Swarm Dialogues:

Luis Yallico Ylquimiche

***Abstract*—LOREM IPSUM**

I. Introduction

This study profiles swarm communication on ESP32 hardware, evaluating embodied evolution performance under varied conditions such as agent density, locomotion, topology inference, imposed message budgets and stochastic transmission policies. By focusing on direct peer-to-peer communication links we provide (i) empirical data that probes bottlenecks flagged in recent surveys, and (ii) extract tangible design rules for swarm networks.

Swarm engineers draw inspiration from social biological systems such as ants, bees or termites to build decentralised robot collectives that are inherently robust to failure, flexible across tasks and scalable in number [1]. In swarm systems, collective intelligence emerges when individual robots trade packets of information among neighbouring robots. Classic ant-colony-optimisation work in the early 2000s has already proven that an indirect information exchange "virtual-pheromones" can lead to agents collectively discovering optimal routing behaviours [2]. Thus, highlighting the importance of communication design in swarms and the impact it can have over behaviour emergence.

While coordination and task allocation have been widely studied in swarm robotics, two recent surveys agree that bandwidth, latency and energy usage are the main blockers to real-world swarm deployments [3][4]. These issues become more pronounced as swarms sizes scale, resulting in an increase in data volumes being transmitted among peers, often overwhelming individual agents' limited compute capabilities. Beyond sheer capacity, the architecture of the communication also matters. Many swarms rely on blind broadcast communication schemes that scale poorly, with collision rates rising sharply beyond a few dozen peers which tend to reduce the reliability of the network [4].

Communication is equally critical when controllers evolve on-line. Recent embodied evolution studies explain that less communication can enhance swarm performance, as trimming neighbourhood size helps populations forget outdated beliefs and re-adapt faster [5][3]. Because most of these results stem from simulations, empirical evidence is still needed to further understand how agent density, throughput and packet content jointly affect evolutionary dynamics and adaptation performance.

Viewing communication as a dynamic resource that swarms must actively manage, we present a testing framework that integrates an island-model genetic algorithm (GA) with three distinct communication strategies. These experiments expand our understanding of how communication choices shape embodied evolution performance and supply data needed to tackle the integration challenges outlined above. Accordingly, the following three hypotheses are put forward:

H1: Ranking nearby peers by link quality before deciding where to send packets should form a sturdier network with fewer packets being lost.
H2: Even if each robot is allowed to transmit only a fixed number of messages, the group should still reach a common solution over time, the cap is expected to slow premature convergence, not prevent it.
H3: Inserting a brief randomly chosen delay before every transmission should reduce radio collisions and latency, as spreading out broadcasts should address interference in a congested channel.

II. Related Work

In [6], controllers are evolved for ad-hoc aerial relays without positional information, using periodic single-hop broadcasts to neighbours. This concept is leveraged on our experimental setting where the agents exploit link-quality proxies to bias their connections. In non-local communication schemes such as [7], network-wide broadcasting is used due to the accelerated diffusion of information, with benefits in search and rescue applications. In contrast, our study uses *unicast* peer-to-peer links at the data-link layer, enabling direct round-trip time measurements.

Constrained connectivity repeatedly appears beneficial. The *"less is more"* effect in [8], demonstrates that fewer links and lower swarm densities improve adaptation in swarm consensus tasks, by helping robots discard stale beliefs. A similar effect is also observed by [5] over embodied evolution, where limiting the genome-exchange range can support the evolutionary search maintain higher diversity for longer and escape local optima. The expectation is that by imposing message quotas and prioritising "distant" islands, the swarm could avoid premature convergence.

Transmission timing also matters. Experiments from [9] show that limiting the frequency of communication stabilises swarm behavior, while [10] explains that the speed of consensus among peers tolerates random transmission delays even when agents receive multiple out-of-order messages.

Furthermore, non-radio data links have also been explored by [11], which shows that infra-red (IR) line-of-sight local communication can be used with benefits such as lower energy requirements per bit and reduced 2.4GHz interference information exchanges albeit with limits on bandwidth and communication range. Complementary work by [12] examines real-time, decentralised peer-to-peer middleware at the application layer, addressing IP protocols. It provides a contrast for where to place communication control in swarm systems, whereas we implement this at the MAC layer via ESP-NOW to keep timing and load directly measurable.

Over-the-air (OTA) mechanisms tailored to swarms have been explored by [13], where updates are distributed as compact patches using gossip-style communication protocols to ensure version uniformity across large robot groups. We adapt this concept in Section-IV-A1 to support synchronised firmware across our deployment.

III. Experimental Setting

In this study, a global optimisation problem known as the Rastrigin function Eq. 1 is used to benchmark the communication performance. This task was chosen to emulate evolutionary controller optimisation, while no controller was actually evolved the concept of robots sending and receiving genomes from their peers remaines the same.

The environment for the experiments was a rectangular arena without obstacles (Figure 1), where the initial positions, and number of the robots was determined by the experiment schedule (Table VI). All of the experiments were conducted in the same room environment within the department of Engineering Mathematics at the University of Bristol.

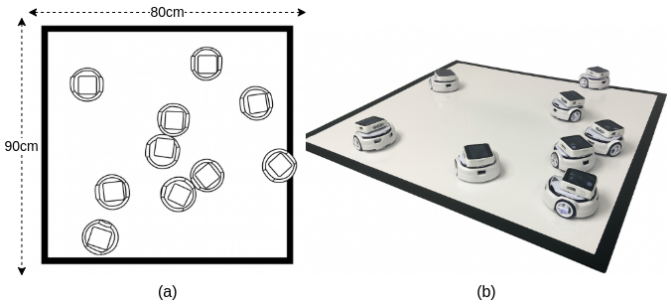

Fig. 1. Robot Arena: (a) dimensions and (b) example of a Brownian locomotion experiment with 8 robots.

Across the study, we evaluated swarm under different communication and system configurations. Each experiment manipulates a specific independent variable while holding all other conditions constant. These include:

- **Swarm density**: The number of agents deployed simultaneously, ranging from 3 to 13.
- **Locomotion**: Robots were either *stationary* or navigated using a *Brownian motion* gait (Section III-B1).
- **Topology inference**: Transmission priority was governed by either a *stochastic* shuffle or a *comm aware* ranking strategy based on link quality metrics (Section IV-D1).
- **Message limits**: A token-bucket rate limiter controlled how frequently agents could transmit messages (Section IV-D2).
- **Transmission frequency**: Each message was optionally delayed by a random interval derived from the maximum observed peer latency to reduce potential network collisions (Section IV-D3).

A. Rastrigin Function

The Rastrigin function is defined as follows:

$$f_R(\mathbf{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right) \tag{1}$$

where n is the number of dimensions, in this case the number of genes. Whereas, x is the individual genome being evaluated. The function has a global minimum at $f_R(\mathbf{x}) = 0$ when $x = [0, 0, ..., 0]$ for all dimensions [14]. For the purpose of this study we used $n = 10$ and a solution bounded at $-5.12 <= x_i <= 5.12$.

