Autonomous Network Self-Healing in Drone Swarms

1st Federico Corò
University of Padua: Computer Science
University of Padua
Padua, Italy
federico.coro@unipd.it

2nd Gianluca Bresolin
University of Padua: Computer Science
University of Padua
Padua, Italy
gianbreso02@gmail.com

3rd Riccardo Fabbian
University of Padua: Computer Science
University of Padua
Padua, Italy
fabb.riccardo@gmail.com

***Abstract*—This research reports a self-healing protocol for a swarm of drones, where drones are able to detect loss of connection with the base station and reorganize themselves to restore connectivity, through local communication and decentralized control. Specifically, after the loss of communication between a drone and the base station, drones autonomously reorganize through a formation control method based on a virtual spring damper model to form a multi-hop relay chain, where agents are attracted by preceding and following drones, while being repelled by others to avoid collisions.**
The experiment was carried out in *ns-3*, a network simulator, where we modeled a swarm of drones communicating through radio signals and moving in a 3D space.
Results demonstrate the feasibility of the proposed method. After a drone loses communication, the swarm successfully reorganizes to maintain network connectivity with the base station. This is achieved by forming a stable multi-hop relay chain, where intermediary drones position themselves at the geometric midpoint between their preceding and following neighbors.

***Index Terms*—Drones, Swarm, Wireless, Flooding, Self-Healing.**

I. Introduction

In recent years, the use of drone swarms to collaboratively perform tasks has gained significant attention and widespread adoption in various fields. Compared to single drones, swarms provide increased coverage area, greater redundancy, and improved operation efficiency. Moreover, their modular nature allows different units to carry specific equipment, making the system highly adaptable for complex multi-task operations.
However, drone swarms also face challenges, particularly in maintaining robust communication among drones and with the base station and in coordinating the formation control laws required to perform tasks effectively.
Specifically, in decentralized swarm operations, communication between a specific agent and the base station is frequently interrupted by signal jamming or geographic occlusions. In such scenarios, it is crucial for the swarm to autonomously detect the communication loss and reorganize itself through a "self-healing" protocol, able to form a multi-hop relay chain in order to avoid network partitioning and ensure continuous connectivity with the base station, all without centralized control.

In this study, we implemented a self-healing protocol in which drones inside the swarm are able to detect loss of communication with the base station. By exchanging state information with local neighbors, the agents leverage a formation control method based on a virtual spring-damper to reorganize the swarm and form a multi-hop relay chain.
Our main contribution lies in demonstrating the effectiveness of this protocol through an *ns-3* simulation, validating its ability to maintain network integrity in adverse scenarios.

The rest of this paper is organized as follows. Section II presents the self-healing protocol and its related formation control method. Section III details the simulation scenario and the *ns-3* implementation. Section IV reports the results obtained from the simulation and their analysis. Finally, Section V concludes the paper and outlines potential future work.

II. Self-Healing Protocol

Our self-healing protocol is designed to enable a drone inside a swarm to detect loss of communication with the base station and autonomously reorganize to form a multi-hop relay chain. The protocol comprises two main components: the communication loss detection mechanism and the formation control method based on a virtual spring-damper model.
Upon detecting a loss of communication with the base station by a timeout mechanism (i.e., it does not receive any acknowledgment from the base station within a predefined time interval), a drone initiates the self-healing process by running the formation control algorithm based on the virtual spring-damper model with the received information from its local neighbors.
This information exchange involves the current position and the respective number of hops to the base station, defined through a flood mechanism.
Based on these data and the current drone's state, the formation control algorithm calculates attractive forces towards its preceding and following drones, identified via hop count, while applying repulsive forces against other drones to prevent collisions (i.e., when the distance falls below a predefined safety threshold).
The resulting force determines the trajectory and the velocity command. After applying this command, the drone broadcasts

its updated position and hop count to its local neighbors.
This process continues iteratively, with the drones guided to a new position that corresponds to the geometric midpoint between their preceding and following drones. This position strategy is specifically chosen to maximize the Signal-to-Noise Ratio (SNR) for both links within the multi-hop relay chain.

