Developing Algorithms for the Internet of Flying Things Through Environments With Varying Degrees of Realism

Thiago Lamenza, Josef Kamysek, Bruno José Olivieri de Souza, Markus Endler

[1]Laboratory for Advanced Collaboration
Departamento de Informática – Pontifical Catholic University of Rio de Janeiro (PUC-Rio)

{tlamenza,bolivieri,endler}@inf.puc-rio.br, josef@kamysek.com

Abstract. *Developing for the internet of flying things is a complicated task. The fragility and cost of the equipment required to deploy in the field motivates the use of simulation software for prototyping and developing robust applications. This work proposes GrADyS-SIM NextGen as a solution that enables development on a single programming language and toolset over multiple environments with varying levels of realism. Finally, we illustrate the usefulness of this approach with a toy problem that makes use of the simulation framework.*

1. Introduction

When working with the development of distributed systems populated by autonomous nodes capable of movement, you end up dealing with networking and mobility, which highly affect the algorithm's performance and behavior [Olivieri De Souza et al. 2023]. Algorithms running in these environments require specialized features to achieve their desired level of robustness. The importance of these features doesn't become apparent until the algorithm is exposed to the conditions that motivate their development. The process necessary to create and validate these systems is very time-consuming and monetarily draining when reliant on field tests. For these reasons, using a simulated environment to aid the research and creation processes is essential. Simulation is a common approach to implementing solutions to problems in these scenarios, [Fabra et al. 2020], [Guillen-Perez and Cano 2018], [Sánchez-García et al. 2018].

Representing and simulating too many specific aspects of the real world at the same time hinders the development process. The increment in realism comes at a cost because with the increase in complexity of the simulation software, the development experience tends to deteriorate. A realistic simulation has a heavy computational cost that adds a bigger overhead to the execution of simulation scenarios, slowing down the development process. The complexity of the software also comes with other disadvantages as they tend to be harder to set up, have a steeper learning curve and be very specialized, making it hard to write code that translates well to other environments and the real world.

This work presents *GrADyS-SIM NextGen*, a framework for simulating distributed algorithms in a simulated network environment populated by nodes capable of communication and mobility. A short video showcase of the simulator is available, it also shows the installation process [1]. There are ample use cases for a simulator like this. Examples are simulating systems where unmanned aerial vehicles communicate with stationary sensors, the simulation of UAV formations that rely on communication to main-

[1]https://youtu.be/QMmup_nTfhI

tain a desired configuration, predator and prey scenarios and many more. The simulator presented in this work is not unique in the area of simulating networks of UAVs [Baidya et al. 2018] and work has been done in compiling comprehensive lists of existing options [Kang et al. 2016] [Phadke et al. 2023]. GrADyS-SIM NextGen distinguishes itself from the rest by enabling the iterative development process of distributed algorithms.

That main distinguishing factor manifests itself with the proposal of a general and environment agnostic way of implementing distributed algorithms that can run in different simulated environments and, in principle, in the real world. The main appeal of the framework is that no code changes need to be made to the algorithm itself when switching between these environments, making the knowledge obtained in a simulation environment easily transferable to another or to the real world. Having a common interface also enables the creation of tooling that improves the experience of creating distributed algorithms.

The framework enables you to create artifacts called *protocols* which using Python to implement the logic that powers individual nodes. These Python protocols can be used in a simple Python simulation environment, which we call *prototype-mode*. Users can also run them in *integrated-mode*, which will integrate with OMNeT++ [2], an event-based network simulator, to simulate realistic network interactions. In the future, users will be able to integrate it with SITL through the use of MAVSIMNET [3] which provides a realistic mobility models representing several vehicle types. Finally, these protocols can be used to control real-world vehicles in what we call *experiment-mode*, although this last execution mode has not been implemented yet.

This is a continuation of *GrADyS-SIM* [Lamenza et al. 2022] with which it shares its main objectives. The creation of this new simulation framework based on GrADyS-SIM was motivated by our experience in previous works, feedback from both within the team and from external users, and the realization that our current framework at the time made translating our simulated implementations to real-life for experiments very hard.

