each scenario requires its own set of render targets, depth buffers (e.g., for RGB-D and LiDAR), and related resources, leading to the observed linear increase in GPU memory usage.

In summary, under the defined resource limits and sensor load, the system behaves as expected for a sensor-heavy Gazebo + ROS 2 setup. It shows brief initialization spikes followed by stable steady-state performance and scales well as concurrency increases. GPU usage reflects a cost driver, while CPU and RAM usage grow predictably with each additional swarm due to physics simulation, sensor processing, and DDS overhead. Architectural isolation keeps control loops realistic from the perspective of the cyber

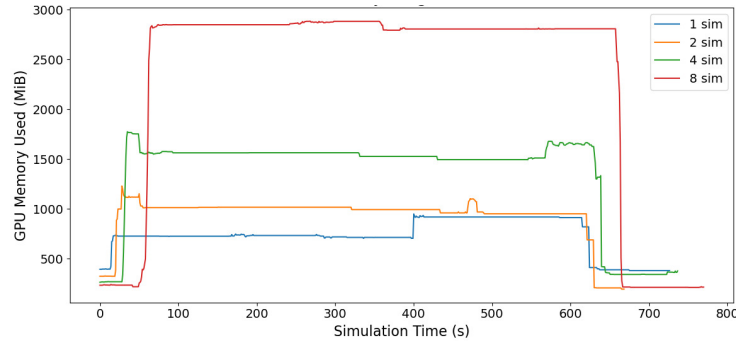


Figure 5. GPU VRAM usage for different numbers of concurrent swarm coordination scenarios.

domain, enabling reproducible measurements, safe parallel execution, and scalable drone cybersecurity experimentation.

6. Conclusion

The experimental results show that CyDronE fills a key gap in existing approaches by providing an integrated cyber–physical simulation platform for cybersecurity research in drone swarms. By combining a containerized ROS 2 control stack, high-fidelity Gazebo simulation, and network isolation, the platform enables scalable and observable experiments in which cyber attacks produce measurable physical effects. This allows accurate evaluation of cyber threats and their kinetic impact in both military and civil contexts, without the cost or risk of real-world testing.

The DDoS experiments demonstrate CyDronE’s effectiveness as a cyber–physical assessment tool by showing how attacks on DDS/RTPS communications propagate through the ROS 2 middleware and disrupt swarm coordination, degrading motion continuity and formation stability. These results validate CyDronE as a suitable environment for evaluating system-level security impacts and emphasize the need to assess drone cybersecurity at the system level rather than in isolated software components.

Finally, the performance and scalability analysis confirms that CyDronE can sustain increasing levels of concurrency while preserving stable and predictable behavior. The results show that CPU and RAM consumption grow in a controlled manner as additional swarm instances are deployed, with clear and well-defined bottlenecks emerging only at high levels of parallelism. GPU usage scales linearly with the number of simulations, reflecting the expected cost of high-fidelity physics and sensor rendering. Overall, these findings demonstrate that CyDronE provides a scalable foundation for repeatable cybersecurity experimentation, supporting the execution of multiple, isolated drone swarm scenarios on a single host without compromising realism or experimental integrity.

7. Acknowledgement

This work is funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the LASIGE Research Unit, ref. UID/00408/2025, DOI <https://doi.org/10.54499/UID/00408/2025>.

References

- Abdulrahman S. Al-Batati, Anis Koubaa, and Mohamed Abdelkader. 2024. ROS 2 Key Challenges and Advances: A Survey of ROS 2 Research, Libraries, and Applications. *Preprints* (October 2024). doi:10.20944/preprints202410.1204.v1
- Gelei Deng, Guowen Xu, Yuan Zhou, Tianwei Zhang, and Yang Liu. 2022. On the (In)Security of Secure ROS2. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*. ACM, Los Angeles, CA, USA, 1583–1597. doi:10.1145/3548606.3560681
- Vincenzo Di Luoffo, William R. Michalson, and Berk Sunar. 2019. Credential Masquerading and OpenSSL Spy: Exploring ROS 2 Using DDS Security. *arXiv preprint arXiv:1904.09179* (2019). doi:10.48550/arXiv.1904.09179
- Vincenzo DiLuoffo, William R. Michalson, and Berk Sunar. 2019. Credential Masquerading and OpenSSL Spy: Exploring ROS 2 using DDS security. *arXiv preprint arXiv:1904.09179* (2019). <https://arxiv.org/abs/1904.09179> arXiv:1904.09179 [cs.CR, cs.RO].
- eProsima. 2025. Fast DDS Documentation. <https://fast-dds.docs.eprosima.com/en/stable/>. Accessed 2025.
- Gazebo. 2025. Gazebo Simulator Homepage. <https://gazebo.org/home>. Accessed March 2025.
- Dong Young Kim, Dongsung Kim, Yuchan Song, Gang Min Kim, Min Geun Song, Jeong Do Yoo, and Huy Kang Kim. 2024. RTPS Attack Dataset Description. *arXiv* (2024). doi:10.48550/arXiv.2311.14496
- Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. 2022. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics* 7, 66 (2022), eabm6074. doi:10.1126/scirobotics.abm6074
- Victor Mayoral-Vilches. 2022. Robot Hacking Manual (RHM). <https://rhm.cybersecurityrobotics.net> (2022). doi:10.48550/arXiv.2203.04765
- Victor Mayoral-Vilches, Ruffin White, Gianluca Caiazza, and Mikael Arguedas. 2022. SROS2: Usable Cyber Security Tools for ROS 2. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 11253–11259. doi:10.1109/IROS47612.2022.9982129
- Francisco José Mañas-Álvarez, María Guinaldo, Raquel Dormido, and Sebastian Dormido-Canto. 2023. Scalability of Cyber-Physical Systems with Real and Virtual Robots in ROS 2. *Sensors* 23, 13 (2023). doi:10.3390/s23136073
- Armin Mokhtarian, Patrick Scheffe, Maximilian Kloock, Simon Schäfer, Heeseung Bang, Viet-Anh Le, Sangeet Ulhas, Johannes Betz, Sean Wilson, Spring Berman, Amanda Prorok, and Bassam Alrifae. 2024. A Survey on Small-Scale Testbeds for Connected and Automated Vehicles and Robot Swarms. (2024). doi:10.13140/RG.2.2.16176.74248/1
- Open Robotics. 2025. ROS Discourse. <https://discourse.openrobotics.org>. Official ROS-related news, accessed 2025.
- Zina Oudina, Makhlof Derdour, Ahmed Dib, and Mohammed Mounir Bouhamed. 2024. Empirical analysis of the security threats and risks that drones face, represent, and mitigation. In *2024 6th International Conference on Pattern Analysis and Intelligent Systems (PAIS)*. IEEE, 1–8.