A. System Architecture

The system architecture is built around two node roles: a stationary Base Station (B) and a set of mobile Drones (D_i) running the same overall control stack. To enable a future hardware verification phase, the architecture mirrors the Crazyflie 2.1 ecosystem as closely as possible. In particular, it adopts the characteristic "two-sided" communication pattern offered by the Crazyflie 2.4 GHz radio: the base station communicates with individual drones using unicast, while drones coordinate with each other using broadcast. This design choice is reflected throughout the architecture and strongly influences how information propagates in the swarm. Specifically, this means that control traffic to the base station is sent point-to-point, while swarm coordination mechanisms (flooding and neighbor information exchange) are propagated through broadcast.

At runtime, every message is first handled by the `CommunicationManager`, which packs and unpacks `Packet` envelopes and delegates actual delivery to the `Transport` implementation. The envelope contains a source ID, a destination ID (or a broadcast ID), a high-level type (CORE, FLOOD, or NEIGHBOR) and a payload whose interpretation depends on the selected type. Keeping a single wrapper format makes the stack easy to extend and, more importantly, lets higher-level modules ignore the details of how bytes move through the network.
The received packets are then passed to a `DispatchManager`, whose job is simply to forward each packet to the module responsible for that protocol. This results in three main pathways:

- `FloodManager` handles hop-discovery floods;
- `NeighborManager` maintains a local table of neighbor state (*id*, *hops-to-base*, *coordinates*);
- *Node Logic* handles core control messages, such as position reporting to the base station and the exchange of help-proxy requests and ACKs.

On the drone side, the control loop closes through motion: a `Controller` reads the current position and the latest neighbor information offered by the `NeighborManager`, computes a movement command produced by a virtual spring-damper model, and applies it through a `VelocityActuator` implementation.
The base station, instead, remains stationary and unicast-only: it tracks drone position updates, replies with acknowledgments, and periodically triggers a new flood.

B. Help-Proxy Triggering

Our self-healing protocol requires that each drone is able to autonomously detect loss of communication with the base station and trigger the self-healing mission.
To track reachability, each drone sends periodic `PositionUpdate` messages to the base station using unicast and waits for an acknowledgment (ACK). If no ACK is received within a safety timeout window, the drone considers the link broken and triggers the mission activation: this is meant to be a simple, swarm-visible indication that direct connectivity to the base has been lost.
Mission activation is triggered by an explicit "help-proxy" signal that is generated and broadcasted by the disconnected drone through an `HELP_PROXY` message (as a CORE packet). Any other drone that receives a `HELP_PROXY` message targeting the same base station switches into mission mode and starts the repositioning. The message is sent via broadcast to match the Crazyflie communication model.

C. Flooding-Based Hop Discovery

The flooding component provides a simple way for drones to estimate how far they are (in hops) from the base station, even when direct connectivity is lost. In line with the Crazyflie communication model, the flood is initiated from the base station using unicast and then propagated towards other drones in the swarm using broadcast.

The base station periodically sends a `START` message to one drone chosen as an initiator. After receiving this message, the initiator starts the process by broadcasting a `DISCOVERY` packet with hop count set to 1. When a drone receives a discovery packet, it calculates its hop-to-base value based on the following logic: if it currently has a direct link to the base (detected through the ACK-based reachability machanism during periodic position reporting previously described), it can immediately declare hop 1, otherwise it increments the hop carried by the discovery message and uses that value as its distance. This coupling between the flooding layer and the reachability mechanism ensures that hop estimates remain consistent with the current network conditions and prevents stale "direct" hops from being used after a disconnection: indeed, before any more recent flood is observed, a node detecting a link failure prevents itself from reporting the stale "hop 1", thereby avoiding false attraction toward a base station link that no longer exists.

To avoid blind rebroadcasting, each node only reacts when the new value is better than the best hop previously stored for the same flood instance. When improved, the node broadcasts a `REPORT` packet with its current best hop-to-base, and it also rebroadcasts `DISCOVERY` with the new hop count so the information can propagate further. Reports are forwarded at most once per improvement per reporter, which keeps the protocol lightweight. The resulting hop estimates are then used by the formation controller to infer relay ordering and drive the physical reconfiguration.

D. Neighbor Information Exchange

To react locally and without centralized coordination, each drone in the swarm maintains a lightweight "situational awareness" view of nearby agents. This is implement using periodic neighbor broadcasts, where every drone announces its own state to the swarm and listens for the state broadcast by others.

