AIAA SciTech Forum
12-16 January 2026, Orlando, FL
AIAA SCITECH 2026 Forum

10.2514/6.2026-1797

Networked Simulation for Cybersecurity Evaluation of Small Unmanned Aircraft Systems in Dense Urban Environments

Abel Diaz-Gonzalez, Austin Coursey, Bryce Bjorkman, Daniil Shatokhin, Cailani Lemieux-Mack, Noah Dahle, Abenezer Taye, Robert Canady, Xenofon Koutsoukos, Gautam Biswas, and Bryan C. Ward
Vanderbilt University, Nashville, TN, USA

Integrating unmanned aerial vehicles (UAVs) into urban areas presents significant opportunities for fast delivery, traffic monitoring, and surveillance to support disaster response. However, existing simulators inadequately address critical elements like Remote Identification broadcasting, multi-operator coordination, and cyberattack modeling, all essential for realistic urban deployment. We introduce a new networked simulation platform for multiple fleets of small unmanned aircraft systems (sUAS) in urban settings, with each fleet managed and flown by different operators. We assume ArduPilot controls each UAV. We simulate the broadcast dynamics of the drone Remote Identification, required by drones in the U.S., in dense urban areas using OMNet++. Our simulation platform offers 2D visualization with QGroundControl and 3D visualization with Gazebo. As a distributed simulation, it scales for numerous drones and operators and allows for realistic simulation of cyberattacks. This simulator can address challenges in operating large fleets of UAVs flown by multiple operators in urban environments, such as cybersecurity.

I. Introduction

Unmanned aerial vehicles (UAVs) will play an essential role in the future of urban environments. Small UAVs are being used for applications such as delivery [1], traffic monitoring [2], situational awareness, and delivery for disaster management [3]. Larger UAVs, like electric vertical take-off and landing vehicles (eVTOLs), are being considered for passenger transportation for urban air mobility [4]. Applications of UAVs in urban environments have moved beyond academic research settings. For example, companies like Wing* and Zipline† are developing drone delivery systems. Looking toward the future, NASA has announced its Advanced Air Mobility Mission [5] that categorizes applications of UAVs into urban air mobility, low-altitude local missions, and regional air mobility. Research in vehicle [6] and vertiport design [7, 8] has emerged to support air mobility. In low-altitude missions, such as those flown for package delivery, medical supply delivery, or surveillance in urban environments, small UAVs are heavily impacted by wind [9] and limited battery capacity [10]. Additionally, small unmanned aircraft systems (sUAS) face unsolved challenges such as flight planning, flight replanning, cybersecurity, and more.

While sUAS hold great promise for urban applications, addressing and validating solutions for their safe and smooth operations remains challenging. In dense urban areas, numerous small UAVs may fly crisscrossing trajectories simultaneously for delivery and pickup. Therefore, decisions can be complex. For example, a UAV suddenly choosing to reroute may be reasonable in rural environments where there is ample airspace. However, in urban settings, traffic density complicates the introduction of new trajectories and the rerouting of existing ones. Arbitrary rerouting can result in collisions or, at the very least, increase congestion, slowing down all UAVs in that region. Additionally, as a new technology, sUAS face cybersecurity concerns. To develop reliable and safe sUAS operations and ensure flight safety in urban environments, we need a comprehensive simulation testbed to simultaneously model multiple UAVs alongside air traffic management infrastructure. This simulation testbed must be able to model multiple fleets, each controlled by independent operators, that are operating concurrently in the same region of urban airspace. It must be able to study multiple configurations and communication scenarios while developing effective traffic management algorithms, model cyberattacks, and create schemes to make sUAS operations resilient to various threats.

Currently available simulators are limited in scope. Many simulators focus on the dynamics and control of multiple UAVs [11–14]. While these are useful for developing control algorithms for UAV swarms or fleets, they do not consider realistic communication between UAVs or ground control stations. When low-level control of individual UAVs is not a focus, many simulators utilize ArduPilot [15], a state-of-the-art UAV autopilot software, to control simulated UAVs. For

*https://wing.com
†https://www.flyzipline.com

example, FlyNetSim [16] employs ArduPilot's basic software in the loop (SITL) simulator to model the dynamics of multiple UAVs. ArduPilot manages each UAV, and the communication between UAVs and the ground control station is simulated using NS-3 [17], a network simulator. ArduSim [18] utilizes ArduPilot's SITL simulator for dynamics simulation and ArduPilot for control. UAV-to-UAV communication is simulated using UDP with three options to model which UAVs should receive messages.

These simulators often connect ArduPilot to Gazebo [19] for more realistic drone and environmental dynamics and 3D visualization. For instance, [20] employs Gazebo to mimic wireless communication. They connect ArduPilot to Gazebo/ROS to simulate multiple UAVs. In [21], they also connect ArduPilot to Gazebo, simulating communication between UAVs and the ground station using NS-3. While these simulators implement realistic control logic, dynamics simulations, and network simulations for communication, which are all necessary for real sUAS, they lack important components that will be needed in urban environments in the near future. For instance, there will be multiple UAV operators with their own "ground control stations." This will lead to complex planning and airspace allocation problems that must be resolved. UAVs must also adhere to regulations, such as broadcasting their Remote ID [22] in the United States, a feature not directly supported by these simulators. Finally, the systems and communication channels of sUAS in urban environments face cybersecurity concerns. To preemptively defend against cyberattacks and determine the potential impacts of threats, such as Remote ID spoofing, we need to simulate these attacks, a feature missing in current simulators.

To address the limitations of current simulators and support the evaluation and development of sUAS, this paper introduces a networked simulation of small unmanned aircraft systems in urban environments. Our simulator supports multiple UAVs flown with ArduPilot controllers, controlled by several operators in a city. These operators propose and validate flight plans using an external plan approver. Once approved, the UAVs execute their prescribed mission. During flight, the UAVs broadcast their Remote ID, allowing nearby UAVs to receive critical information from neighboring UAVs belonging to different operators. The Remote ID broadcast is simulated using OMNeT++, a network simulator, to model the dynamics of the signal in a dense urban environment with tall buildings. Our simulator utilizes both the ArduPilot SITL dynamics simulator with QGroundControl as a 2D visualization and the Gazebo simulator and 3D visualization as options for the user. The simulator is distributed, enabling scalability in the number of drones and operators, along with easy hardware-in-the-loop simulations. It is designed to implement a realistic multi-UAV and ground station communication network that enables the study of cyberattacks in small unmanned aircraft systems.

The contributions of this work are as follows.

1) We develop an open-source, realistic networked small unmanned aircraft systems simulator that accommodates a flexible number of UAVs and operators and implements the remote identification standard.
2) We demonstrate how the simulator can be utilized to study challenges in sUAS, such as cybersecurity.

The remainder of this paper is organized as follows. Section II introduces the multi-UAV, multi-provider urban scenario that motivates the simulator. Section III details the simulator architecture and its main components. Section IV describes how to access the simulator and how to define and run scenarios using it. Section V presents a multi-UAV simulation demonstration and qualitative analyses of additional scenarios relevant to cybersecurity and urban operations. Section VI discusses next steps and extensions to the simulator. Finally, Section VII concludes the paper.

II. Scenario Description

This simulator aims to go beyond simple multi-UAV simulations by considering realistic UAV operation applications that include multiple ground control stations in urban environments. To achieve this, we start by outlining the scenario, referencing NASA and FAA ConOps and protocols, and making assumptions about future sUAS operations only when necessary to enhance the simulator's realism and completeness. Together, these elements shape the design of our simulator. A visualization of an example scenario covered by our simulation environment is shown in Fig. 1.

Our scenario involves multiple operators flying UAVs in the same airspace in altitudes under 400 feet, closely following the FAA UAS traffic management (UTM) ConOps [23]. An operator owns several UAVs and is responsible for developing plans, communicating with their UAVs, adhering to regulations, and ensuring safe operations. These operators may have cloud-based or physical ground control stations, represented as colored stars in Fig. 1. The operators communicate only with the UAVs they own, using the MAVLink protocol [24, 25], over 5G. Each UAV follows a prescribed flight plan composed of a sequence of waypoints corresponding to real applications such as aerial inspection or medical delivery.

