GrADyS-SIM - A OMNET++/INET simulation framework for Internet of Flying things

Thiago Lamenza[1], **Marcelo Paulon**[1], **Breno Perricone**[1], **Bruno Olivieri**[1], **Markus Endler**[1]

[1]Laboratory for Advanced Collaboration, Departmento de Informática
Pontificial Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Brazil

Abstract. *This paper describes GrADyS-SIM, a framework for simulating cooperating swarms of UAVs in a joint mission in a hypothetical landscape and communicating through RF radios. The framework was created to aid and verify the communication, coordination, and context-awareness protocols being developed in the GrADyS project. GrADyS-SIM uses the OMNeT++ simulation library and its INET model suite and allows for the addition of modified/customized versions of some simulated components, network configurations, and vehicle coordination to develop new coordination protocols and tested through the framework. The framework simulates UAV movement dictated by a file containing some MAVLink instructions and affected on the fly by different network situations. The UAV swarm's coordination protocol emerges from individual interactions between UAVs and aims to optimize the collection of sensor data over an area. It also allows simulation of some types of failures to test the protocol's adaptability. Every node in the simulation is highly configurable, making testing different network topographies, coordination protocols, node hardware configurations, and more a quick task.*

1. Introduction

When developing new network solutions, simulation is a powerful tool for testing, analyzing performance, and evaluating scalability. This fact becomes more apparent when these networks have mobile air vehicles as some nodes. Simulation allows for multiple hypothesis checks through the easy setup of different network configurations and topologies and faster play of specific use case scenarios, enabling much faster debugging and adjustments than deployment and testing in the field. However, any simulation operates according to models (wireless transmission, mobility model, terrain model, etc.) and is not completely precise or accurate. Therefore, we believe in the combined and interleaved use of simulation and field tests and are particularly interested in identifying in which aspects simulations and field tests - of the same scenario - differ and to what degree.

This work, more specifically the development of GrADyS-SIM, is one of the tangible results of the research project entitled GrADyS [Olivieri De Souza et al. 2020] (Ground-and-Air Dynamic sensors networkS). An important application of network technology is the creation of vehicle ad hoc networks, populated by mobile nodes that communicate with static nodes and other mobile nodes to implement some behavior. The GrADyS[1] project was created to investigate the applications of these UAV swarm networks in the monitoring of remote, dangerous, or hard-to-reach regions through the collection of sensor data using UAV swarms. UAV or Aerial swarms are a specific type

[1]http://www.lac.inf.puc-rio.br/index.php/gradys/

of multi-agent robotic system with three-dimension freedom of movement and communication through wireless communication. In UAV swarms, one of the biggest challenges is to ensure correct action/movement coordination of the UAVs since this is usually done on the fly (literally) and must rely on intermittent and unreliable wireless connections[Abdelkader et al. 2021].

This paper describes the simulation framework created to aid and validate the project's development. Through the OMNeT++ simulation library and the INET model suite and with the creation and modification of simulated components, network configurations and vehicle coordination protocols are created and tested in this simulation framework.

The framework simulates UAV movement dictated by a simplified file containing some MAVLink instructions and is affected on the fly by different network situations. The UAV swarm's coordination protocol emerges from individual interactions between UAVs and has the objective of optimizing the collection of sensor data over an area. It also allows simulation of some types of failures to test the protocol's adaptability. Every node in the simulation is highly configurable, making testing different network topographies, coordination protocols, node hardware configurations, and more a quick task. OMNeT++ uses a configuration file system that lets the developer easily create and keep track of these situations. A typical use case would be benchmarking the data collection performance of different coordination protocols with a different amount of UAVs on a waypoint mission.

The project and documentation are available on Github[2], and a video showcase of the project can be found on Youtube[3].