B. Robot Platform

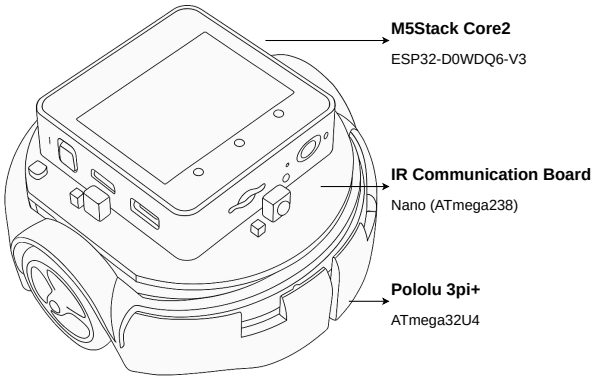

Fig. 2. The Swarm-B2 platform: M5Stack Core2, IR Communication board, and Pololu 3Pi+

Table I summarises the core hardware components of the Swarm-B2 platform used in this study [REF]. Coordination between devices is handled over a I^2C 100kHz bus. Although the IR board can transmit and receive 32-byte frames, we did not conduct experiments with this feature. The board therefore behaves as a I^2C bridge between the M5 and the Pololu 3Pi+. This guarantees that the communication results in this study stem only from a single data link.

The ESP32 was used for running dual-core FreeRTOS parallel tasks, logging data on the local SD card and handling communication via the ESP-NOW data link (2.4 GHz). The 8MB PSRAM and larger flash memory on the master device

TABLE I
SWARM-B2 HARDWARE STACK

Component	Interface(s)	Function
M5Stack Core2 (240 MHz)	I^2C master and SPI	Embodied evolution, ESP-NOW communication, data logging to SD and user interface
IR board (16 MHz)	I^2C slave (0x04)	Bus hub, optional IR feature
Pololu 3Pi+ (16 MHz)	I^2C slave (0x08)	Locomotion, bumper and line following sensors

allowed for concurrent task execution without peripheral starvation.

1) Locomotion

The Pololu 3Pi+ is equipped with a line following sensor array and two bump sensors, which can be used to detect obstacles and detect the arena edges. A *Brownian-motion* gait was selected to maintain unbiased mobility across the arena and ensure constant movement.

The gait code exposes the Pololu 3pi+ slave to the ESP32 master node, this interface lets the ESP32 act as a passenger with override, that can set wheel-speed scaling factors or raise `START`/`STOP` flags without touching the low-level control loop. In effect, the Pololu 3pi+ driver handles continuous motion, while the ESP32 decides when to go for each experimental condition.

2) Local Data Storage

We implemented local data logging mechanism (Section IV-E) on the ESP32 with a 16GB SD card peripheral. The shared SPI bus (with the LCD) was set at a frequency of 20MHz and configured to use the FAT32 file system for storage. This approach was chosen to emulate a realistic swarm system capable of operating remotely without relying on a stable Wi-Fi connection to a central server.

3) User Interface

Figure 3 shows the on-board user interface (UI) each Swarm-B2 agent displays during trials. A real-time clock (RTC) seeds a unique *experiment_id*, just below a $T \pm SS$ counter tells the operator how long until the next minute aligned run. Status lines on the display list the robot's ID (last four hex digits of the MAC address), the live fitness score, and a single debug message. The bottom-left tag logs which software build is running on the device. Two icons round out diagnostics, the Wi-Fi symbol flashes during S3 log upload, and the circular arrow signals an over-the-air update.

IV. IMPLEMENTATION

This section outlines the software design and implementation of the swarm firmware. The Espressif IoT Development Framework (ESP-IDF) was used as it provides low-level hardware access, offering greater flexibility for ESP-NOW communication (Section IV-D). Unlike the Arduino

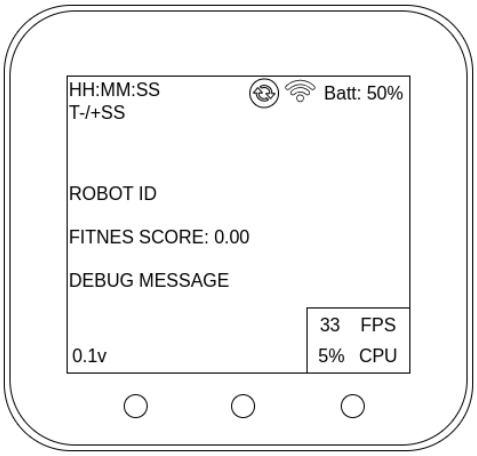

Fig. 3. M5 user interface showing the current fitness score and device information.

framework, ESP-IDF provides direct access to FreeRTOS, enabling fine-grained control over concurrent task creation and dual-core processing, such as isolating communication tasks to a specific core. It also supports advanced features like over-the-air (OTA) updates (Section IV-A1), unit testing, and custom debugging tools [15].

A. Software Development Environment

We employed a Continuous Integration and Continuous Deployment (CI/CD) pipeline via GitHub to automate OTA deployment, ensuring synchronized updates and easier debugging across the swarm (Fig. 4).

The CI/CD integration proved especially valuable during experiments, where consistent updates across multiple agents were necessary. It also facilitated easier rollbacks in the event of unexpected bugs. The automated build pipeline in GitHub ensured that only validated firmware versions were propagated to the swarm, catching any environment discrepancies early in the process.

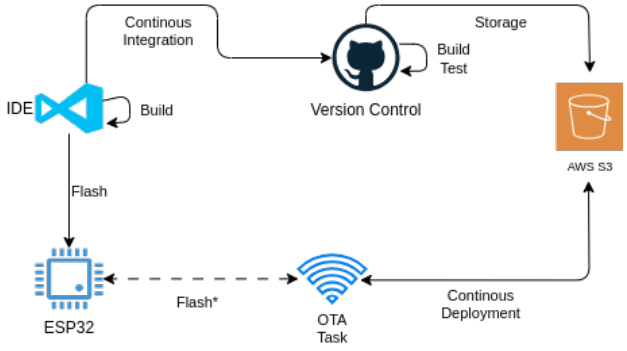

Fig. 4. CI/CD Architecture: Local development, GitHub repository, AWS S3 storage, and OTA updates.

Our software development framework for the swarm is as follows:

1) **Local environment development**: Application development takes place locally using VSCode, our local environment uses version 5.1.4 of ESP-IDF and Python 3.11 to build and flash the code in-situ.

2) **Version control**: We push updates of the codebase to a public repository on GitHub: https://github.com/yallico/robotics_dissertation, this allows for version control and triggers a custom build and test process. The ESP32 project is then compiled remotely and generates the binary file used for OTA.