Flight plans must be pre-approved to ensure that the airspace remains de-conflicted for all UAVs and their operators sharing the space. In traditional aircraft, airspace is operated by a centralized authority, such as air traffic controllers. In

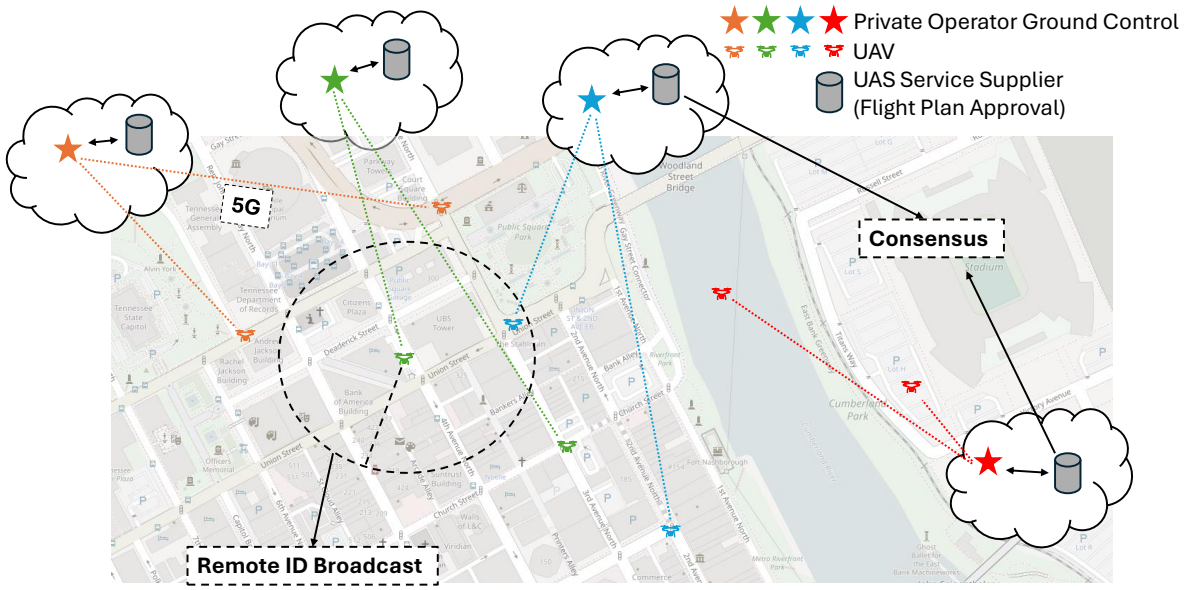


Fig. 1 Example urban multi-UAV simulation scenario. This example includes eight UAVs, two belonging to each private operator with a remote ground control station. The operators communicate with their UAVs using MAVLink over 5G. Each UAV broadcasts its Remote ID locally over WiFi or Bluetooth. The operators propose flight plans to an external flight approver to deconflict flights.

UAS operations, the FAA aims to replace this with one or more UAS service suppliers [23]. This is an industry partner that serves as a bridge between the FAA and UAS operators. UAS operators communicate their intended flight plans with the service supplier, which considers the FAA constraints and flight plans of other operators to notify the operator of constraints and potential conflicts with the plans of other operators. In the case of a conflict, the operator must work with the service supplier and other operators to resolve the conflict.

This flight plan approver could operate in a centralized or decentralized mode. In the centralized mode, it maintains a single database of flight plans and only requires a single node for flight approval. While this is efficient and closely follows the FAA ConOps [23], it raises some challenges. For example, the flight approval process relies on a single node. This lack of redundancy can make the airspace more vulnerable from a cybersecurity perspective. To mitigate the problems of a single point of failure and cooperative flight planning, we also consider the possibility of the flight approver using a decentralized flight approval process, such as using a *hashgraph* distributed ledger [26]. In this case, the service supplier enhances the security of its flight approval mechanism by increasing the number of nodes that maintain a copy of the flight plans.

During a flight, drones must transmit their Remote Identification. This is a message consisting of the elements presented in Fig. 2, copied verbatim from the FAA regulation § 14 CFR 89.305 [22].

This message must be broadcast at least once per second on a radio frequency that is readable by personal devices. Some of the signals broadcast, such as the position and velocity of the UAV, may be utilized by neighboring UAVs for tasks like collision avoidance, particularly for small UAVs lacking high-quality sensors. Since the Remote ID broadcast can influence the behavior of surrounding UAVs, we also model it in our simulation scenario, as shown for a single UAV in Fig. 1. Importantly, we also simulate the signal dynamics so that only UAVs within range receive the broadcast.

III. UAV Simulator

The development of our current networked small Unmanned Aerial System (sUAS) simulator comprises various distributed Python processes and standard drone software processes that communicate using either User Datagram Protocol (UDP) or Transmission Control Protocol (TCP). These processes encompass those associated with the Unmanned Aerial Vehicle (UAV), including the ArduPilot controller and onboard logic, as well as the ground control station logic, dynamic simulations, and visualization. Additionally, an environment oracle is incorporated to simulate

Remote Identification Message Requirements

(a) The identity of the unmanned aircraft, consisting of:
  (1) A serial number assigned to the unmanned aircraft by the person responsible for the production of the standard remote identification unmanned aircraft; or
  (2) A session ID.
(b) An indication of the latitude and longitude of the control station.
(c) An indication of the geometric altitude of the control station.
(d) An indication of the latitude and longitude of the unmanned aircraft.
(e) An indication of the geometric altitude of the unmanned aircraft.
(f) An indication of the velocity of the unmanned aircraft.
(g) A time mark identifying the Coordinated Universal Time (UTC) time of applicability of a position source output.
(h) An indication of the emergency status of the unmanned aircraft.

Fig. 2 Remote Identification Message Requirements

the dynamics of the remote identification broadcast signal. An overview of the simulator's architecture is presented in Fig. 3. In this section, we go through the technical details of each component of this architecture.

A. Unmanned Aerial Vehicles (UAVs)

In our simulation environment, each Unmanned Aerial Vehicle (UAV) is represented using two primary processes: (1) its onboard logic and (2) its ArduPilot controller. The onboard logic is a custom Python module that receives MAVLink messages from the ground control station as well as environmental information from the simulation oracle, and then performs high-level decision-making. For example, when the oracle issues a Remote Identification (Remote ID) alert indicating a potential collision, the onboard logic evaluates the situation and generates an avoidance maneuver. In the current simulator, this is implemented through a simple geometric collision-avoidance algorithm that computes an immediate safe waypoint. The onboard logic issues MAVLink commands to ArduPilot using the `pymavlink` library over a dedicated TCP connection. In cases where no replanning or additional decision-making is required, such as at the start of a flight, the onboard logic simply passes the ground control station's commands to ArduPilot. Internally, all commands flow through an intermediate proxy process, which handles message routing between the onboard logic and the UAV's ArduPilot instance.

ArduPilot processes these commands, which may originate from the onboard logic or the ground control station, and generates control outputs. These outputs are transmitted to the dynamics simulation via a separate UDP connection. The subsequent state updates are returned to ArduPilot, which forwards them to the onboard logic. The onboard logic then disseminates the updated state to the simulation oracle and the ground control station.

B. Ground Control Stations

Ground Control Stations (GCS) serve as the operational logic on the operator's side, responsible for managing a fleet of Unmanned Aerial Vehicles (UAVs). Each GCS operates as an independent Python process, acting as the primary interface for submitting plans, replanning missions, and high-level coordination with its affiliated fleet. Operators are presumed to hold authority solely over their designated UAVs and interact exclusively with them.

In the simulation, each GCS establishes a direct connection to the onboard logic of its corresponding UAVs through MAVLink messages transmitted over designated UDP ports. These interactions are managed programmatically utilizing the `pymavlink` Python library, which facilitates autonomous flight plan uploading, telemetry parsing, and command dispatching. This configuration allows for responsive, distributed, and scalable control of multi-agent simulations, independent of graphical tools such as QGroundControl.

In addition to UAV coordination, ground control stations approve flight plans through the flight approval process described in Section III.E. Once a flight is approved, the GCS launches the corresponding UAV. Fig. 3 illustrates the GCS role within the system architecture, showing its connections to UAV onboard logic processes and the flight approval process.