2. OMNET++/INET

OMNeT++ is a discrete event simulator implemented as a component-based C++ library. It primarily allows developers to build complex network simulations by creating and extending components that communicate with each other to implement behavior. It is highly extensible and modular, and the INET model suite provides a large component library to aid developers in implementing network simulations. These highly parameterized and extensible components provide solutions for simulating environments, a complete network stack, mobility, and many other things.

This environment provides the necessary tools for developing flexible simulations capable of representing different network situations with enough fidelity to serve as a tool for testing and validating coordination and network protocols and the network topographies needed to implement real-life mobile networks. Data collection and observation test many variations in the network's implementation rapidly. The simulation can be easily configured to represent these variations by customizing component parameters and by the extension or creation of new components when needed.

3. Architecture

The framework's architecture is mainly composed of three models: one responsible for communication between UAVs (communication), one for controlling the node's move-

[2]https://github.com/brunoolivieri/gradys-simulations
[3]https://youtu.be/Im6d5TEes4Y

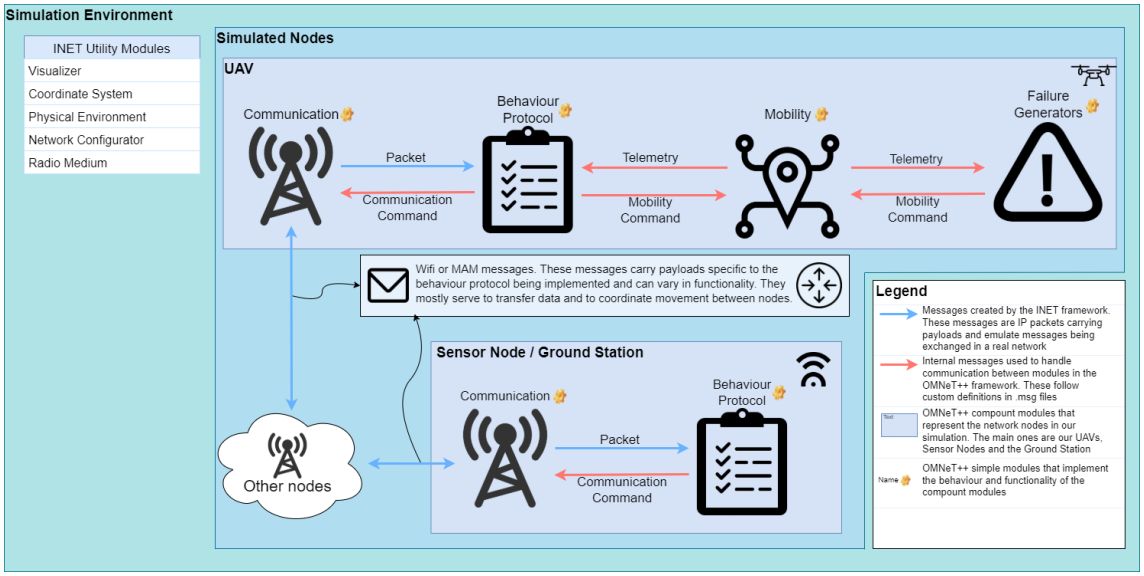

Figure 1. Project's Architecture Diagram

ment (mobility), and the last to manage the interaction between the last two (protocol). The behavior and implementation of these modules is detailed further below. They were made in such a way that the messages exchanged between them are sufficiently generic to allow the creation of a new protocol by creating a new protocol module, with no changes to the other ones by leveraging these generic messages to carry out different procedures. The messages exchanged between them are described on `.msg` files like `MobilityCommand.msg`, `Telemetry.msg` and `CommunicationCommand.msg` and define the format of the data being exchanged between them.

These three modules are loaded in a compound module defined by a `.ned` file. In OMNeT++ `.ned` files define modules that can use other modules forming a module tree. These modules can be simple (the leaves of the module tree) or a compound module that connects simple modules or other compound modules with gates. A network is a special kind of compound module that can be run as a simulation.