3) **Cloud storage**: The OTA binary file is uploaded to an AWS S3 public bucket making it accesible to the swarm via HTTPS.

1) OTA Process

Upon initialization, each agent checks its local firmware version against the latest version stored in S3. If a mismatch is detected, the ESP32 downloads and installs the updated .bin file. While relying on a central server and exposing the swarm to the internet for updates may seem counterintuitive, a similar update mechanism could be implemented in a decentralized manner using consensus protocols. However, this would have introduced additional complexity beyond the scope of this study.

TABLE II
SYSTEM PARTITION TABLE

Name	Type	Size	Description
`nvs`	data	16KB	Non-volatile storage
`otadata`	data	8KB	OTA metadata
`phy_init`	data	4KB	PHY layer calibration data
`factory`	app	4MB	Default application
`ota_0`	app	4MB	OTA slot 0
`ota_1`	app	4MB	OTA slot 1

Table II shows how each device partitions was configured with a dual-partition OTA scheme with two application slots: `ota_0` and `ota_1`. During an update, the new firmware is written to the inactive partition. Once the write and integrity checks pass, the bootloader switches to boot from the updated partition on the next reboot. This allows safe rollback in case of update failure. The update binaries ranged from 1 to 1.2 MB, note that update propagation and version control were managed manually through a quick check of the robot's LDC display before the experimental run.

B. Embodied Evolution

The swarm uses a distributed genetic algorithm (GA) to find the global minimum for Eq. 1.A visual representation of this is shown in Fig. 5. As the local population in each agent evolves, the swarm begins to communicate their local best fitness and corresponding genes to their peers.

Our implementation employs an elitist migration strategy, this happens when the local GA reaches a patience threshold and the agent pushes its "best" (lowest fitness score) genome to other swarm members via ESP-NOW. The incoming remote genes from another peer are integrated into the local population by replacing the worst performing 5% individuals, this value was chosen to preserve genomic high locally.

To avoid stagnation over a local-minimum, a mass extinction event together with a hyper-mutation mechanism tracks consecutive non-improving generations. Once a set of conditions is reached (Table X), the mutation probability is temporarily increased to escape local optima and lowest performing half of the population is re-initialised. This is done to promote exploration across the swarm and prevent premature convergence.

C. Real Time Operating System

Each swarm member boots into a FreeRTOS runtime by calling `app_main()`, which performs the initialisation of the following components: non-volatile storage, I2C peripherals, RTC, SD card, and ESP-NOW.

Figure 6 illustrates the sequence of these and their relation to the tasks that are spawned during run time. These tasks include:

- `i2c_task`: Handles communication with the I^2C peripherals, including the AXP192 power supply, the IR board, the display and the Pololu 3pi+.
- `gui_task`: Manages the UI on the M5Stack display, used for real-time feedback and debugging.
- `pololu_heartbeat_task`: Handles the I^2C signal from the Pololu 3pi+, ensuring that the robot is operational and responsive.
- `ota_task`: Manages OTA updates if a new version is available in S3 (Section IV-A1).
- `espnow_task`: Manages ESP-NOW communication between swarm members, handles message sending and receiving.
- `ga_task`: Runs the local GA and coordinates with other tasks to log and transmit data.
- `write_task`: Handles SD card operations, including data logging, managing file storage and uploading experimental results.

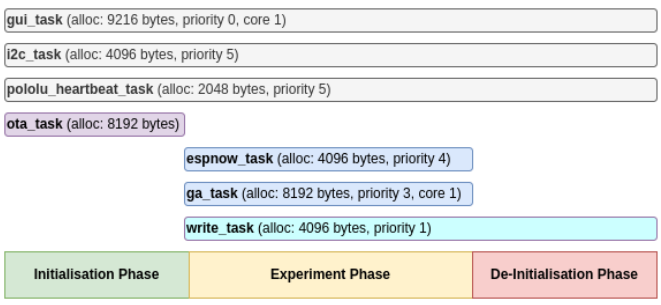

Fig. 6. Task schedule with memory allocation and priority.

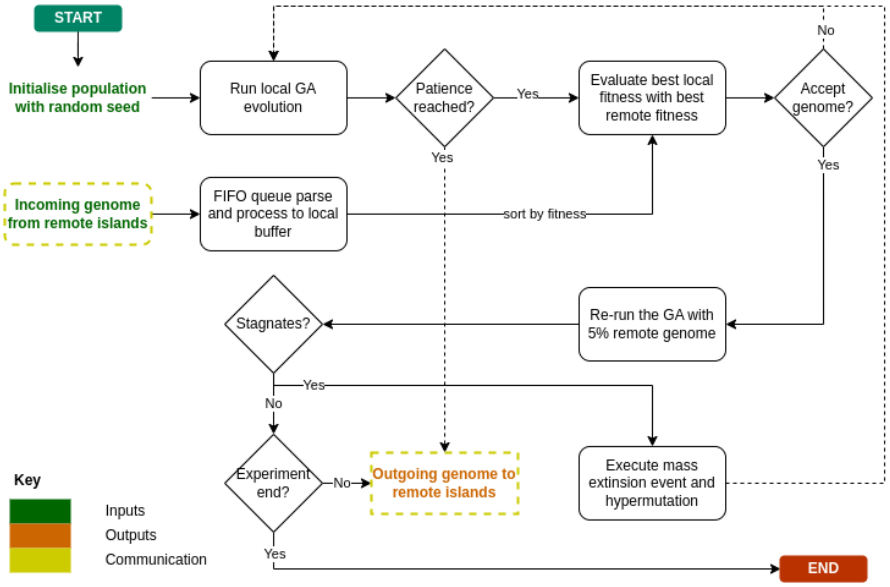

Fig. 5. Island Model Flowchart

Inter-task coordination is managed by event groups and queues. The event groups are used to signal task completion and synchronise downstream operations, whereas queues are used to pass data between tasks that are operating in parallel. This enables multitasking and real-time processing of the measurements.

Note that the local population undergoes evolution until the experiment phase is terminated. The experiment phase is terminated when one of the following criteria is met:

1) **Global solution**: Any robot attains the exact global minimum of the Rastrigin function (fitness = 0).
2) **Time limit**: A maximum duration of 60 seconds elapses without global convergence.

Upon reaching the de-initialisation phase, the `espnow_task` drains pending message queues, final fitness and communication metrics are uploaded to an Amazon S3 bucket via HTTPS via the `write_task`. Finally, the system peripherals are de-initialised to ensure a clean shutdown.

D. Communication Layer

We implemented ESP-NOW by pre-registering the MAC addresses for all peers in each agent, enabling direct unicast communication between swarm robots. Our design uses push only event-driven peer-to-peer messaging, where each device transmits data to peers without waiting for requests. Using this scheme avoids the complexity of pull-based communication protocols. Recall that once a local elite genome is found, the best solution is sent to all peers, if there are incoming messages these are queued for later processing, allowing task operations to continue uninterrupted. In this section we describe the specific communication independent variables that were manipulated during experiments.