Neighbor information is distributed as `NEIGHBOR` packets sent in broadcast. The payload is kept simple to reduce overhead: it contains the sender ID, its current estimated hop distance to the base station, and its current coordinates encoded as 64-bit floating point values.
On reception, the `NeighborManager` performs basic sanity checks (e.g., the packet header source must match the ID stored in the payload) and updates a local neighbor table keyed by neighbor ID. This table is then exposed to the controller as a list of `NeighborInfo` entries.

When a drone is in proximity of the base station (i.e. it can directly communicate with it), it will integrate into the same neighbor abstraction the base itself. Specifically, whenever a drone receives a position acknowledgment from the base, the ACK embeds the base position and a hop value of 0. The drone then converts this information into the same neighbor payload format and injects it into the `NeighborManager`. As a result, the controller can treat the base station just as another neighbor in its computations, without requiring special-case logic.

E. Formation Control via Virtual Spring-Damper Model

Once hop distance and neighbor positions are available, the physical "self-healing" behavior is implemented through a controller based on a virtual spring-damper model. The controller runs periodically and produces a motion command by combining two effects: attraction toward relevant neighbors (to close the gap and form a relay) and repulsion at short range (to preserve safe separation).

At each step, a drone first retrieves its current coordinates via `PositionManager` and the latest neighbor table from the `NeighborManager`. It then iterates over all known neighbors to compute the relative displacement vector and accumulates an attractive contribution proportional to that displacement (scaled by K_{att}). This linear force is obtained only from neighbors whose hop count is either greater or less than the current drone's hop count (i.e., the preceding and following drones in the relay chain), thus driving the drone toward the geometric midpoint between them.
To avoid collisions, when the neighbor is closer than a safety threshold D_{safe}, the controller adds a repulsive term whose magnitude grows as $1/d^2$ (scaled by K_{rep}), pushing the drone away from the neighbor along the unit displacement direction. We underline how this non-linear repulsive force is obtained from all neighbors, including also peers (i.e., drones with the same hop count value).

The resulting net force is mapped to an acceleration vector according to Newton's second law, factoring in the drone's mass parameter. This acceleration vector is then applied through the `VelocityActuator` with a maximum speed constraint V_{max}.

The controller can be toggled via a mission flag: when the mission is inactive, drones keep broadcasting their neighbor information while following an idle motion behavior; once the mission is activated (e.g., after an help-proxy trigger), the virtual spring-damper model loop drives the repositioning dynamics. Indeed, each drone alternates between two macro-states: an idle monitoring state and a mission (self-healing) state. The intent behind this split is to keep the system lightweight under normal conditions while still being able to "switch on" coordinated motion when a disconnection is detected.

III. Simulation Tested

The proposed self-healing protocol was evaluated through a custom simulation scenario implemented in *ns-3*, a discrete-event network simulator widely used for research and educational purposes.

A. Ns-3 Environment

The proposed protocol was evaluated in a custom *ns-3* scenario designed to reproduce the key connectivity conditions that trigger self-healing. All nodes within the simulation share a common radio and IP environment implemented through a `RadioEnvironment` component. This environment installs an ad-hoc Wi-Fi network (IEEE 802.11b, constant 1 Mbps data and control rate) and assigns IPv4 addresses within a single subnet.

Coverage uses a strict distance cutoff through `RangePropagationLossModel`. Since swarm broadcast is implemented as a unicast fan-out at the transport layer, a receive-time distance check is used to ensure that packets beyond the maximum range are dropped. All communications (unicast and broadcast) are carried over UDP sockets bound to a single configurable port.

B. Scenario Configuration

The simulation generates four *ns-3* nodes: one base station (B) and three drones (D_1, D_2, D_3). The initial positions are chosen to create a partition between one drone and the base station, while keeping it connected to the rest of the swarm. The base station is placed at the origin $(0, 0, 0)$ while the drones are located at $(20, 10, 0)$, $(60, 15, 0)$ and $(40, 20, 0)$.
The base station is responible for triggering flooding to enable hop discovery: in this simulation, our base station schedule a `START` flood message with drone D_1 as initiator every 0.5 s.
Regarding drones, each one sets its periodic tick at 0.5 s: each

tick implies three main actions, performed in the following order:

- sends a `PositionUpdate` message to the base station via unicast;
- checks if the ACK timeout has expired and, if so, broadcasts a `HELP_PROXY` request exactly once;
- runs one controller step (idle or mission mode).