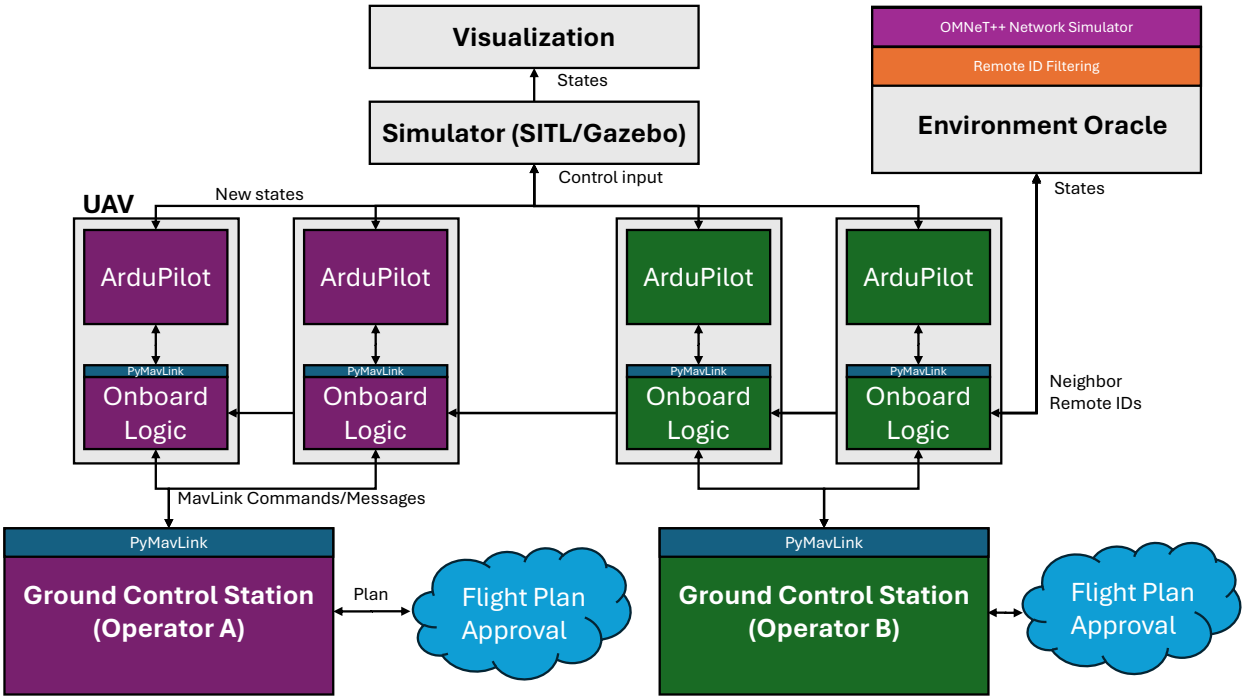


Fig. 3 Technical architecture of networked small unmanned aircraft systems simulation. Each block is a separate process, and connections are network connections using UDP/TCP, allowing distributed simulation. Shown for 2 operators with remote ground control stations with 2 UAVs each.

C. Dynamics Simulation and Visualization

The simulator supports three visualization modes: (1) Gazebo, (2) QGroundControl, and (3) No Visualization. Gazebo provides a fully rendered 3D environment with terrain, physics, and collision modeling, making it suitable for debugging, demonstrations, and realistic closed-loop testing. In contrast, QGroundControl is a lightweight 2D ground control interface for telemetry monitoring, flight path tracking, and operator supervision.

1. Gazebo

The Gazebo visualization is tightly integrated with ArduPilot [15] through a dedicated plugin, `ardupilot_gazebo` [27], to create a closed-loop control architecture. ArduPilot functions as the flight controller, executing low-level control laws, interpreting sensor data, and generating motor commands. Gazebo models the UAV's flight dynamics, environmental interactions, and onboard sensors such as GPS, IMU, and barometer. Thus, Gazebo serves as the visualization and the dynamics simulator.

Upon initiating the ArduPilot software-in-the-loop simulation, two UDP ports are established for each unmanned aerial vehicle (UAV): port 9002 is designated for receiving sensor data from Gazebo, while port 9003 is utilized for transmitting actuator commands. The `ardupilot_gazebo` plugin automatically connects to these ports, thereby creating a closely integrated feedback loop between control and physics:

1) Gazebo simulates sensor data based on the UAV's state and environment.
2) This data is transmitted to ArduPilot via port 9002.
3) ArduPilot estimates the current state and computes motor commands.
4) Commands are sent back to Gazebo over port 9003.
5) Gazebo applies these commands to update the UAV's physical state.

This control loop does not use MAVLink. Instead, it relies on a custom binary protocol optimized for high-frequency, deterministic updates. MAVLink is used separately for mission-level communication such as telemetry, commands, and plan uploads, as described in Section III.B.

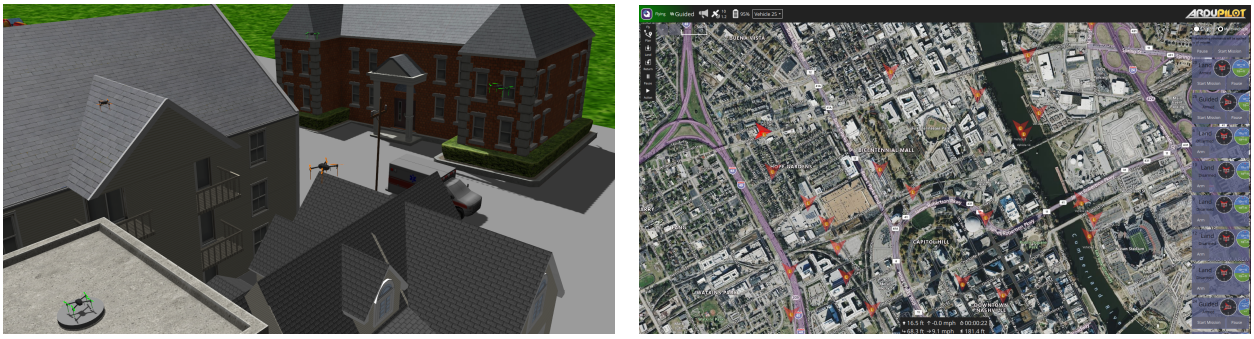


Fig. 4 Gazebo simulation (left) and QGroundControl interface (right).

2. QGroundControl

When using QGroundControl, Unmanned Aerial Vehicles (UAVs) function within isolated virtual environments devoid of any physical interactions, wherein variations in environmental conditions, collisions, or inter-agent interactions are not represented. In contrast to Gazebo, it does not replicate the physical characteristics of the environment; rather, the movement of each UAV is calculated internally by the built-in Software In The Loop (SITL) dynamics engine of ArduPilot, with every UAV operating within its own distinct simulation realm.

In this configuration, QGroundControl establishes a direct connection to each ArduPilot instance via TCP. In scenarios involving multiple UAVs, QGroundControl is capable of managing each unit independently by connecting to sequentially incremented port numbers (e.g., 5763, 5773, and so on), thereby facilitating the parallel monitoring of multiple agents. This mode is particularly appropriate for lightweight and scalable deployments, where visual feedback is beneficial, yet 3D visualization is not requisite.

Fig. 4 illustrates both visualization modes. On the left, Gazebo showcases five UAVs in a shared, physics-enabled environment with color-coded ownership (e.g., orange and green). On the right, QGroundControl presents telemetry overlays and flight paths for individual UAVs, facilitating real-time monitoring without environmental interaction.

3. No Visualization Mode

When simulating complex scenarios for data collection, such as cyberattacks in large-scale urban environments, visualizing each UAV may cause an unnecessary computational burden. In these settings, the simulation can run without visualization, simulated equivalently to the QGroundControl visualization scenario described above. In Section V, we demonstrate the impact of this visualization choice on the simulator's scalability.

D. Environment Oracle

Given that the Unmanned Aerial Vehicles (UAVs) are simulated and not actually broadcasting their Remote ID, we simulate the Remote ID transmission utilizing an environmental oracle. This oracle possesses awareness of the states of each UAV. It acquires information from the onboard logic of each UAV through a network connection. The oracle simulates the broadcast of each Remote ID transmission using the OMNeT++ network simulator [28]. This models the dynamics of the Remote ID signal in urban environments. It simulates signal dynamics in complex scenarios with multiple moving UAVs and environmental obstacles, like buildings in urban environments.

E. Flight Approval

Prior to the commencement of a flight, it is imperative for each operator to secure approval for their flight plan. In urban environments, this approval will likely come from a UAS service supplier, as represented in Fig. 1 and Fig. 3. This service supplier could operate either a centralized or decentralized approval process. In the centralized approval model, all operators (ground control stations) will communicate their proposed plans to a singular centralized authority. This authority will either accept or reject the proposed flight plans to guarantee a fair allocation of the airspace. In the event of a rejection, the authority will likely either respond with an alternative, or the operator will need to resubmit.