1) Topology Inference

We investigate the impact of different communication schemes on the emergent swarm topology by introducing two distinct message transmission strategies: **STOCHASTIC** and **COMM_AWARE**. Implemented in the communication layer, these were designed to influence the order and priority with which each agent sends data to its peers. The aim is to understand how these strategies affect the connectivity and robustness of the swarm network.

These are implemented as follows:

- **STOCHASTIC**: Each agent calls a random seed to apply a Fisher-Yates shuffle over the list of peer MAC addresses to sort them before sending its message. This ensures that the order of communication is random for each transmission cycle, preventing biases.
- **COMM_AWARE**: Each agent ranks its peers based on the most recent measurements of communication quality, specifically the last known latency and Received Signal Strength Indicator (RSSI). Peers with unknown metrics (during initialisation) are prioritized first to ensure all peer links are measured. Then, peers are scored by normalizing both latency and RSSI, and those only in the worst half (highest latency, lowest RSSI) are prioritized for message transmission.

The following pseudocode outlines the logic for each scheme:

Algorithm 1 Randomized Peer Selection

Input: List of peer MAC addresses
Fisher-Yates shuffle using a random seed
for each peer in shuffled list **do**
    if peer is not self **then**
        Send message to peer
    end if
end for

Algorithm 2 Communication-Aware Peer Ranking

Input: List of peer MAC addresses, last known RSSI and latency for each peer
for each peer **do**
    if RSSI or latency is null **then**
        Assign highest priority
    else
        Normalize RSSI and latency across all peers
        Compute score: $score = norm\_latency + norm\_rssi$
    end if
end for
Sort peers: null metrics first, then by descending score (worst first), only **50% scope of network**
for each peer in sorted list **do**
    if peer is not self **then**
        Send message to peer
    end if
end for

Note that these schemes are specifically designed for **unicast** communication, where messages are sent directly to individual peers and round-trip latency can be measured via acknowledgements (ACKS). Having said that, we can think of the **STOCHASTIC** algorithm as a pseuso-broadcast communication scheme as the message is sent to all peers with negligible delays between transmissions.

2) Limited-Rate Communication

Inspired by the "less-is-more" effects reported by [9] using infrared links, we implemented a token-bucket limiter to the ESP-NOW layer and treat this quota as an independent variable. Each agent is given a small budget of 1 message in a sliding window of length 8 seconds. When the bucket is empty the agent must keep silent until the window refreshes, regardless of how often its GA stagnates or improves. This caps the total interaction rate per robot rather than solely spacing individual transmissions.

Algorithm 3 Token-Bucket Throttled Send

Require: B: message budget per window, W: window length (ms), t_{last}: window start time, $tokens$: remaining sends
procedure MAYBESEND($payload$)
    $now \leftarrow$ CURRENTTIMEMS
    if $now - t_{\text{last}} \geq W$ **then** ▷ Window refresh
        $tokens \leftarrow B$
        $t_{\text{last}} \leftarrow now$
    end if
    if $tokens = 0$ **then**
        return ▷ Bucket empty → no send
    end if
    if IMPROVEDFITNESS **then**
        $tokens \leftarrow tokens - 1$
        ESP_NOW_SEND($payload$)
    end if
end procedure

3) Transmission Frequency

To further explore the communication behaviour of the swarm under flooding conditions, we modulate the transmission frequency. In this communication mode, each message transmission is delayed by a random interval, otherwise no delay is explicitly applied. This random delay, drawn from a range determined by the maximum observed latency among peers, is described by Algorithm 4.

Algorithm 4 Stochastic Transmission Frequency

Input: List of peer MAC addresses, maximum latency observed (max_rand)
for each peer in the target list **do**
    if peer is not self **then**
        Compute a random delay: $delay \leftarrow rand(0, max\_rand)$
        Wait for $delay$ milliseconds
        Send message to peer
    end if
end for

In the firmware, this is implemented by checking if `DEFAULT_MIGRATION_FREQUENCY` is set to `FREQUENCY_RANDOM` and, if so, randomly delaying each call to `esp_now_send` by a value within the range of $[0, max\_rand]$, where max_rand is derived from the maximum measured latency among peers. The intention of this approach is to control the flooding of messages in a stochastic manner that can help reduce unintended collisions in the swarm network.

E. Data Logging

Reliable and precise data logging is a pre-requisite for evaluating the communication performance and evolution dynamics of the swarm. To achieve this, our firmware implements several mechanisms to capture and record key

metrics such as latency, message exchanges, internal state changes, and experiment metadata. These measurements are logged using well-defined data structures and are incrementally written to an SD card.

1) Messaging Structure

Table III summarizes the `out_message_t` structure used for transmitting messages between swarm peers via ESP-NOW. Note that any floating point values in the message content are rounded to $3d.p.$ to ensure compact representation. The total size of the struct is kept within the raw payload limits (250 bytes) imposed by ESP-NOW to guarantee reliable transmission.

TABLE III
MESSAGE STRUCTURE (`OUT_MESSAGE_T`) FOR ESP-NOW DATA TRANSFER

Field	Type	Description
`log_id`	uint32_t	Unique internal identifier of the event.
`robot_id[5]`	char	MAC identifier for the sender robot plus a null terminator.
`created_datetime`	time_t	Timestamp based on the internal RTC.
`message[128]`	char	Content of the message including the fitness score and genome delimited by "|".

2) Data Processing

The data logging pipeline is designed to ensure that all key experimental metrics are captured and preserved locally by each agent. Log entries are posted to two FreeRTOS queues: `LogQueue` for internal logging events and `LogBodyQueue` for detailed message logs. A `QueueSet` allows the dedicated `write_task` to efficiently monitor and process both queues in real time.

The core logging framework relies on three primary data structures:

- `experiment_metadata_t`: Stores overall experiment parameters, including experiment and robot IDs, random seed, GA parameters, migration settings, and application version.
- `event_log_t`: Used for logging system events such as latency measurements, RSSI, CPU usage, and state changes.
- `event_log_message_t`: Holds the parsed version of the `out_message_t` structure and links it to the internal event ID.

When an entry is retrieved from either queue, it is serialized into a JSON-formatted string (via `serialize_log_to_json()`) to ensure structured and consistent downstream analysis. The serialized data is then written incrementally to SD card files, with each file capped at 1MB to prevent memory overflow and ensure robust storage. Figure 7 illustrates the internal data logging pipeline, from event generation to SD card storage.