To avoid drift accumulation, each next tick event is scheduled exactly 0.5 s after the previous tick event, regardless of the time taken to perform the actions above, even if the protocol does not require strict synchronization.

In this configuration, D_2 starts outside the coverage area of the base station (50 m): this will lead the drone to send periodic `PositionUpdate` messages, without receiving `PositionAck` responses from the base station, while still remaining within the range of other drones, so that its broadcast help request can be received.
After the expiration of the ACK timeout, set in this simulation at 1.5 s, the victim drone will send a single `HELP_PROXY` broadcast. It is important to note that in the current implementation, the requester does not switch to mission mode active: this design choice prevents the relay chain to collapse due to the requester suffering only attractive forces from preceding neighbors, since it represents the estremity of the chain. For this reason, the self-healing protocol is driven by drones that hear the request.
Once the help request is broadcasted, drones D_1 and D_3 switch to mission mode and begin applying the virtual spring-damper controller at each tick.

Accordingly with the *Crazyflie 2.1* specifications, each drone is characterized by the following parameters:

- mass = 0.029 kg;
- maximum velocity V_{max} = 2.5 m/s.

C. Mobility Model

To focus on network-driven coordination instead of flight dynamics, the drones use a lightweight kinematic mobility model.
Each node in the simulation is equipped with a mobility model; specifically, we developed a `CustomMobility` component that wraps around the existing `ConstantPositionMobilityModel`, a mobility model offered by *ns-3*. While the *ns-3* model only stores and provides node coordinates within a Cartesian space without inherent motion logic, our implementation introduces dynamic updates.
Explicitly, `CustomMobility` integrates commanded accelerations over time to update the node state. At each acceleration command or position request, the component computes the new position and velocity since the previous update, accordingly to the following fundamental equations of motion.

$$\mathbf{p}_{new} = \mathbf{p}_{old} + \mathbf{v}_{old}\Delta t + \frac{1}{2}\mathbf{a}\Delta t^2 \tag{1}$$

$$\mathbf{v}_{new} = \mathbf{v}_{old} + \mathbf{a}\Delta t \tag{2}$$

where $\mathbf{p}$ is the position vector, $\mathbf{v}$ is the velocity vector, $\mathbf{a}$ is the acceleration vector, and Δt is the time elapsed since the last update.
Furthermore, the model ensures physical consistency by enforcing a maximum speed constraint on each agent.

To visualize the simulation results, we integrated the `AnimationInterface`. This component generates an XML trace file based on the drones' mobility states stored in the *ns-3* mobility models, which can be subsequently rendered using the *NetAnim* visualization tool.

IV. Results and Analysis

A. Parameter Tuning

The parameters of the virtual spring-damper model were tuned through a systematic grid search.
The tuning process explores the parameter space defined by the triples $(K_{att}, K_{rep}, D_{safe})$ over user-provided minimum, maximum and step values for each parameter.
For each candidate configuration, the tuner executes the simulation executable and logs the resulting outputs into a repositioning CVS trace file.

The current score function is intentionally simple and is focused on suppressing oscillations while keeping the run valid:

$$\text{score} = \overline{o}_{first} + \overline{\overline{o}} + \overline{o}_{last} + P, \tag{3}$$

where $\overline{o}_{first}$, $\overline{\overline{o}}$, and $\overline{o}_{last}$ are the averages across drones that produced oscillation samples, and P is a penalty term. If no drone produces any oscillation samples (e.g., too few turning points), a large penalty is added to force the candidate to the bottom of the ranking. A second penalty is added if the minimum-distance metric cannot be computed from the trace.

In addition to the score, the tuner prints diagnostic indicators such as average minimum separation, average time-to-target, and whether all drones exhibit strictly decreasing oscillation amplitudes.
The simulation parameters are described in Table I.

The tuning process yields the configuration detailed in Table II. These parameters ensure a robust trade-off between convergence speed and system stability, enabling effective self-healing behavior while mitigating excessive oscillations.
These defaults can be overridden from the command line to test different configurations.

B. Evaluation Metrics

To evaluate stability and safety during the repositioning phase, the tuner derives a concise set of scalar metrics from the simulation logs. Each log entry consists of a tuple (t, id, x, y, z); the tuner aggregates these samples by drone ID to analyze individual trajectories.