Conversely, in the decentralized approval model, flight approval will not fall onto a single entity. Instead, the service supplier could oversee a distributed ledger technology, such as a hashgraph. This hashgraph would encompass flight plans and their respective timestamps. Each ground control station could host a node of this hashgraph, allowing it to

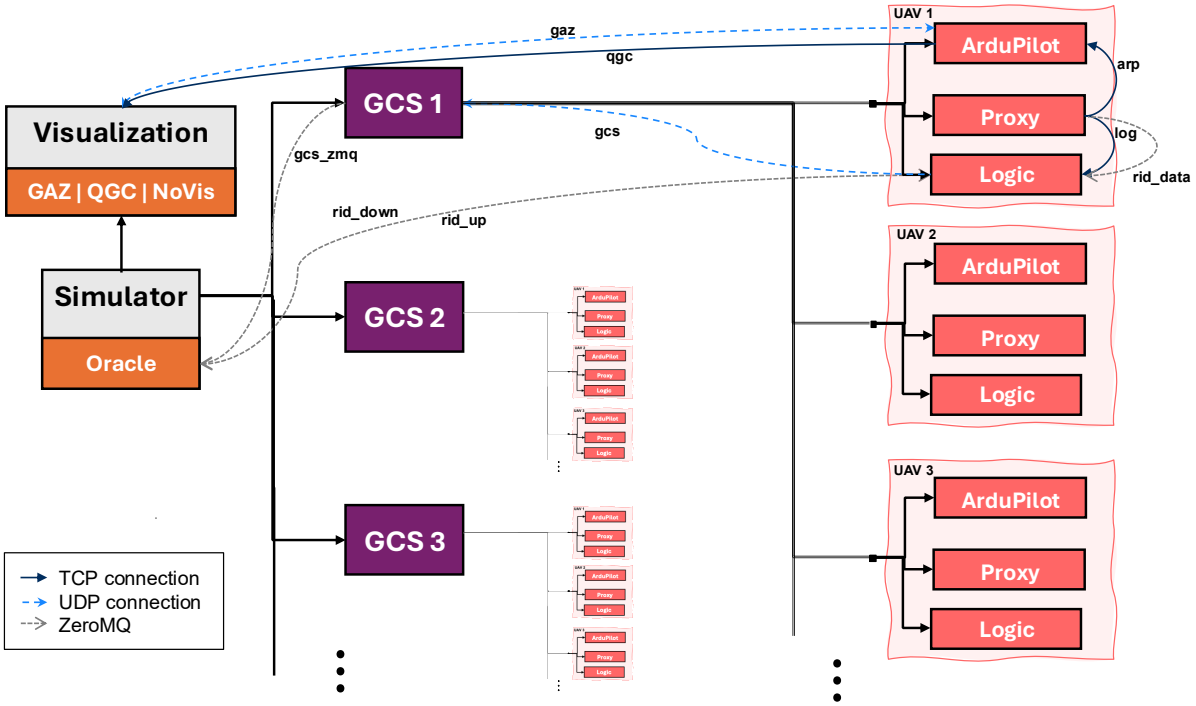


Fig. 5 Simulator network connectivity overview.

assess scheduled flights for conflicts before proposing a new one. Should there be no discrepancies, the operator would propose the addition of their plan to the hashgraph at the current timestamp. If a consensus is reached by two-thirds of the other operators (as in [26]), the proposed plan would be accepted. Each operator would have to put up a financial stake to join the hashgraph. If the operator acts maliciously (by disapproving valid flights or approving invalid flights), their stake would be automatically slashed. A decentralized flight approval service supplier can encourage open and fair cooperation among operators, not needing to rely on a single centralized authority for decision-making. We encourage future work or users of our simulator to consider this option.

In the current version of the simulation, we approve every proposed UAV flight. Since we have full control over each ground control station's logic, we only propose flights that are conflict-free. In more complex scenarios, flight approval may be required. For example, this may be necessary when cyberattacks cause emergency replanning or when ground control stations dynamically choose flight scenarios. In future work, we aim to implement both a centralized and decentralized approval mode.

F. Network Connectivity

In the simulator, processes are distributed and networked. Each process has an address, allowing most components to be simulated across different devices. Aside from improved scalability, this allows for hardware-in-the-loop experiments or real flight tests. A real drone could interact with the simulated drones and ground control stations by establishing a network connection to the other processes in the simulator.

To provide better clarity about the types of network connections in the simulator beyond the high-level overview in Fig. 3, Fig. 5 illustrates the network connectivity. For simplicity, only the connections for the first GCS and UAV are shown, but the same pattern extends to all remaining GCSs and UAVs in the system. Each box in the diagram corresponds to a distinct independent process, and the arrows indicate process-launch relationships.

In the simulator, we consider three types of connections: TCP connections, UDP connections, and connections using the `ZeroMQ` Python library. TCP and UDP connections are used for connections between processes that will be present in a real networked UAV scenario. For example, the ground control station communicates MavLink commands to the UAV through UDP. Conversely, `ZeroMQ` connections are used for internal simulation transmission that will not be

necessary in real settings, e.g., the environment oracle.

As illustrated in Fig. 5, the ground control station communicates to the oracle on the `gcs_zmq` connection. The oracle also has a two-way connection with each UAV's onboard logic on the `rid_up` and `rid_down` links to gather information needed for the Remote ID simulation (e.g., the drone's position) and transmit any received broadcasts from nearby drones. Each ground control station communicates with its UAVs using MavLink messages over UDP. Internally, each UAV uses TCP connections to communicate among the onboard logic, proxy, and ArduPilot processes. It also uses `ZeroMQ` connections to pass data from the proxy to the onboard logic for Remote ID broadcast along the `rid_data` link. Please note that since the simulator is still being developed, the configuration and types of connection are subject to change. However, Fig. 5 should provide intuition in the use of each type of network connection.

IV. Using the Simulator

Source code for the simulator is currently hosted at `https://github.com/4belito/uav-cyber-sim`. As the simulator is still being developed, some details on its usage or installation may change. The README file of this repository will contain updated instructions[‡].

A. Installation

Since the simulator uses versions of software specific to the Ubuntu operating system, we containerize the simulator within a Docker environment. This automatically installs and manages all dependencies inside an isolated environment that can be run on any standard operating system. Both the networked UAV simulator presented in this paper and the OMNeT++ Remote ID simulation are contained in Docker environments, with installation instructions presented at the bottom of the README.

B. Defining and Running a Scenario

The simulator repository contains Jupyter Notebook (`.ipynb`) files with demos showcasing simple scenarios with a single UAV and ground control station, all the way up to hundreds of UAVs broadcasting their Remote IDs and controlled by multiple ground control stations. To showcase how to use the simulation, we annotate the code snippets needed to run a 5 UAV simulation using the Gazebo simulator below. Additional code, such as boilerplate imports needed to run the following code blocks, is available in the notebook files.

Before creating the Gazebo visualizer, we specify the simulation origin `gra_origin`, a fixed latitude–longitude–altitude point with a heading that ties the simulation to a specific location on Earth for ArduPilot. We also define the corresponding ENU (East–North–Up) origin, typically set at $(0, 0)$ with the same altitude and heading. While `gra_origin` provides the geodetic anchor for the simulation, the ENU origin represents the same point expressed in a local, abstract coordinate system that is easier to interpret and manipulate. All UAV homes defined in ENU coordinates are interpreted relative to the simulation origin `enu/gra_origin`.

```
gra_origin = GRAPose(lat=-35.3633280, lon=149.1652241,alt=0,heading=0)
enu_origin = ENUPose(x=0, y=0, z=gra_origin.alt, heading=gra_origin.heading)

base_homes= ENUPose.list([  # east, north, up, heading
    (5, 5, 0, 0),
    (10, 0, 0, 0),
    (-5, -10, 0, 0),
    (-15, 0, 0, 0),
    (0, -20, 0, 0),
])
```

Then, we can define the waypoints each UAV must follow. The following code uses a helper function to generate square paths for each UAV relative to its home.