- Raymond C. Parks and David P. Duggan. 2011. Principles of Cyberwarfare. *IEEE Security and Privacy* 9, 5 (2011), 30–35. doi:10.1109/MSP.2011.138
- Yash Patel, Parag H. Rughani, and Dhruvil Desai. 2023. Analyzing Security Vulnerability and Forensic Investigation of ROS2: A Case Study. In *Proceedings of the 8th International Conference on Robotics and Artificial Intelligence* (Singapore, Singapore) (ICRAI '22). Association for Computing Machinery, New York, NY, USA, 6–12. doi:10.1145/3573910.3573912
- David Portugal, Rui P. Rocha, and João P. Castilho. 2024. Inquiring the robot operating system community on the state of adoption of the ROS 2 robotics middleware. *International Journal of Intelligent Robotics and Applications* (2024). doi:10.1007/s41315-024-00393-4
- Tommaso Puccetti, Simone Nardi, Cosimo Cinquilli, Tommaso Zoppi, and Andrea Ceccarelli. 2024. ROSPaCe: Intrusion Detection Dataset for a ROS2-Based CyberPhysical System and IoT Networks. *Scientific Data* 11 (2024). doi:10.1038/s41597-024-03311-2
- Tommaso Puccetti, Simone Nardi, Cosimo Cinquilli, Tommaso Zoppi, and Andrea Ceccarelli. 2025. *ROSpace Dataset*. https://github.com/TommasoPuccetti/rospace_dataset GitHub repository, accessed 2025.
- Craig W. Reynolds. 1987. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. Association for Computing Machinery, New York, NY, USA, 25–34. doi:10.1145/37401.37406
- ROS. 2024. ROS Homepage. <https://ros.org>. Accessed September 2024.
- Salvatore Sanfilippo. 2025. hping3: Packet Crafting Tool. <https://www.kali.org/tools/hping3/>. Accessed April 2025.
- Enrique Soriano-Salvador, Francisco Martín-Rico, and Gorka Guardiola Múzquiz. 2024. Implementing a Robot Intrusion Prevention System (RIPS) for ROS 2. arXiv:2412.19272 [cs.RO] <https://arxiv.org/abs/2412.19272>
- Luís Sousa, José Cecílio, Pedro M. Ferreira, and Alan Oliveira de Sá. 2024. Reconfigurable and Scalable Honeynet for Cyber-physical Systems. *Ada Lett.* 44, 1 (Dec. 2024), 90–94. doi:10.1145/3706601.3706617
- Rafael R. Teixeira, Igor P. Maurell, and Paulo L. J. Drews. 2020. Security on ROS: Analyzing and Exploiting Vulnerabilities of ROS-Based Systems. In *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*. 1–6. doi:10.1109/LARS/SBR/WRE51543.2020.9307107
- wireshark. [n.d.]. Wireshark Official page. <https://www.wireshark.org> Accessed: 03-2025.
- Shuo Yang, Jian Guo, and Xue Rui. 2024. Formal Analysis and Detection for ROS2 Communication Security Vulnerability. *Electronics* 13, 9 (2024). doi:10.3390/electronics13091762
- Aaron Yu, Iuliia Kolotylo, Hashim A. Hashim, and Abdelrahman E. E. Eltoukhy. 2025. Electronic Warfare Cyberattacks, Countermeasures, and Modern Defensive Strategies of UAV Avionics: A Survey. *IEEE Access* 13 (2025), 68660–68681. doi:10.1109/ACCESS.2025.3561068
- Jiawei Zhang, Yang Liu, Xinyu Chen, and Haoyu Wang. 2025. Enhancing the Resilience of ROS 2-Based Multi-Robot Systems via Kubernetes Edge Orchestration. *Sensors* 25, 16 (2025), 5067. doi:10.3390/s25165067