This paper is organized as follows. In section 2 we will discuss the concepts that guided the framework's implementation. In section 3 the framework's components will be presented on a lower level. In section 4 a demonstration of the usage of GrADyS-SIM NextGen is presented as empirical evidence that the proposed development approach has tangible benefits for algorithm development.

GrADyS-SIM NextGen is completely free and open-source, and the location of the source code and documentation will be detailed in the Architecture section. An extended version of this paper is available in [de Souza Lamenza et al. 2024].

2. Motivation

The GrADyS-SIM simulator was used in several projects internal to the GrADyS team and by some external users. A common pain point observed by both of these groups was the difficulty of setting up and using OMNeT++ and its component library INET [4]. OMNeT++ is absolutely essential to the project for its network simulation capabilities but using it comes with a couple of downsides. It is a very large piece of software, it is slow

[2] https://omnetpp.org/
[3] https://github.com/Thlamz/MAVSIMNET
[4] https://inet.omnetpp.org/

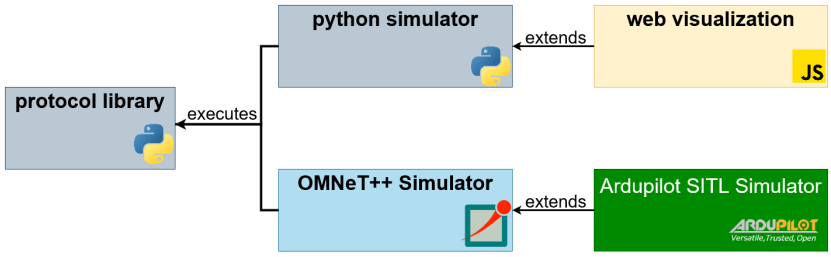

Figure 1. Framework's architecture

even on modern machines, build times and even simple tasks like checking a function's usages in code take a while, making development inconvenient. Its complex structure and ample set of features means that it has a steep learning curve and many possible points of failure. These failures are not always easy to debug. A significant portion of the simulator user's effort is non-productive.

OMNeT++ will no longer be a requirement to run this new version of the simulation framework. Instead, it would be an optional dependency if the user desired a realistic network simulation. In order to enable this, a new and simpler environment needed to be created. Also necessary, is a way to enable seamless integration between this new environment and the existing OMNeT++ one.

GrADyS-SIM uses C++ as its implementation language. Our project's real world test bed uses Python to implement node behavior. This means that anything developed inside the simulation would then need to be translated into Python, potentially introducing errors and requiring more development time. This led to the choice of Python as the language in which protocol logic would be written in.

The protocol interface was created to serve as an environment-agnostic interface to implement algorithms. Code that uses it should be able to run in any environment supported by the framework, be it simulated or real. It establishes a well-defined set of rules a protocol should follow in order to be compliant with the interface and enjoy the benefits of being decoupled from their environment. The main benefit is reducing or even removing the effort required to adapt them to new environments. This reduces the overhead required when moving tests between simulated environments and the real world.

Having a generalized API that protocols adhere also enables the creation of a toolset dedicated to aiding the creation of new protocols. Repetitive boilerplate work is required when creating new protocols and has to be re-implemented every time a new protocol needs to use it. Without a standard interface, protocol code is not easily reusable.

The last of the requirements to be fulfilled is introducing a new environment to serve as an entry point to the framework, replacing OMNeT++. This environment consists of a Python event-based simulator created specifically for the simulation framework. Protocols running in this mode are said to be running in *prototype-mode*. It is trivial to install, light on dependencies and works on Python version 3.8 or higher. Where it loses on is realism and richness of features, but the framework as a whole does not lose those qualities, as they can still run their protocols on OMNeT++ without any code changes.