The compound module that represents UAVs in the simulation is the `MobileNode.ned` and `MobileSensorNode.ned` represents our sensors. These modules contain Communication and Mobility modules (defined in AdhocHost, this module's parent) and the Protocol module (defined in the file). The `mobilityDrones.ned` file connects all the UAVs (called quads), sensors and some other modules necessary to the simulation.

3.1. Components

3.1.1. Mobility

The mobility module is responsible for controlling UAV movement and responding to requests from the protocol module to change that movement through MobilityCommand messages. It also needs to inform the protocol module about the current state of the UAV's movement through Telemetry messages. As part of the module initialization, the

waypoint list is attached to a Telemetry message, so the protocol module has access to the tour the mobile node is following.

Currently, the only mobility module being used is the `DroneMobility.ned` module. The module responds to several commands defined by the MobilityCommands message type that is used by the protocol module to control it. Its default behavior is to follow a list of waypoints defined in a waypoint file, but through commands, it can perform several tasks like reversing course, going to a specific point, or sitting idle. This module was adapted from the VehicleMobility module provided by INET to simulate ground vehicle movement through a series of waypoints. It was adapted to work in three dimensions, allowing flight, and following a waypoint file akin to the MAVLink waypoint file format.

An optional feature of the mobility module is attaching a failure generator module. They connect to the mobility module using the same gates the protocol module does and use that to send commands in order to simulate failures. This can be used to trigger random shutdowns and even to simulate energy consumption. An example of a module that simulates energy consumption is the SimpleEnergyConsumption, a parametrized component to simulate consumption and battery capacity. It sends RETURN_TO_HOME messages to the vehicle when the UAV's battery reaches a certain threshold and shuts it down when the battery is depleted.

3.1.2. Communication

INET provides built in support for the simulation of real communications protocols and the communication module takes advantage of this to simulate communication between nodes. It also has to inform the protocol module of the messages being received by sharing the messages themselves and listen to orders from the protocol module through CommunicationCommands.

Several implementations of the communication module are used. These implementations contain functions that interface with INET's communication capabilities but don't implement interaction with any other module. The interactions themselves are defined and controlled by the protocol module.

3.1.3. Protocol

The protocol module manages the interaction between the movement and communication of the mobile nodes. It makes use of the messages provided by its two sibling modules to create node interaction strategies. It primarily reacts to messages it receives from those modules and determines which orders to give them to achieve the desired result.

It gathers information about the current state of the simulation by analyzing Telemetry messages received from the Mobility module and Packets forwarded to it by the Communication module. An important task it performs is the definition of the message sent by the Communication module. These messages will be sent to other nodes that will themselves handle them. The messages are inserted into IP Packets as payloads. They can have different formats depending on the protocol being implemented.

An example of a UAV coordination protocol that was implemented in this framework was DADCA [Olivieri de Souza and Endler 2020]. The DADCA investigates whether it is possible to implement a distributed algorithm to coordinate several fully autonomous (i.e., non-human-controlled) UAVs collecting data from a WSN without centralized control or knowledge of internal UAV states and relying on only ad-hoc communication.

Another example of a protocol implemented was MAM[Paulon J. V.. et al. 2021]. MAM proposes two alternatives to BTMesh's default relay algorithm (MAM_0 and MAM_Δ) that may achieve higher packet delivery rates and lower energy draw when routing data towards a Mobile-Hub.

Protocols implement the IProtocol interface and extend `CommunicationProtocolBase.ned`, which provides useful stub functions to use when implementing protocols.