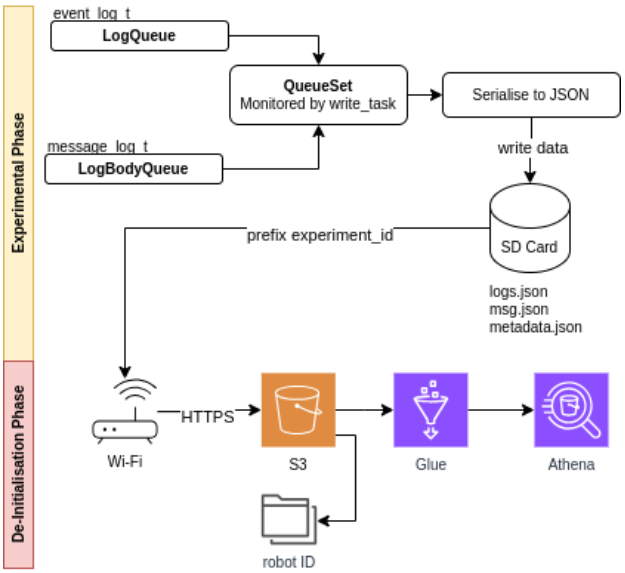

Fig. 7. Internal data logging pipeline

After the experimental phase concludes, the system enters a data upload mode. In this mode, the files stored on each agent's SD cards are uploaded to an AWS S3 bucket via HTTPS. This was done by design as we did not want to stream the data from the peers whilst they were running the experiment, as this could have introduced additional collisions potentially impacting the measurements.

As depicted by Figure 7, the S3 bucket is structured to store each agent's data in a separate folder, named by the robot's ID. Each file is named with a timestamp and the experiment ID, ensuring that all data is uniquely identifiable and traceable. The data is then processed using AWS Glue to prepare it for analysis using Amazon Athena. This process allowed us to automate the storage and evaluation of data logs from over 500 experiment runs.

F. Communication Performance Metrics

To evaluate the quality of network communication in the swarm, the following metrics were computed, (i) L *latency (ms)*, the time taken for a message to be sent and acknowledged by a peer, (ii) J *jitter (ms)*, the variation in latency between messages, (iii) P *packet loss (%)*, the percentage of messages that were sent but not acknowledged, and (iv) T *throughput (kbps)*, the rate at which data is successfully sent and received by each agent.

To assess overall network performance, a Quality of Service (QoS) score was computed for each run using a weighted sum of normalized network metrics. The equation is defined as:

$$\text{QoS} = w_0 \cdot (1 - \hat{L}) + w_1 \cdot (1 - \hat{J}) + w_2 \cdot (1 - \hat{P}) + w_3 \cdot \hat{T} \quad (2)$$

Here, each metric is normalized to the range [0,1] and the weights w_i are defined in Table IV. A higher QoS value indicates better overall network communication performance.

TABLE IV
QoS UTILITY FUNCTION WEIGHTS FOR DIFFERENT SWARM ROBOTICS APPLICATIONS

QoS	w_0	w_1	w_2	w_3	**Application**
QoS_c	0.5	0.25	0.15	0.1	Swarm consensus and voting
QoS_s	0.3	0.15	0.45	0.1	Swarm sparse deployments

V. PRELIMINARY ANALYSIS

Using a single robot, a preliminary analysis was conducted to identify suitable parameters for solving the Rastrigin function under varying population sizes (10, 20) and gene dimensions (2, 3, 4, 5). Over 100 experimental runs were performed, each terminating if fitness failed to improve beyond a 0.001 threshold over 20 consecutive epochs. As shown in Figure 8, larger populations yielded lower median fitness scores.

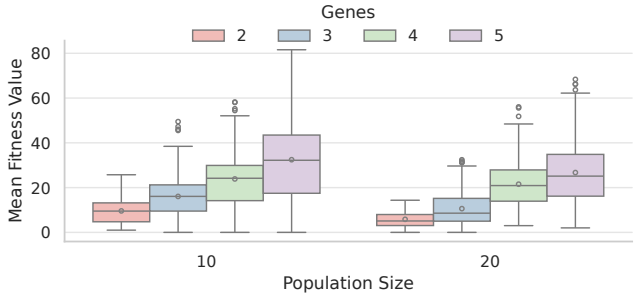

Fig. 8. Fitness distribution by population size and gene dimensionality

The time taken to converge ranged between 0 to 3 seconds, reflecting the early stopping triggered by the patience setting. The data gathered from these early experiments suggested that using five genes was not sufficiently challenging for a single agent as convergence was achieved too quickly. Due to this we decide to expand the Rastrigin search dimensionality to 10 genes and extend the local population and threshold epochs to 60. Hypermutation parameters are also set at this stage and implemented the most practical experiment run time, firmware improvements are summarised in Table V.

TABLE V
FIRMWARE UPDATES AND CHANGE LOG

Version	**Notes**
v0.3	Single agent experiments, GA parameter tuning.
v0.4	Improved late message queue handling. Use 10 Genes.
v0.5	Reduced experiment time from 120 seconds to 60 seconds.
v0.6	Increased queue size to 40 (previously 25), to ensure no messages are lost.

With two agents, we built a baseline shown in Table VI. The dual agents did not reach the global minimum in these trials, but consistently converged to a local minimum within the first 20 seconds of the experimental phase.

VI. RESULTS

As shown in Figure 9(a), the latency and jitter of the swarm network differ significantly between topology inference schemes. Across experiments the *COMM AWARE* topology strategy generally achieves lower latency and more stable delivery compared to the *STOCHASTIC* mode. Figures 9(b) and 9(c) show that throughput is highest in unconstrained conditions, and drops with the introduction of message budgets. *STOCHASTIC* communication yields higher raw throughput but also has a wider range of variability.

Figure 10(a) shows that larger swarms converge to lower mean fitness scores, indicating improved global convergence as agent density increases. Figure 10(b) presents the adaptation rate over time. Swarms using the *STOCHASTIC* communication scheme tend to converge more rapidly, with approximately 80% of the agents reaching the best-known solution roughly 20 seconds earlier than those using the *COMM AWARE* strategy. Here, adaptation rate is defined as the proportion of agents that share the current lowest fitness score, averaged over 2-second windows.

Figure 11(a) shows the mean packet error rate across different swarm densities. The error rates remain low overall but increase with the swarm size, and it is particularly impacted by the *STOCHASTIC* scheme (Figure 11(b)).

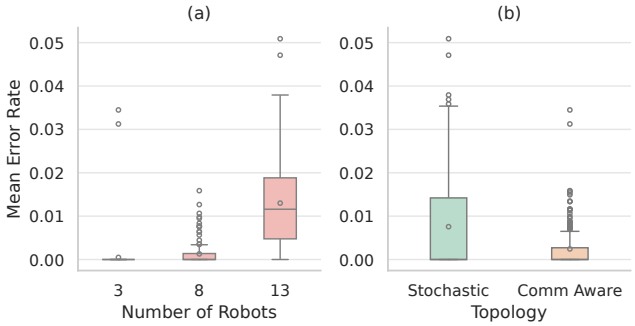

Fig. 11. (a) Mean error rates by device across varying swarm sizes, (b) mean error rate for different topology inference schemes

With regards to locomotion Figure 12(a) shows that Brownian motion results in slightly improved RSSI values, indicating more stable connections compared to static swarms. Whereas 12(b) suggests no statistical difference between RSSI values across topology inference schemes.