3. Architecture

GrADyS-SIM NextGen is a simulation framework with several components. All components are publicly available and open-source. The framework is distributed in three GitHub repositories, *gradys-sim-nextgen* [5] contains the python components of the framework including the protocol library and the python simulator, *gradys-sim-visualization* [6] hosts the visualization website for python simulations and *gradys-simulations* [7] hosts the OMNeT++ components of the framework. The documentation for each of these components is also hosted in their own repositories.

There are three main ways of running simulations in the framework, which the user chooses based on his requirements:

1. The first one is running your simulations in the python simulator. Using this scheme, you will have access to all protocol features while running in a simplified environment with low implementation overhead and fast execution times.
2. When looking for more serious results using the same code base, one can try integrating with OMNeT++ and running the code built on the protocol library in the OMNeT++ simulated environment.
3. Lastly, building simulations completely in OMNeT++ is still supported, you will be creating all your code in C++ and won't be able to use it elsewhere. Using this option you can also integrate with Ardupilot SITL Simulator for a better mobility model, in the future this will also be available in option 2.

In the rest of this section, we will talk about each of the individual components seen in figure 1, detailing their purpose in the framework and how they are distributed.

3.1. Protocol library

To allow distributed algorithm logic to run in supported environments, be it simulated or real, a common and general interface needed to be established defining how information from the environment would be available to the protocol and how it interacts with it. Code showcasing the protocol library can be seen in listing 1.

Protocols are implemented in an event-oriented way. The events available to the protocol were chosen carefully as to give them the necessary information to implement their behavior, but not couple them to a specific environment. It is also essential that the protocol can act upon the environment. These actions are performed with the provider interface, named this way because it *provides* the protocol with the means to act in its environment.

Since we want to build an environment agnostic piece of code, we cannot deal with details like the physical means of locomotion of the node or the hardware that makes it capable of communication. For this reason, a generic set of messages were created to hide these details from the protocol through a layer of abstraction. Commands are sent by the protocol through the provider interface. The actual implementation of the provider interface is environment-dependent, it is injected into the protocols at run-time and the protocol does not rely on its design, only that it follows the specification defined in the interface.

[5] https://github.com/Project-GrADyS/gradys-sim-nextgen
[6] https://github.com/Project-GrADyS/gradys-sim-nextgen-visualization
[7] https://github.com/Project-GrADyS/gradys-simulations

Listing 1. A protocol that periodically broadcasts ping

```
# imports omitted
class PingProtocol(IProtocol):
    sent: int = 0
    received: int = 0
    def initialize(self):
        self.provider.schedule_timer("", self.provider.current_time() +
            random.random() + 2)
        # Using a plugin to move randomly inside the environment
        self.movement = RandomMobilityPlugin(self)
        self.movement.initiate_random_trip()
    def handle_timer(self, timer):
        # Broadcasting ping when timer fires
        command = BroadcastMessageCommand("ping")
        self.provider.send_communication_command(command)
        # Keeping track of messages sent
        self.sent += 1
        # Rescheduling the timer. We will keep pinging periodically
        self.provider.schedule_timer("", self.provider.current_time()+2)
    def handle_packet(self, message: str):
        # Keel track of pings received
        if message == "ping":
            self.received += 1

    # Unused methods from the protocol interface omitted
```