TABLE I
SIMULATION PARAMETERS DESCRIPTION

Parameter	Symbol	Description
Base station coverage radius	R_{max}	Area inside which drones have a direct link with the base
Drone mass	m	Mass of a drone (CrazyFlie 2.1 in our simulation)
Max drone speed	V_{max}	Max speed (2.5x higher compared to the CrazyFlie 2.1)
Attractive gain	K_{att}	Forces attracting drones with different hop count
Repulsive gain	K_{rep}	Forces repulsing drones
Safety distance	D_{safe}	Distance below which the repulsive forces start acting
Controller tick distance	T_{ctrl}	Unit of time inside the simulation
Simulation duration	T_{sim}	Total duration of the simulation

TABLE II
SIMULATION PARAMETER VALUES

Parameter	Value
Base station coverage radius	50 m
Drone mass	29 g
Max drone speed	2.5 m/s
Attractive gain	1
Repulsive gain	5
Safety distance	1
Node tick distance	0.5 s
Simulation duration	600 s

Time-to-target. For each drone, the tuner measures the *settling time*, defined as the first instant the agent enters a tolerance band around a desired target position $\mathbf{p}_{target}$ (defined as the midpoint between the base station and the initial position of the disconnected drone). Formally, for each drone, the tuner computes:

$$T_{target} = \min \{t \mid \|\mathbf{p}(t) - \mathbf{p}_{target}\| \leq \varepsilon\} \tag{4}$$

With $\varepsilon = 1$ m as the tolerance threshold. If the condition is never met during the simulation, the tuner assigns $T_{target} = -1$.

Oscillation Analysis via Local Extrema. Rather than employing windowed statistics, the tuner estimates stability directly from the trajectory $p(t)$. It identifies a sequence of turning points $\{p_1, p_2, \ldots, p_n\}$ at timestamps where the discrete derivative Δp undergoes a sign change. From these extrema, the tuner derives a sequence of oscillation amplitudes:

$$o_i = |p_{i+1} - p_i|, \quad i = 1, \ldots, n-1 \tag{5}$$

For each agent, the tuner extracts the initial amplitude $o_{first} = o_1$, the final amplitude $o_{last} = o_{n-1}$, and the mean value:

$$\bar{o} = \frac{1}{n-1} \sum_{i=1}^{n-1} o_i \tag{6}$$

Additionally, a monotonicity check is performed to determine if the sequence $\{o_i\}$ is strictly decreasing:

$$o_{i+1} < o_i, \quad \forall i \in \{1, \ldots, n-2\} \tag{7}$$

This provides a binary indicator of the system's damped response and asymptotic stability.

Minimum inter-drone distance. To assess collision risk and safety violations, the tuner monitors the minimum Euclidean distance observed between all pairs of drones. Given the sequential nature of log processing, the distance is computed at each time step using the most recent position for each drone. This metric serves as a safety proxy to ensure that the repulsive forces effectively avoid collisions.

C. Simulation Results

The simulation scenario described in Section IV was executed using the default parameter configuration reported in Table II.

As the simulation runs, drones D_1 and D_3 receive the `HELP_PROXY` request from D_2 and switch to mission mode, applying the virtual spring-damper controller at each tick. Over time, both drones move toward the geometric midpoint between the base station and the disconnected drone D_2, forming a multi-hop relay chain that restores connectivity.

The trajectory of each drone during the simulation is visualized in Figure 1, where the repositioning behavior and final formation can be observed.

Regarding the run-level metrics, they are computed by aggregating drones data as averages (e.g., average o_{first} between drones that produced at least one oscillation value) and are summarized in Table III.

TABLE III
SIMULATION PARAMETER VALUES

Measurement	Value
Average First Oscillation	0.424 m
Average Oscillation	0.395 m
Average Last Oscillation	0.353 m
Average Min Distance	1.296 m
Average Time to Target	145.35 s

The observed trajectories reflect a trade-off in the parameter tuning. Specifically, the sensitivity analysis reveals that increasing the attractive gain K_{att} reduces convergence time but exacerbates the risk of overshoot, especially when the velocity saturation V_{max} is active. Conversely, higher K_{rep} or larger D_{safe} enhance safety margins; however, if repulsive forces dominate attractive ones, the swarm may fail to compress into a relay.