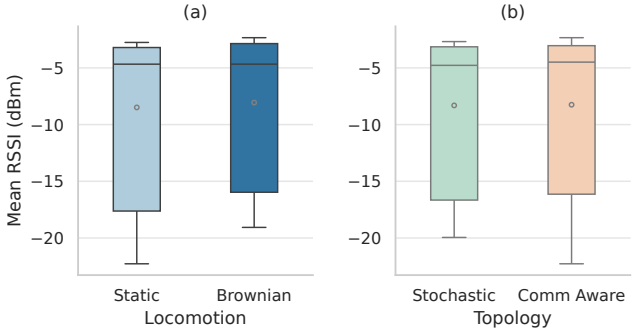

Fig. 12. (a) Mean RSSI values across different locomotion modes, (b) mean RSSI values accross different topology inference schemes

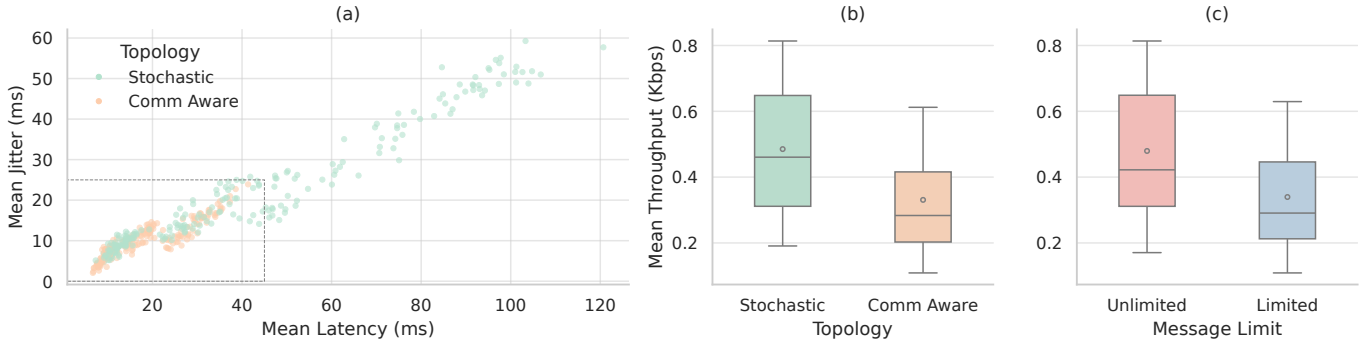

Fig. 9. (a) Latency and Jitter relationship under varying topology schemes, (b)(c) throughput distribution under topology and message limit variables

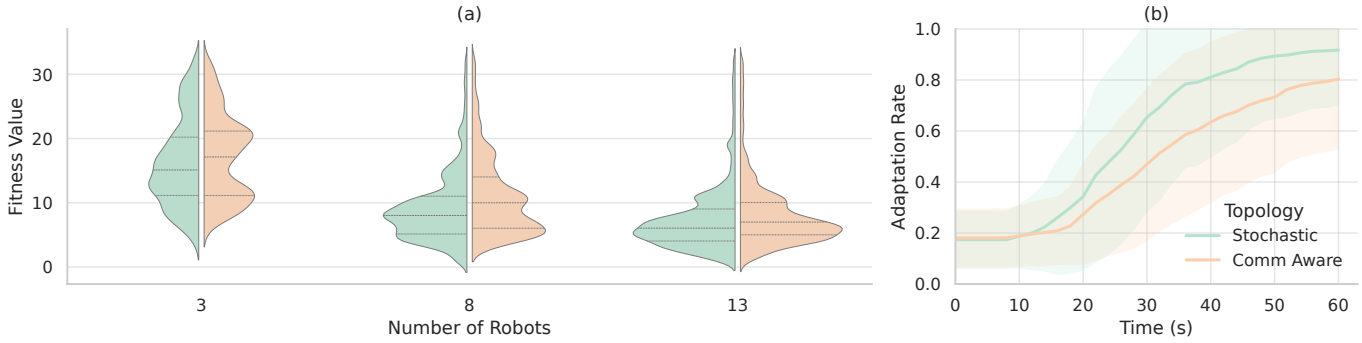

Fig. 10. (a) Mean device fitness values across different swarm sizes, (b) mean cumulative percentage of agents converging to the lowest fitness value across $2s$ windows

Considering the impact of transmission frequency for experiments under the STOCHASTIC scheme, Figure 13(a) shows that modulated transmission policies also restrict the network to lower latency and jitter though not to the same degree. Figure 13(b) illustrates how latency varies over time under different transmission settings, with modulated policies achieving lower more stable latencies. Figure 13(c) shows that throughput is generally similar between policies, a contrast compared to the topology inference schemes. Similar to performance under varying topology inference conditions, the performance for alternating transmission policies remains impacted mainly by the increase in swarm density Figure 14(a). Moreover, the adaptation rate between transmission policies remains similar Figure 14.

An evaluation of the characteristics of the QoS metrics for the 13-agent swarm is shown in Figure 15. Key observations include, (i) COMM AWARE topology inference achieves high QoS metrics, (ii) modulated transmission also scores high in QoS but tends to score higher in QoS_s, (iii) message limits do not have a clear effect on QoS, both kernel densities overlap each other.

VII. Discussion

Results from the effects of the topology inference suggest that the primary driver in the reduction of latency ¡50ms is driven by the number of links being formed rather than the number of messages being limited. This is because although we observe a similar reduction in throughput for variables, the effect size over latency and jitter is not as pronounced when messages only budgeted. The concept that number of links formed over time drives latency reduction is elicidated in Figure 16, which show the pseduo-flooding effect of the *STOCHASTIC* scheme, over a similar time periods it yields more agents connecting to one-another whereas on the *COMM AWARE* regime the formation of local pockets of information is possible due to the number of link formation being capped. The effect of having slower adaption rates in an elitist island model such as this one supports the idea that avoiding premature convergence between agents is beneficial from an evolutionary perspective, as it allows for more diverse genome to be explored for longer time spans. Conversely, overall fitness values for Rastrigin are mainly influenced by an increasing number of agents, supporting the theory that under more genetic variation results in an improved search.