[‡]Although we recommend using the latest version for improved features or bug fixes, to get the specific version of the simulator used in this paper, download the latest commit from December 2025.

```
base_paths = [Plan.create_square_path(side_len=slen, alt=5) for slen in (5, 7, 4, 3, 2)]
```

Then, we define system IDs and colors for each drone and build a vehicle object representing each UAV with its plan.

```
sysids = [1,2,3,4,5]
colors = 2*[Color.BLUE]+3*[Color.GREEN]

vehs:list[SimVehicle] = []
for sysid, base_home, base_path, color in zip(sysids, base_homes, base_paths, colors):
    # Define the location to save waypoints
    mission_path = DATA_PATH / f"mission_{sysid}.waypoints"
    # Define the vehicle's plan
    plan = AutoPlan(
        name="simple_auto_plan",
        mission_path=str(mission_path),
    )

    plan.save_basic_mission_from_relative(
        sysid=sysid,
        gra_origin=gra_origin,
        relative_home=base_home,
        relative_path=base_path,
    )

    # Create a vehicle with an ID, plan, ground control station, and more
    veh = SimVehicle.from_relative(
        sysid=sysid,
        gcs_name=f'{color.name}_{color.emoji}',
        plan=plan,
        color=color,
        enu_origin=enu_origin,
        relative_home=base_home,
        relative_path=base_path,
        model="iris",
    )
    vehs.append(veh)
```

Next, we specify Gazebo as the simulator with a custom runway map as the environment that the UAVs operate in.

```
    gaz = Gazebo(gra_origin,world_path="simulator/gazebo/worlds/runway.world")
```

Finally, we establish the simulator object to launch the simulation with the specified parameters, visualize the UAV flights, and collect data.

```
simulator = Simulator(
        visualizer=gaz,
        terminals=['gcs'],
```

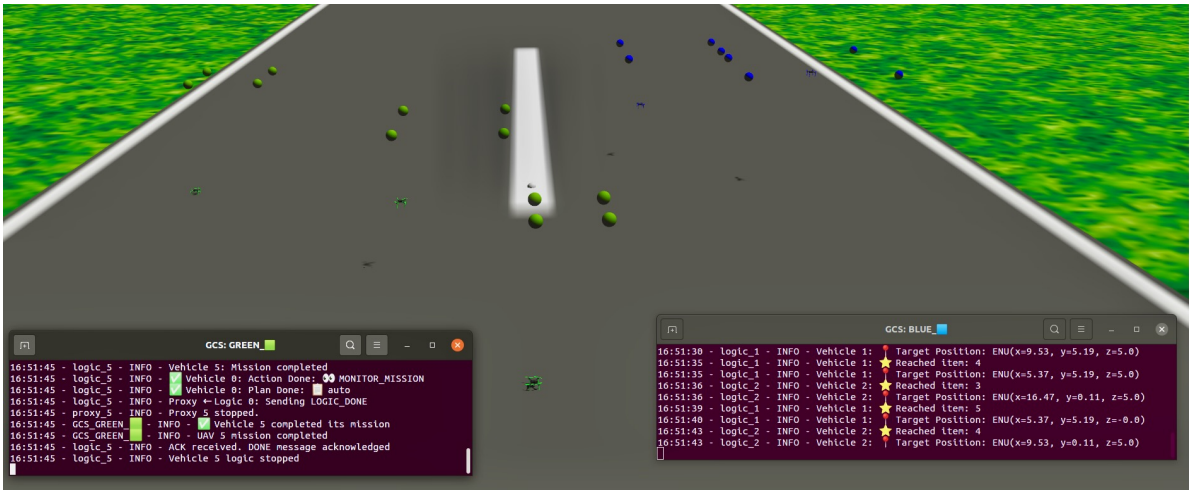


Fig. 6 Overview of the 5-UAV simulation scenario in Gazebo. Each colored sphere represents a UAV's waypoint trajectory in the custom runway environment. Colors indicate the GCS or provider responsible for each UAV. The image is a screenshot captured during the simulation.

```
        verbose=1,
)
for veh in vehs:
    simulator.add_vehicle(veh)

orac = simulator.launch()
orac.run()
```

Figure 6 illustrates the resulting multi-UAV scenario in Gazebo. Five vehicles are spawned at their designated ENU homes and begin executing their waypoint trajectories over the custom runway environment. The snapshot highlights how the simulator renders multiple UAVs in a shared physics-enabled world, including their relative spacing, headings, and altitude separation. This provides an example of how user-defined ENU coordinates, plans, and the global `gra_origin` translate into a fully realized 3D mission in Gazebo.

Users of the simulation can modify the parameters, such as the UAV waypoints, simulator type, or environment map, to quickly and easily define custom multi-UAV experiments. While running a simulation, all ground control stations will automatically log CSVs of all data from their drones during the simulation. By specifying what data to log, the simulator can be used for data collection for training of data-driven machine learning models. Additionally, the simulator generates comprehensive logs of internal processes that can be used for algorithm or scenario debugging.

C. Data Logging for Cyberattack Detection

Beyond defining and running multi-UAV scenarios, the simulator also provides comprehensive logging capabilities. These features operate automatically during every execution and complement the usage workflows described above.

The system records detailed data streams for all software components involved in a simulation. This includes sensor messages, MAVLink message exchanges, logic-level decisions, mission files, and operator configurations. Every message is timestamped and aligned across processes, which yields a complete description of each run independent of the chosen visualization mode.

Real-world collection of cyber–UAV interaction data is expensive, difficult to reproduce, and often unsafe when intentional attacks are performed. The simulator offers a controlled and repeatable environment where both nominal and adversarial conditions can be generated on demand. Because all messages and decisions are logged with precise timing and consistent labels, the resulting datasets are well suited for supervised and self-supervised learning methods.

The attack mechanisms described in Section VI.A can be enabled during any simulation run, allowing the system to generate paired sets of nominal and corrupted behaviors. Since the simulator records both the injected adversarial signals

and the resulting UAV responses, each execution naturally produces labeled examples of attack onset, propagation, and effect. This makes the simulator an effective tool for producing benchmark datasets for cyberattack detection, diagnosis, and resilience evaluation in networked sUAS operations.

V. Simulation Demonstrations

A. Multi-UAV Gazebo Simulation with Real-time Data Collection

To illustrate the capabilities of our simulation, we executed a straightforward simulation involving five Unmanned Aerial Vehicles (UAVs) utilizing the Gazebo dynamics simulator. We ran this simulation using the code outlined in the previous section. Of the five drones, two were operated by a blue operator, while the remaining three were managed by a green operator. The UAVs were assigned compact square flight plans and instructed to initiate their takeoff simultaneously. The networked architecture presented in Figure 3 was executed on a single 48-core Linux machine. The resulting execution logs are shown below.

Simulation Log

- Vehicle 1 launched (PID 1324608)
- Vehicle 1 logic launched (PID 1324609)
- Vehicle 2 launched (PID 1324610)
- Vehicle 2 logic launched (PID 1324611)
- Vehicle 3 launched (PID 1324612)
- Vehicle 3 logic launched (PID 1324613)
- Vehicle 4 launched (PID 1324614)
- Vehicle 4 logic launched (PID 1324615)
- Vehicle 5 launched (PID 1324616)
- Vehicle 5 logic launched (PID 1324617)
- UAV logic 1 is connected to Oracle
- UAV logic 2 is connected to Oracle
- UAV logic 3 is connected to Oracle
- UAV logic 4 is connected to Oracle
- UAV logic 5 is connected to Oracle
- UAV logic 1 is connected to GCS blue
- UAV logic 2 is connected to GCS blue
- UAV logic 3 is connected to GCS green
- UAV logic 4 is connected to GCS green
- UAV logic 5 is connected to GCS green
- Vehicle 4 terminated
- Vehicle 5 terminated
- Vehicle 3 terminated
- Vehicle 1 terminated
- Vehicle 2 terminated

From the logs, we can see that each vehicle, represented as an ArduPilot instance, along with its onboard logic, is initiated as a distinct process. Subsequently, the onboard logics establish a connection to the environment oracle responsible for facilitating Remote ID broadcasting. Each UAV logic interfaces with its corresponding ground control station. The flights are then carried out, and their processes are terminated when done. During the flight, the onboard logic relays its status to the operator. Figure 7 illustrates 3D representations of the UAV positions obtained by each operator in real-time throughout the flight. As depicted, an operator receives information solely from the UAVs they possess, which mirrors the realistic communication expectations typical of actual sUAS applications. This data can be inspected during flight for replanning and decision-making. It can be used post-flight for model training and simulation analysis.