Some of the currently implemented protocols are as follows:

- `ZigZagProtocol.ned` and `ZigZagProtocolSensor.ned`
  These files implement the mobile node and the sensor side of the ZigZag protocol. This protocol manages a group of UAVs following a set path passing above several sensors from where they pick up imaginary data from those sensors. The UAVs also interact with each other sending several messages to coordinate their movement.
  Heartbeat messages are sent to a multicast address. If these are picked up by sensors, they respond with data. If other UAVs pick them up, they initiate a communication pair by sending a Pair Request message which is confirmed by the other UAV with a Pair Confirmation message. The UAV furthest away from the starting point of the path sends its data to the other UAV in the pair, and they both reverse their movement. The objective is that, over time, the UAVs will each occupy an equally sized section of the course, picking up data on the way and sharing it at their section's extremities.
- `DadcaProtocol.ned` and `DadcaProtocolSensor.ned`
  This protocol is similar to the ZigZagProtocol. It also manages data collection by mobile nodes in a set path. The difference is that this method aims to speed up the process of equally spacing the UAVs in the course by implementing a more advanced movement protocol.
  When the Pair Confirmation message is received by both UAVs, confirming the pair, both UAVs take note of the number of neighbors on their left (closer to the start) and their right (further from the start) and share this information with their pair. Both update their neighbor count and use it to calculate a point in the course that would represent the extremity of both their sections if their current count of neighbors is accurate. Then they both travel together to this point and revert. This behavior is implemented with a sequence of commands that get queued on the mobility module.

3.2. Operation

The nature of the operation of a OMNeT++ simulation is dictated by the messages exchanged between modules and how they react to them. In the case of the GrADyS project,

the focus of our simulations is the use of quadcopters to collect data from static sensors spread in a field. The UAVs and sensors are the network nodes of the simulation and are the main focus of development efforts. Each of these nodes is composed of three main modules, illustrated in the previous sections, that interact with each other to form the node's behavior.

Since all our nodes are composed of the same types of modules, development is fast and simple, and the implemented coordination protocols focus on creating complex behavior emerging from individual actions taken by the nodes. In the protocols that we have developed, every node of the same type is functionally identical, and there is no coordinator, but they still need to be coordinated to ensure efficient data collection. In the Dadca protocol, this is achieved by the collection of information by each of the UAVs to gather a basic understanding of the layout and the distribution of other UAVs in the formation, namely by the counting and sharing of other neighbor UAVs they have encountered. The Zigzag protocol, more primitively, has the UAVs reverse at every encounter with each other, with the vehicle farthest away from the ground station passing on the data to the one closest, ensuring it will eventually reach it.

The way that these nodes act is mainly determined by the implementation of their protocol module. It uses the other two modules as both sensors and actuators, gathering information about the network's state through information about the node's movement and messages received from other nodes and using this information to command the other two modules to perform the desired behavior. An example of this is how in the previously described Zigzag coordination protocol, the protocol module, on receiving communication from another UAV, compares the location visible in their message with its own location by analyzing telemetry received from the mobility module and uses all this information to decide if it should collect or send data to the other vehicle and commands the mobility module to invert the UAV's movement.

4. Usage

OMNeT++ simulations are initialized by .ini files. These files are used to set the many parameters of the simulation. These parameters control everything from the flight speed of the UAVs to the implementation of the protocol module to be used for the sensor nodes. They are neatly organized in these files and can be grouped in launch configurations. The launch configurations set a group of parameters for the simulation and can be used to easily switch between different sets of simulation types and node behaviors.

The mobilityDrones-omnetpp.ini file contains some launch configurations for Wifi only communication and shared Wifi and MAM communication, each with configurations for one to four UAVs. Launch configurations are defined in the same .ini file denoted by the [Config Sim2drone] tag, where Sim2drone is the name of the launch configuration. The [Config Wifi] and [Config MAM] configs are base configs for the other ones and should not be run directly.

These configurations have allowed us to benchmark different coordination protocols on their efficiency in transporting data from sensors to a ground station. Translating the technical behavior of the protocol into actual simulated behavior is simplified by the framework's structure. Since the mobility and communication modules are blind to the protocol's behavior, the only module that needs to be changed is the Protocol module.