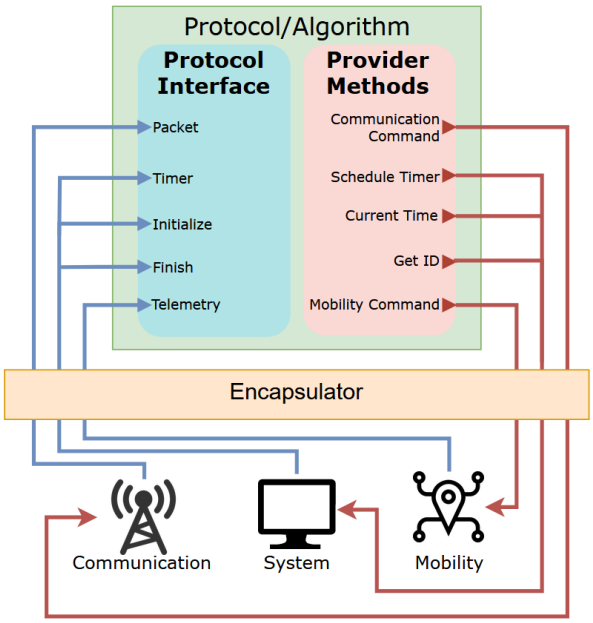

Figure 2. Diagram showcasing how protocols work

Another feature available in the protocol library is a set of tools named plugins. They come with varying objectives, but mainly focused on making implementing new protocols easier. They range from abstracting common behaviors in distributed algorithms from facilitating the implementation of movement patterns. These tools are only exposed to the protocol interface, and thus protocols that use them are still completely compatible with it.

In the figure 2, the protocol interface and provider methods are shown to be separated from the environment by the encapsulator, containing glue-code. The environment is represented in high abstraction by three components, mobility, communication, and system. *Mobility* is an abstraction for whatever empowers nodes with mobility, be that hardware or simulated code. Likewise, *Communication* represents what gives the node the power to communicate. Lastly, *System* represents a component that provides computation context and tasks, this could be the operating system in a real scenario or the simulator itself in a simulated scenario.

Everything described in this section is available in the *gradys-sim-nextgen* repository and can be installed directly from PyPI, Python's package repository under the name *gradysim*.

3.2. Python simulator

This component fits in to the framework as a first step for protocol development. The simpler simulator also serves as a great entry-point for new users who can learn how to create and implement distributed algorithms. The Python simulator is also part of the *gradysim* Python package, and its source code is available in the *gradys-sim-nexgen* repository.

Although simple in terms of functionality, it is still capable of emulating a lot of circumstances a real node will be exposed to. It simulates communication, allowing users to specify communication range and introducing common network failures such as network delay or dropped messages. It also simulates mobility, which is essential since this entire framework has been created to allow for the simulation of networks populated by mobile nodes.

3.3. Web visualization

This component integrates with the python simulator to provide a visual representation of a running simulation. Nodes are displayed as spheres positioned in a 3D visual environment, where the ground is marked as a black mesh. Information about the running simulation like the simulation time and the value of some variables marked for tracking inside protocols can be seen in the user interface. Users can also color specific nodes to distinguish them from the rest, this is very important to visually interpret some simulations. The component is available as a website [8].

3.4. OMNeT++

The next big component is the OMNeT++ simulator. This whole component was already available in the previous version of the simulation framework. It has been modified

[8] `https://project-gradys.github.io/gradys-sim-nextgen-visualization/`

slightly to account for interacting with the encapsulator module. The previous version of the simulator was already built on the idea of a central protocol module implementing the node's behavior, so adapting was easy. To bridge the communication gap between Python and OMNeT++, a proxy protocol sits in the C++ code redirecting all calls to Python. It acts as a middleman, ensuring that commands and information flow between the two environments.

Users can select the Python implemented protocol they want through OMNeT++'s configuration system. In execution time, this protocol will be imported and wrapped with an *encapsulator*. All interactions with python are managed through pybind11 [9].

4. Demonstration

A demonstration will be presented showcasing GrADyS-SIM NextGen as a tool for creating simulated environments and easily translating code between them. The demonstration presents a hypothetical scenario which is solved by a distributed algorithm implemented as a protocol in the framework. This protocol will then be tested and iterated as more realistic conditions are added. It is not meant as a revolutionary solution to a real world problem, but as a toy problem where the development of the algorithm itself is more important than the solution it proposes. The scenario will be better explained below.

The problem is a data-collection scenario set in some remote location deprived of any network infrastructure. A set of stationary sensors has been distributed in arbitrary known locations in the location of interest. These sensors collect data from their environment. They have limited communication range. The remote nature of the location and lack of infrastructure makes remote collection impossible, and the frequency with which data is generated makes manual collection impractical. Some quad-copter UAVs are available. These vehicles are capable of communication with the sensors and each other, and of autonomous flight. A ground station (GS) has been set up with short-range communication equipment capable of talking to the UAVs.