B. Video Analysis

Next, we qualitatively analyze videos captured using the simulator to show the simulator's ability to model complex multi-UAV situations relevant to urban operations.

1. UAV Collision

UAVs flying in urban airspaces will operate near other UAVs or obstacles like buildings. With this in mind, Fig. 8 shows frames of a video from the Gazebo simulator with two drones flying toward each other. In Fig. 8b, the drones collided, and in Fig. 8c, the leftmost drone lost altitude. The Gazebo simulation option in the network UAV simulator is well-suited for modeling scenarios like this. It can be used by operators and researchers for applications such as testing

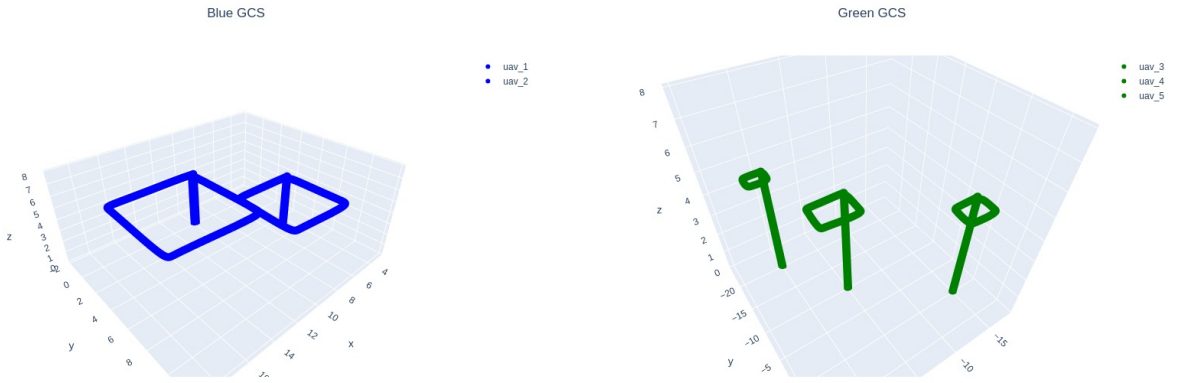


Fig. 7 Plot of UAV positions recorded by blue (left) and green (right) ground control stations during a simulation of 5 UAVs flying in the same Gazebo world. The ground control stations receive messages from their UAVs.

(a) Before collision. **(b) At collision.** **(c) After collision.**

Fig. 8 Images of Gazebo simulation with two UAVs colliding. The images are presented in sequential order and demonstrate the use of the Gazebo simulation option for complex scenarios.

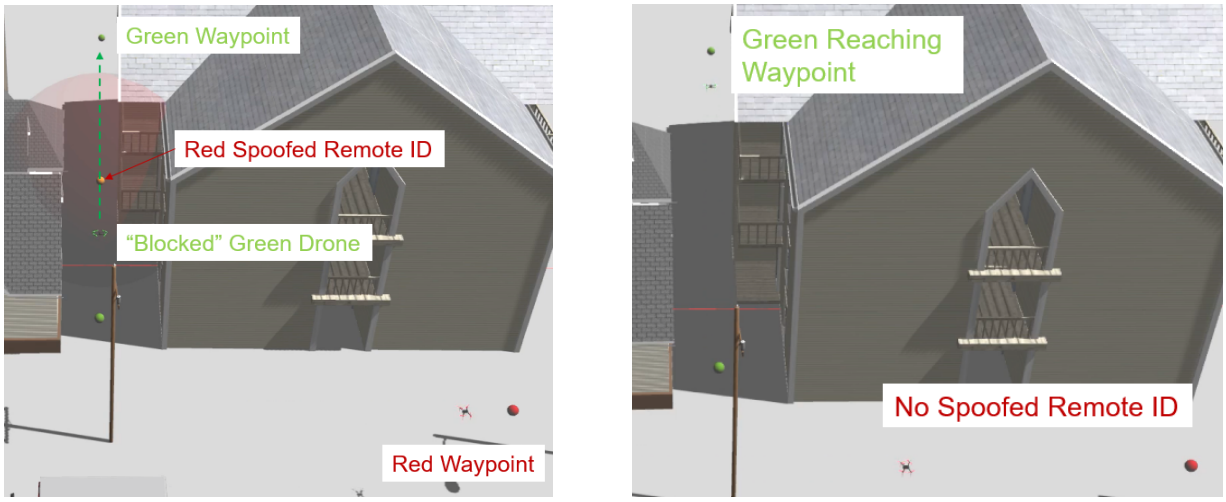
collision avoidance algorithms or recreating scenarios of interest, such as accidents. By simulating scenarios using the more realistic physical models present in Gazebo, the safety and reliability of UAVs in urban settings can be studied in a controlled setting.

2. Remote ID Spoofing Attack

As the simulator models realistic network communication between sUAS software processes, such as ArduPilot, a ground control station, and a UAV's onboard logic, one of its primary use cases is for cybersecurity research. The simulator architecture also supports the modeling of additional cyberattacks, such as man-in-the-middle, denial-of-service, and jamming attacks, as discussed in Section VI.A. To demonstrate the capabilities of the simulator to model cyberattacks, we implemented one such attack, *Remote ID spoofing*. In this scenario, two drones are flying to their respective nearby waypoints, a red drone and a green drone. The green drone plans to fly between two buildings to reach its destination. While flying between the buildings, the green drone observes a spoofed Remote ID from the red drone claiming that the red drone is directly in front of the green drone, blocking its narrow passageway between the buildings. Fig. 9 shows the results of this attack in the simulation. In Fig. 9a, the attack was present, so the green drone could not reach its waypoint and had to find another way around the building. In Fig. 9b, the attack was not present, and the green drone could reach its waypoint as normal. Operators and researchers can use the simulator to study different types of cyberattacks. Since the simulator uses real network communication, these attacks can also be physically executed on the simulation.

3. Large Scale Fleet Operation

As the simulator is distributed, its QGroundControl and No Visualization modes can be used for large-scale fleet operation. In Fig. 10, we show one such example using the QGroundControl option with dozens of UAVs flying above the city of Nashville, Tennessee. These UAVs were commanded independently by their ground control stations and are at various stages of their flight. This is representative of what the airspace could look like in the near future when using



(a) Scenario with red Remote ID spoofed. **(b) Scenario without red Remote ID spoofed.**

Fig. 9 A simple urban scenario with (a) and without (b) a Remote ID spoofing attack. The red drone's Remote ID is spoofed such that it blocks the green drone from flying between buildings, delaying the green drone's mission and forcing it to waste energy.

unmanned aerial systems for various urban applications.

C. Scalability Study

The simulator is designed to support experiments that range from detailed evaluation of a few vehicles to large multi-UAV scenarios involving hundreds of agents. Although scalability is the primary objective, high-fidelity rendering remains valuable for small-scale testing and for examining behaviors that depend on geometry, sensing, or environmental interactions. Because the computational cost of visualization is the dominant factor affecting performance, we assessed real-time scalability under three visualization configurations on a single computer with an AMD Ryzen Threadripper 3960X 24-core processor and two NVIDIA GeForce RTX 3060 graphics cards.

Gazebo offers the richest physical and visual detail, including 3D rendering and environment interaction, but this fidelity limits real-time operation to roughly five UAVs on our test hardware. As shown on the right of Fig. 11, the speed of the simulation on a single computer drops dramatically as the number of UAVs increases in Gazebo, only supporting around 5 UAVs in real time. When visualization is removed entirely, the simulation relies solely on the internal SITL dynamics, which significantly reduces per-UAV overhead and supports real-time execution with approximately 240 vehicles. QGroundControl provides an intermediate option by offering a lightweight 2D map-based visualization. In our system, QGroundControl is used strictly as a passive viewer, while all control logic is executed in Python. Its low rendering cost enables real-time performance for approximately 70 to 80 UAVs, providing visual situational awareness without the expense of 3D simulation. Together, these results highlight the trade-off between visualization fidelity and multi-UAV scalability.

VI. Discussion

As the simulator continues to develop, several extensions can further expand its capabilities and realism. One area of particular interest is the systematic modeling of cyberattacks, which the architecture is already well suited to support.

A. Future Cyberattack Modeling Capabilities

The simulator already demonstrated a Remote ID spoofing attack in Section V, but its architecture supports a much broader family of cyberattacks. These attacks were not evaluated experimentally in this paper; however, the communication interfaces in Fig. 5 expose clear points where they can be instantiated in future work. Below, we outline several classes of attacks and how they can be implemented within the current framework.