The results suggest that using *COMM AWARE* achieves high QoS values as jitter and latency remain low whilst error rates are also abated, with the main trade off for this being a reduction in overall throughput of the network. Although not explicitly shown on the results one the key observations that we made is that the impact on QoS scores is less varied on swarm densities below 13 agents, in which swarms of 3

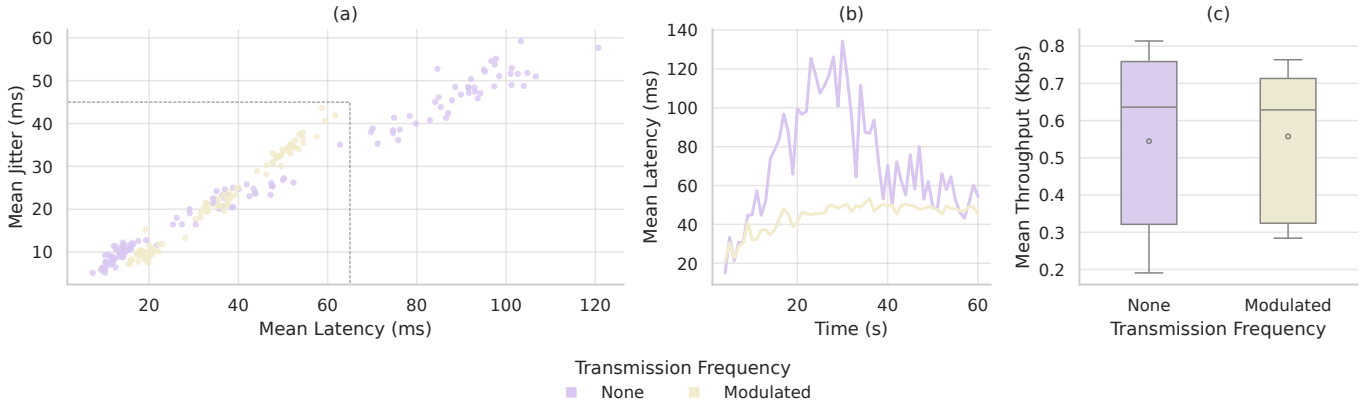

Fig. 13. (a) Latency and Jitter relationship under varying transmission policies, (b) latency across time by transmission setting, (c) throughput distribution by transmission setting

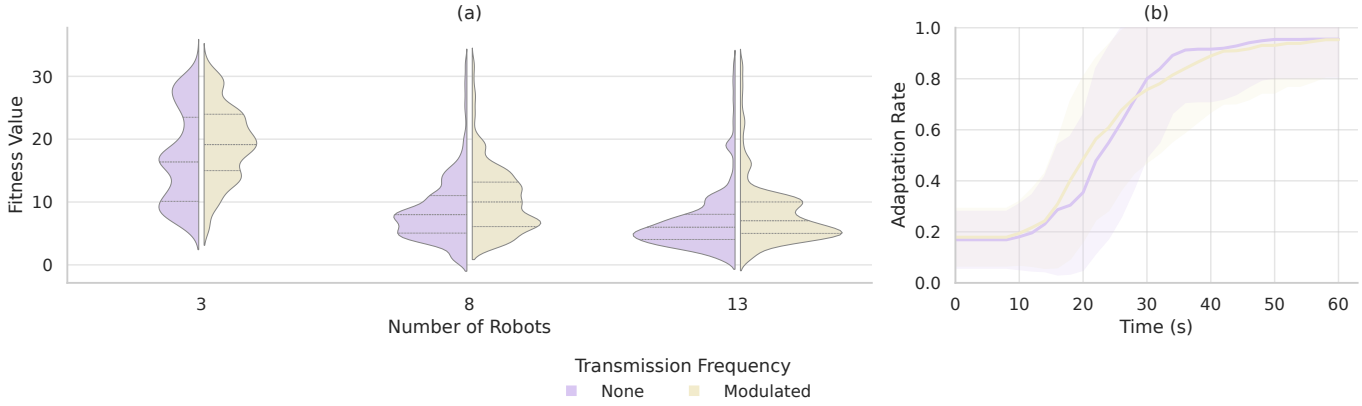

Fig. 14. (a) Mean device fitness values across different swarm sizes, (b) mean cumulative percentage of agents converging to the lowest fitness value across $2s$ windows

and 8 scored high values irrespective of which variable was being isolated, this could potentially mean that at a particular swarm density collision rates become more prominent over the network as the number of connection increases (Figure 17) . This is also supported by the fact that the error rates are higher in larger swarms (Figure 11).

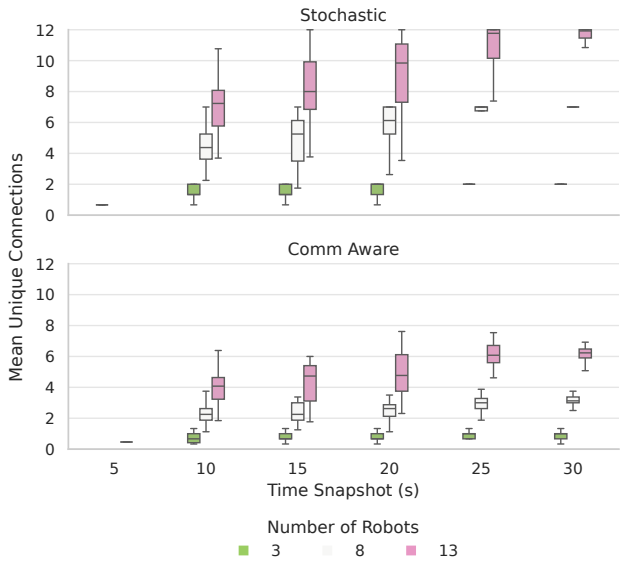

Fig. 17. Per-peer number of mean connections formed by time snapshot up to 30 seconds for both topology inference schemes

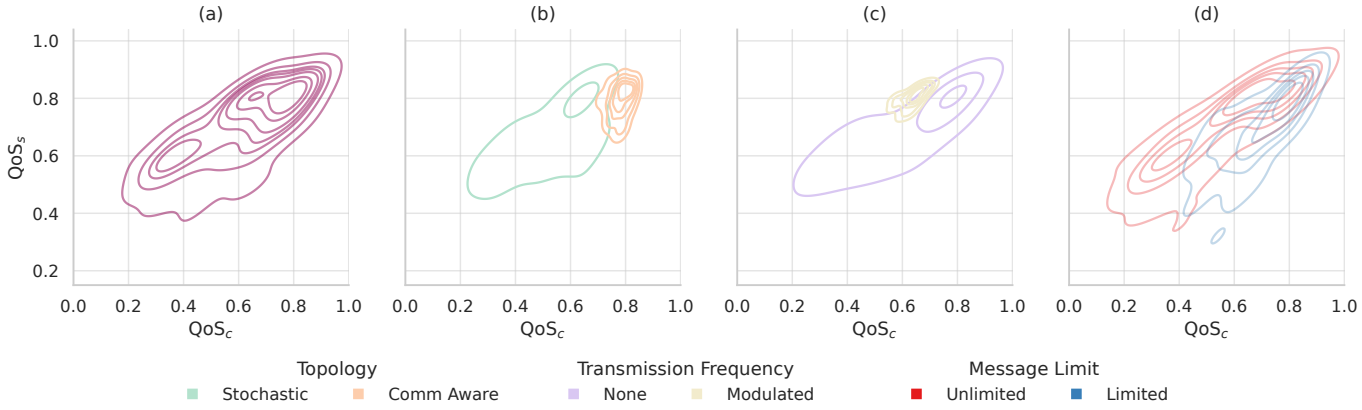

Fig. 15. Kernel-density contours showing the joint distribution of two QoS metrics (x: QoS_c, y: QoS_s) for a 13-agent swarm. (a) pooled across all experimental factors, (b) split by communication topology, (c) split by transmission frequency policy, (d) split by message limit setting