As for the solution, the UAVs will be employed in collecting data from the sensors and bringing it to the GS. They will fly above the sensors, communicating with them to retrieve the data collected and then return to the GS, delivering the data. The real challenge in this scenario is coordinating UAV movement efficiently to maximize their usefulness. This coordination will happen through a communication-based protocol.

5. Conclusion

The creation and demonstration of the GrADyS-SIM NextGen framework aims to address key challenges in the development and testing of distributed algorithms for autonomous vehicles in dynamic, networked environments. It streamlines the intricate processes of creating and evaluating distributed algorithms by offering an environment-agnostic interface for protocol implementation and support for multiple execution environments.

The framework's architecture, as illustrated in Figure 1, is designed with a focus on modularity and flexibility. The protocol module establishes a standardized interface for protocol implementation, ensuring that the same code can run seamlessly across different simulated environments.

[9] `https://github.com/pybind/pybind11`

The introduction of *prototype-mode* provides users with a lightweight and accessible simulation environment. With a simplified setup and a focus on rapid prototyping. The *integrated-mode* leverages the power of OMNeT++, a widely used event-based network simulator, to provide a more detailed and accurate representation of network behaviors. The seamless transition between *prototype-mode* and *integrated-mode* ensures that the developed protocols in the initial stages of development can very easily be used in a realistic network simulation.

The experiments presented in section 4 will show the tangible benefits of the proposed approach.

The next natural step in the development of the framework is supporting real-world scenarios. The foundational for this to be possible has already been laid. All that is left is selecting a hardware test bed and implementing the first integration between it and the framework. This would close the development cycle from inception, to prototyping, to validation and finally to experimentation and deployment.

Acknowledgments

This study was financed in part by AFOSR grant FA9550-23-1-0136.

References

Baidya, S., Shaikh, Z., and Levorato, M. (2018). Flynetsim: An open source synchronized uav network simulator based on ns-3 and ardupilot. In *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 37–45. ACM.

de Souza Lamenza, T., Kamysek, J., de Souza, B. J. O., and Endler, M. (2024). Developing algorithms for the internet of flying things through environments with varying degrees of realism.

Fabra, F., Zamora, W., Reyes, P., Sanguesa, J. A., Calafate, C. T., Cano, J.-C., and Manzoni, P. (2020). Muscop: Mission-based uav swarm coordination protocol. *IEEE Access*, 8:72498–72511.

Guillen-Perez, A. and Cano, M.-D. (2018). Flying ad hoc networks: A new domain for network communications. *Sensors*, 18(10).

Kang, S., Aldwairi, M., and Kim, K.-I. (2016). A survey on network simulators in three-dimensional wireless ad hoc and sensor networks. *International Journal of Distributed Sensor Networks*, 12(9):1550147716664740.

Lamenza, T., Paulon, M., Perricone, B., Olivieri, B., and Endler, M. (2022). Gradys-sim – a omnet++/inet simulation framework for internet of flying things.

Olivieri De Souza, B. J. O., Lamenza, T., Paulon, M., Rodrigues, V. B., Carneiro, V. G. A., and Endler, M. (2023). Collecting sensor data from wsns on the ground by uavs: Assessing mismatches from real-world experiments and their corresponding simulations. In *2023 IEEE Symposium on Computers and Communications (ISCC)*, pages 284–290.

Phadke, A., Medrano, F. A., Sekharan, C. N., and Chu, T. (2023). Designing uav swarm experiments: A simulator selection and experiment design process. *Sensors*, 23(17).

Sánchez-García, J., García-Campos, J., Arzamendia Lopez, M., Gutiérrez, D., Toral, S., and Gregor, D. (2018). A survey on unmanned aerial and aquatic vehicle multi-hop networks: Wireless communications, evaluation tools and applications. *Computer Communications*, 119.