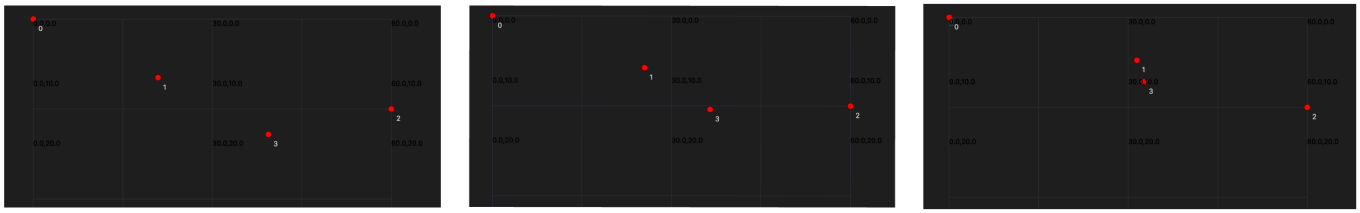

Fig. 1. Drone Trajectories During Self-Healing Simulation

Furthermore, as acceleration is derived from $\mathbf{a} = \mathbf{F}_{tot}/m$, the mass parameter m indirectly acts as damping-like factor on the control effort, with higher mass values leading to slower responses and reduced oscillations, at the cost of longer convergence times.

V. Conclusions

This work presents an autonomous self-healing protocol for drone swarms. The main objective is to restore connectivity to a base station following a link failure, relying exclusively on local communication and decentralized control. The proposed solution combines ACK-based reachability detection, flooding-based hop discovery, and periodic neighbor broadcasts with a formation controller based on a virtual spring-damper model, which is responsible for moving helper drones toward relay positions. Those relay chain positions are reached while maintaining a safe distance between nearby drones.

The protocol was implemented in *ns-3*, using a communication pattern inspired by *Crazyflie* platforms. In particular, unicast messages are sent toward the base station, while broadcast communication is used within the swarm. The system was evaluated in a scenario where one drone starts outside the base station coverage and sends a `HELP_PROXY` request. The results, together with the tuning process, show that the swarm is able to form a multi-hop relay chain. Oscillations remain limited, and the safety margins are acceptable under the tested parameters.

Beside these positive results, our implementation and simulation requires some preconditions and presents some limitations that may affect real-world scenarios. First, the presented self-healing protocol starts only under the condition where the disconnected drone is still within communication range of at least one helper drone: if the lost drone is completely out of range, no mission is triggered. Second, hop count values are determined by floods and the current ACK-based reachability signal. Before any more recent flood is observed, a disconnected node has an undefined hop value: during this transient phase, the controller treats most neighbors as having a different hop count, pulling helpers more aggressively until a stable gradient is established.
Finally, our `CustomMobility` model advances drone states based on elapsed wall-clock time between actuation calls. While this is sufficient for a qualitative evaluation, it introduces sensitivity to host machine load and prevents full deterministic reproducibility. Furthermore, the mobility model implements a simplified kinematic integration that neglects aerodynamic effects, environmental disturbances, and actuator dynamics. Such abstractions may lead to an optimistic estimation of the controller's effectiveness compared to real-world conditions.

Overall, despite the underlined limitations of a purely software-based simulation, the results suggest that a hardware validation could be feasible, especially in the context of *Crazyflie* swarms.

References

- Piemngam, K., Nilkhamhang, I. and Bunnun, P., 2019, October. A virtual spring damper method for formation control of the multi omni-directional robots in cooperative transportation. In 2019 11th International Conference on Information Technology and Electrical Engineering (ICITEE) (pp. 1-6). IEEE.
- Ouyang, Q., Wu, Z., Cong, Y. and Wang, Z., 2023. Formation control of unmanned aerial vehicle swarms: A comprehensive review. Asian Journal of Control, 25(1), pp.570-593.
- Kabore, K.M. and Güler, S., 2021. Distributed formation control of drones with onboard perception. IEEE/ASME Transactions on Mechatronics, 27(5), pp.3121-3131.
- Chen, Q., Veres, S.M., Wang, Y. and Meng, Y., 2015. Virtual spring-damper mesh-based formation control for spacecraft swarms in potential fields. Journal of Guidance, Control, and Dynamics, 38(3), pp.539-546.
- ns-3 Consortium, "ns-3 network simulator", https://www.nsnam.org
- ns-3 Consortium, "NetAnim — ns-3 Network Animator", https://www.nsnam.org/wiki/NetAnim
- Bitcraze AB, Crazyflie 2.x Nano Quadcopter, https://www.bitcraze.io