- **Denial of service.** This attack can be created by overwhelming the `gcs` communication link that delivers MAVLink

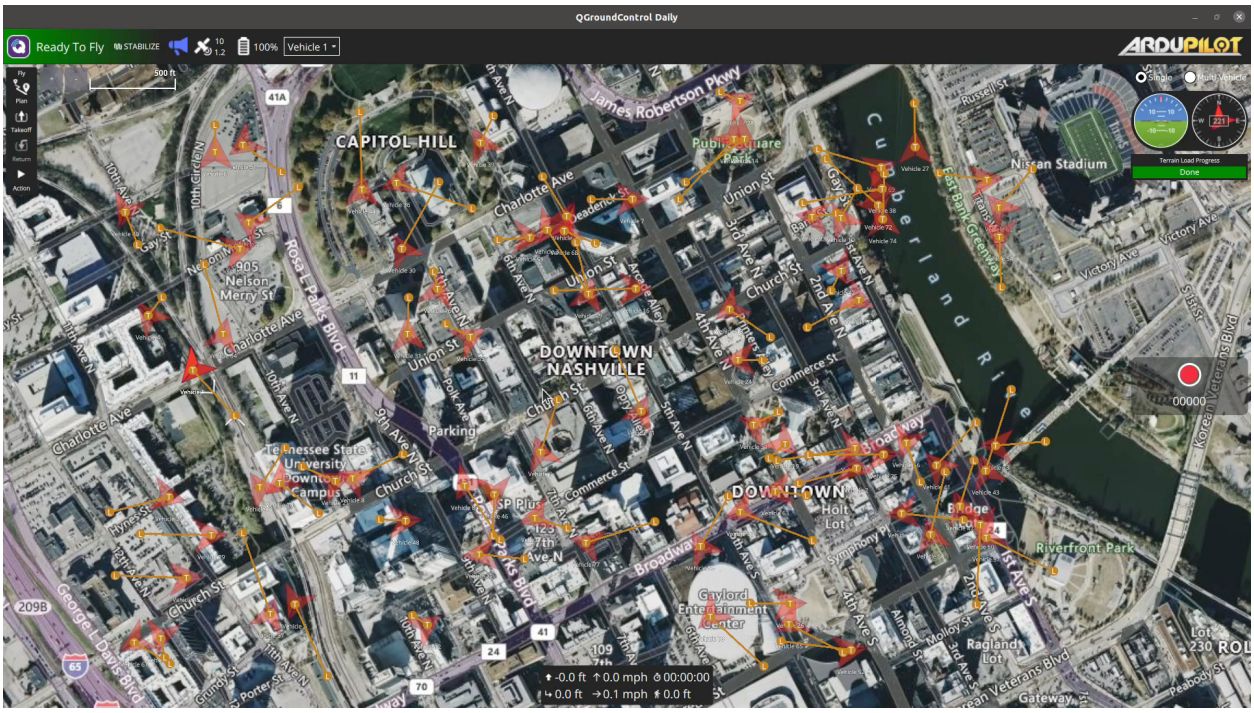


Fig. 10 Large-scale urban UAV fleet visualization using the QGroundControl visualization option in the simulator.

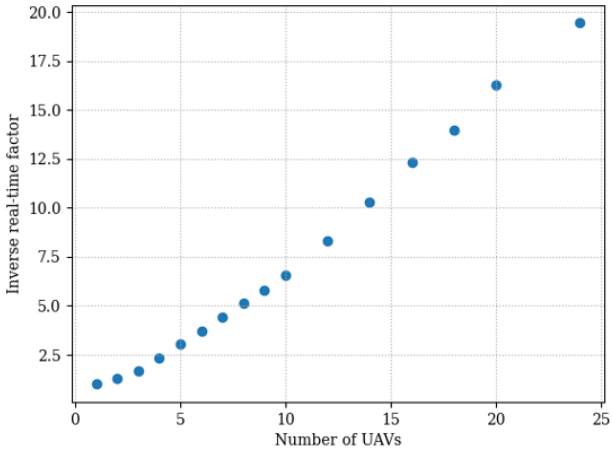
commands from an operator to a UAV. Injecting a high rate of irrelevant or duplicate packets causes delayed or missing commands, reproducing the effect of saturating the command-and-control channel in real systems.

- **Remote ID spoofing.** Demonstrated in Section V, this attack is implemented by inserting falsified Remote ID messages into the `rid_up` channel. Nearby UAVs then react to these fabricated broadcasts as if they were legitimate, allowing controlled experiments on the effect of misleading neighborhood information.
- **Man in the middle.** A man-in-the-middle attack can be modeled by inserting an intercepting process along the `gcs` link between an operator and a UAV. This intermediary can delay, modify, or reorder MAVLink messages, enabling studies of how corrupted command traffic influences flight behavior.
- **GPS or position spoofing.** In real systems, this attack is performed by transmitting counterfeit satellite signals, causing a UAV to mis-estimate its own position. In the simulator, the equivalent effect can be created by modifying the position values received by the onboard logic from its `log` TCP connection to the proxy. A small module placed just upstream of the logic replaces true latitude, longitude, and altitude with fabricated values, leading the UAV to make decisions based on an incorrect self-location.
- **Jamming.** Jamming can be represented by dropping, delaying, or corrupting packets on the `gcs` link. This models radio-frequency interference that prevents operator commands from reaching the UAV, allowing controlled experiments on degraded or intermittent connectivity.
- **Arbitrary control injection.** An attacker may impersonate an operator and send unauthorized MAVLink commands. Writing injected commands directly onto the `gcs` link allows forced mode changes, insertion of new waypoints, or overrides of mission logic.
- **Malicious input.** Malicious inputs include corrupted mission files provided to a ground control station or malformed MAVLink packets injected on links such as `gcs` or `log`. This enables studies on robustness to unexpected or adversarial data during operation.

Together, these attacks illustrate how the simulator can support rich cybersecurity evaluations in future extensions. Since all communication interfaces are exposed and logged, each attack can be precisely controlled, repeated, and analyzed.

Visualization Mode	Max # UAVs
Gazebo	~ 5
QGroundControl	~ 80
No Visualization	~ 240

(a) Maximum UAVs supported in real time.



(b) Gazebo inverse real-time factor vs. number of UAVs.

Fig. 11 Scalability under different visualization modes. The Gazebo plot shows that 3D rendering restricts real-time performance to only a few UAVs. QGroundControl visualization supports tens of vehicles, and the no-visualization execution mode scales to hundreds on a single computer.

B. Broader Simulator Extensions

Beyond cybersecurity evaluations, the simulator offers numerous opportunities for expanding functionality and realism. These enhancements would improve the fidelity of the simulated environment and broaden the scope of research applications that the system can support.

- **Flight planning and replanning.** Real urban sUAS operations require rigorous pre-flight planning and dynamic replanning during emergencies. Incorporating these capabilities, potentially via a UAS Service Supplier interacting with a ground control station (Fig. 1), would more accurately reflect real operational workflows.
- **Hardware-in-the-loop experiments.** Because the simulator operates over the network and does not require all processes to run on the same machine, it can support hardware-in-the-loop experiments. Real drones may connect to the simulation as if they were simulated agents, enabling validation of algorithms on physical platforms.
- **Additional UAV sensing technologies.** Current regulations require only Remote ID broadcasts, but UAVs increasingly employ sensors such as cameras or LiDAR. Incorporating these into the simulator would enhance capabilities for obstacle avoidance and more complex mission tasks such as package delivery or emergency landing.

VII. Conclusion

In this paper, we presented a networked simulation framework for small unmanned aircraft systems operating in urban environments. The simulator comprises distributed Python processes that communicate via UDP/TCP MAVLink messages, enabling realistic interactions among operators, onboard logic modules, and the UAVs themselves. An operator initiates a mission by sending a flight plan to the UAV's onboard logic, which then executes the mission using the ArduPilot controller. ArduPilot forwards control signals to the multi-UAV dynamics simulator, either its built-in SITL engine or Gazebo, and continuously reports UAV states back to the onboard logic. These states are relayed to the operator's ground control station for real-time monitoring and to an environment oracle, which uses OMNeT++ to model and broadcast remote ID signals with realistic wireless dynamics. Visualization is supported through QGroundControl for mission-level 2D views and Gazebo for high-fidelity 3D rendering.