Coordination requires information, and since this framework specializes in dealing with decentralized coordination protocols, information resides in the mobile nodes that populate the network. The first step in implementing a protocol is defining what information the UAV should use to make its decisions. That information needs to be packed into a message definition so that it can be shared between nodes during the simulation. After the information, the next step is to define how that information will affect a UAV's movement and behavior.

Our simplest protocol, the ZigZag protocol, only uses the vehicle's current position in the mission and which way it is traveling as information. Based on that, it is defined that when two UAVs meet, they will only interact if they are traveling in opposite directions on the mission path. If they are, the UAV traveling towards the mission start, where the ground station resides, will set its payload to zero, and the one traveling away from it will add to its payload whatever the other one was carrying. After that, they will both reverse their courses and initiate a timeout, during which they will not interact with any other UAVs. This interaction happens in three steps. The message definition for this protocol, besides carrying the mentioned information, carries a state flag that allows the UAVs to know which step they are one. The first step is a continuous heartbeat sent by the UAVs that, when heard, will be answered with a pair request message which in turn will be answered with a pair confirmation message, confirming that both UAVs are aware and have completed the exchange.

This is only one example of a coordination protocol. As mentioned, this framework has allowed us to benchmark these protocols. By setting them up under the same conditions and observing how much data is transferred to the ground station in a set amount of time. It is easy to see that a more sophisticated protocol like Dadca strongly improves the time the UAVs take to reach a stable state and how that improves the amount of data being passed to the ground station. OMNeT++'s parametric nature also helps set up these conditions quickly, being able to easily tune things like the number of UAVs present in the simulation.

Another use of the framework is to set up and test different communication protocols and simulated hardware. OMNeT++'s statistical features help to create easy-to-understand graphs that show how different network stacks affect things like communication range which in turn affect the number of messages being exchanged and the UAV's sensor data collection rates.

4.1. Installation

In order to run the simulations and use the components in this repository, you need to have both OMNeT++ and the INET framework installed.

Version 5.6 of OMNeT++ is required. To install it, just follow these instructions. INET version 4.2 is also required. When first opening the OMNeT++ IDE, you should be prompted with the option to install INET, and all you need to do is accept it, but if you need help, check out the installation instructions.

After installing both OMNeT++ and INET, you should be able to clone the repository to your active OMNeT++ IDE workspace. To do this, select "File → Import..." then open the "git" section and select "Projects from git" then "Clone Uri". After that, just fill in the URL for this repository and finish the process following the displayed instructions.

5. Conclusion

This work is a step within a set of deliverables for a project. The GrADyS project uses this tool to verify protocols and compare field test results with accurate sensors and UAVs.

This tool is in evolution, is licensed as open-source, and can be accessed freely[4]. From this, we hope that other research groups can reuse it with or without our involvement and contribute to that.

Acknowledgments

This study was financed in part by AFOSR grant FA9550-20-1-0285.

References

Abdelkader, M., Güler, S., and Jaleel, H. e. a. (2021). Aerial swarms: Recent applications and challenges. *Curr Robot Rep*, (2):309–320.

Olivieri de Souza, B. J. and Endler, M. (2020). Evaluating flight coordination approaches of UAV squads for WSN data collection enhancing the internet range on WSN data collection. *Journal of Internet Services and Applications*, 11(1):4.

Olivieri De Souza, B. J., Paulon, M., Vasconcelos, J., and Endler, M. (2020). GrADyS: Exploring movement awareness for efficient routing in Ground-and-Air Dynamic Sensor Networks. Available at `http://arxiv.org/abs/2012.10690`.

Paulon J. V.., M., Olivieri de Souza., B., and Endler., M. (2021). Opportunistic Routing towards Mobile Sink Nodes in Bluetooth Mesh Networks. In *Proceedings of the 18th International Conference on Wireless Networks and Mobile Systems - WINSYS,*, pages 67–75. INSTICC, SciTePress.

[4]https://github.com/brunoolivieri/gradys-simulations