Our simulator captures a realistic multi-operator UAS scenario in which multiple UAVs share airspace, broadcast Remote ID information, and coordinate flight plans through a service provider. We demonstrated the system's capabilities through example simulation logs, operator-side telemetry, and video analyses illustrating scenarios such as UAV collisions, Remote ID spoofing, and fleet-level operations. We further evaluated the framework's scalability, finding that QGroundControl can support approximately 80 UAVs in real time on a single machine, Gazebo can support

around 5 UAVs with high-fidelity dynamics, and simulation without visualization can scale to roughly 240 UAVs.

Because the simulation exposes fine-grained control over inter-process communication and network behavior, it is particularly well suited to cybersecurity research. In particular, the framework enables controlled experimentation with the cyberattacks discussed in this paper, such as Remote ID spoofing, denial of service attacks, and man-in-the-middle attacks, while maintaining realistic dynamics and operator interactions. As a result, it provides a practical testbed for developing and evaluating threat detection strategies, response mechanisms, and overall resilience in networked UAS operations. Further exploring these cybersecurity dimensions is a primary direction for future work.

Acknowledgments

This material is based upon work supported by the NASA Aeronautics Research Mission Directorate (ARMD) University Leadership Initiative (ULI) under cooperative agreement number 80NSSC24M0070. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration.

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 2444112. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Betti Sorbelli, F., "UAV-Based Delivery Systems: A Systematic Review, Current Trends, and Research Challenges," *ACM J. Auton. Transport. Syst.*, Vol. 1, No. 3, 2024. https://doi.org/10.1145/3649224, URL https://doi.org/10.1145/3649224.

[2] Butilă, E. V., and Boboc, R. G., "Urban Traffic Monitoring and Analysis Using Unmanned Aerial Vehicles (UAVs): A Systematic Literature Review," *Remote Sensing*, Vol. 14, No. 3, 2022. https://doi.org/10.3390/rs14030620, URL https://www.mdpi.com/2072-4292/14/3/620.

[3] Erdelj, M., Natalizio, E., Chowdhury, K. R., and Akyildiz, I. F., "Help from the sky: Leveraging UAVs for disaster management," *IEEE Pervasive Computing*, Vol. 16, No. 1, 2017, pp. 24–32.

[4] Cohen, A. P., Shaheen, S. A., and Farrar, E. M., "Urban Air Mobility: History, Ecosystem, Market Potential, and Challenges," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 22, No. 9, 2021, pp. 6074–6087. https://doi.org/10.1109/TITS.2021.3082767.

[5] Patterson, M., "An Overview of NASA's AAM Mission and Select Partnerships," *AIAA Aviation Forum and Exposition*, American Institute of Aeronautics and Astronautics, San Diego, CA, USA, 2023. URL https://ntrs.nasa.gov/api/citations/20230008601/downloads/2023-06-15%20AIAA%20Aviation%20Panel%20v1.pdf, meeting held June 12–16, 2023.

[6] Johnson, W., and Silva, C., "NASA concept vehicles and the engineering of advanced air mobility aircraft," *The Aeronautical Journal*, Vol. 126, No. 1295, 2022, p. 59–91. https://doi.org/10.1017/aer.2021.92.

[7] Mendonca, N., Murphy, J., Patterson, M. D., Alexander, R., Juarex, G., and Harper, C., "Advanced Air Mobility Vertiport Considerations: A List and Overview," *AIAA AVIATION 2022 Forum*, American Institute of Aeronautics and Astronautics, 2022. https://doi.org/10.2514/6.2022-4073, URL https://doi.org/10.2514/6.2022-4073.

[8] Rohrmeier, K., Wei, W., and Ison, D., "Decoding the Vertiport: Planning for Urban Air Mobility," *Journal of Planning Literature*, 2025. https://doi.org/10.1177/08854122251314481, URL https://doi.org/10.1177/08854122251314481.

[9] Wang, B. H., Wang, D. B., Ali, Z. A., Ting Ting, B., and Wang, H., "An Overview of Various Kinds of Wind Effects on Unmanned Aerial Vehicle," *Measurement and Control*, Vol. 52, No. 7-8, 2019, pp. 731–739. https://doi.org/10.1177/0020294019847688, URL https://doi.org/10.1177/0020294019847688.

[10] Mohsan, S. A. H., Othman, N. Q. H., Li, Y., Alsharif, M. H., and Khan, M. A., "Unmanned Aerial Vehicles (UAVs): Practical Aspects, Applications, Open Challenges, Security Issues, and Future Trends," *Intelligent Service Robotics*, Vol. 16, No. 1, 2023, pp. 109–137. https://doi.org/10.1007/s11370-022-00452-4, URL https://doi.org/10.1007/s11370-022-00452-4.

[11] Chen, Z., Yan, J., Ma, B., Shi, K., Yu, Q., and Yuan, W., "A survey on open-source simulation platforms for multi-copter UAV swarms," *Robotics*, Vol. 12, No. 2, 2023, p. 53.

[12] Soria, E., Schiano, F., and Floreano, D., "SwarmLab: A MATLAB drone swarm simulator," *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2020, pp. 8005–8011.

[13] Frye, A. J., and Mehiel, E. A., "Modeling and simulation of vehicle performance in a UAV swarm using horizon simulation framework," *AIAA SciTech 2019 Forum*, 2019, p. 1980.

[14] Panerati, J., Zheng, H., Zhou, S., Xu, J., Prorok, A., and Schoellig, A. P., "Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 7512–7519.

[15] ArduPilot, "ArduPilot," https://ardupilot.org/, 2024. Accessed: 2024-05-13.

[16] Baidya, S., Shaikh, Z., and Levorato, M., "FlyNetSim: An open source synchronized UAV network simulator based on ns-3 and ardupilot," *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2018, pp. 37–45.

[17] Riley, G. F., and Henderson, T. R., "The ns-3 network simulator," *Modeling and tools for network simulation*, Springer, 2010, pp. 15–34.

[18] Fabra, F., Calafate, C. T., Cano, J. C., and Manzoni, P., "ArduSim: Accurate and real-time multicopter simulation," *Simulation Modelling Practice and Theory*, Vol. 87, 2018, pp. 170–190.

[19] Koenig, N., and Howard, A., "Design and use paradigms for gazebo, an open-source multi-robot simulator," *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*, Vol. 3, Ieee, 2004, pp. 2149–2154.

[20] Moon, S., Bird, J. J., Borenstein, S., and Frew, E. W., "A gazebo/ros-based communication-realistic simulator for networked suas," *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2020, pp. 1819–1827.

[21] D'Urso, F., Santoro, C., and Santoro, F. F., "An integrated framework for the realistic simulation of multi-UAV applications," *Computers & Electrical Engineering*, Vol. 74, 2019, pp. 196–209.

[22] Federal Aviation Administration, "14 CFR Part 89 – Remote Identification of Unmanned Aircraft," https://www.ecfr.gov/current/title-14/chapter-I/subchapter-F/part-89, 2021. Authority: 49 U.S.C. 106(f), 106(g), 40101(d), 40103(b), 44701, 44805, 44809(f); Section 2202 of Pub. L. 114-190, 130 Stat. 629. Source: Amdt. 89-1, 86 FR 4505, Jan. 15, 2021.

[23] Federal Aviation Administration, "Unmanned Aircraft System (UAS) Traffic Management (UTM) Concept of Operations v2.0," Tech. rep., U.S. Department of Transportation, Mar. 2020. URL https://www.faa.gov/sites/faa.gov/files/2022-08/UTM_ConOps_v2.pdf, accessed: 2025-05-20.

[24] MAVLink Development Team, "MAVLink Protocol," https://mavlink.io/en/, 2025. Accessed: 2025-05-13.

[25] Koubâa, A., Allouch, A., Alajlan, M., Javed, Y., Belghith, A., and Khalgui, M., "Micro air vehicle link (mavlink) in a nutshell: A survey," *IEEE Access*, Vol. 7, 2019, pp. 87658–87680.

[26] Canady, R., Bhowmick, C., and Koutsoukos, X., "Decentralized Learning using Hashgraph Consensus," *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2025.

[27] Team, A. D., "ardupilot_gazebo: Gazebo plugin for ArduPilot SITL," https://github.com/ArduPilot/ardupilot_gazebo, 2024. Accessed: 2025-05-13.

[28] Varga, A., and Hornig, R., "Overview of the OMNeT++ simulation environment," *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–10.