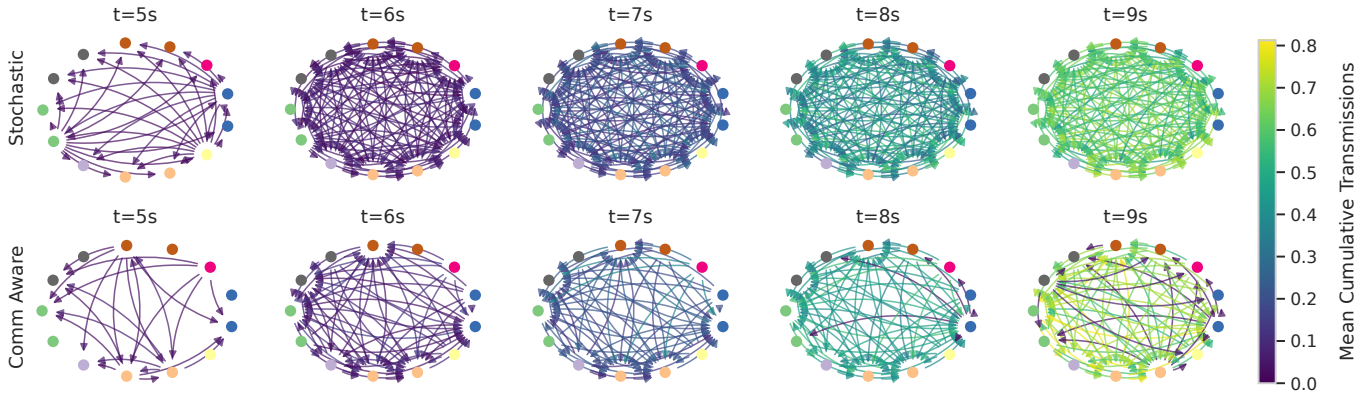

Fig. 16. Temporal snapshots of inferred communication topologies for a 13-agent swarm. Rows: inference scheme, columns: time samples. Nodes are on a fixed circular layout, edges show inferred links and its colour indicates mean cumulative transmissions per node.

Regarding the effect of RSSI, results shows that during the experiments there was no significant difference between the topological inference schemes, this was expected as the maximun possible distance that the agents were sending messages across was around $1m$, this means that the *COMM AWARE* was mainly driven biased by latency instead of RSSI, further studies on sparse swarms would be needed to see if RSSI could be used in combination with latency to obtain similar results. Interestingly, we note a small effect in the improvement of RSSI values when Brownian locomotion is induced in the swarm, this is might be due to the fact that the agents are not stationary and therefore they are able to move around and find better communication links with their peers having said that further investigation is needed to fully understand this phenomenon.

On the other hand, frequency modulation experiments all of which under the *STOCHASTIC* scheme suggest an alternative approach to conventional flooding, as suggested by [10] the swam can tolerate transmission delays as shown by the similar fitness values achived with and without the modulation. More importantly, bounding the stochastic transmission delay to latency observer per-peer yields a smoothing effect over latency over time (Figure 13(b)) but is capably to maintaining similar thoughtputs to the unconstratined flooding with increased latencies and jitter as a trade-off. Another key observation is that frequency modulation maitains high adaptation rates compared to the *COMM AWARE* scheme, resulting in fewer pockets of stale information. We observe that this type of communication policy shifts the QoS scores towards QoS_s that prioritises sensing tasks in a swarm that favor higher throughpus and lower error rates. In [7], we note that this type of communication might be beneficial in search and rescue operations where the goal is to quickly disseminate information about the environment and adapt to changing conditions.

The evidence suggests that the throughput is not the primary limiting factor over the performance of the elitist island-model. The message OUT_MESSAGE_T that was exchanged by the swarm was kept constant with 10 genes and size of 145 bytes and transmitted over a single channel, yet we know from [6] that multiple channels can be used in swarms. We yet do not know what the impact of the data structure might have over the

TABLE VI
EXPERIMENTAL CONFIGURATIONS

Robots	Topology	Locomotion	Message Limit	Transmission	Latency (ms)	Jitter (ms)	Error Rate (%)	Throughput	QoS_c	QoS_s
2 (Baseline)	Stochastic	Static	Unlimited	None	–	–	–	–	–	–
3	Stochastic	Static	Unlimited	None	–	–	–	–	–	–
			Unlimited	Modulated	–	–	–	–	–	–
			Limited	None	–	–	–	–	–	–
		Brownian	Unlimited	None	–	–	–	–	–	–
			Unlimited	Modulated	–	–	–	–	–	–
			Limited	None	–	–	–	–	–	–
	Comm Aware	Static	Unlimited	None	–	–	–	–	–	–
			Limited	None	–	–	–	–	–	–
		Brownian	Unlimited	None	–	–	–	–	–	–
			Limited	None	–	–	–	–	–	–
8	Stochastic	Static	Unlimited	None	–	–	–	–	–	–
			Unlimited	Modulated	–	–	–	–	–	–
			Limited	None	–	–	–	–	–	–
		Brownian	Unlimited	None	–	–	–	–	–	–
			Unlimited	Modulated	–	–	–	–	–	–
			Limited	None	–	–	–	–	–	–
	Comm Aware	Static	Unlimited	None	–	–	–	–	–	–
			Limited	None	–	–	–	–	–	–
		Brownian	Unlimited	None	–	–	–	–	–	–
			Limited	None	–	–	–	–	–	–
13	Stochastic	Static	Unlimited	None	–	–	–	–	–	–
			Unlimited	Modulated	–	–	–	–	–	–
			Limited	None	–	–	–	–	–	–
		Brownian	Unlimited	None	–	–	–	–	–	–
			Unlimited	Modulated	–	–	–	–	–	–
			Limited	None	–	–	–	–	–	–
	Comm Aware	Static	Unlimited	None	–	–	–	–	–	–
			Limited	None	–	–	–	–	–	–
		Brownian	Unlimited	None	–	–	–	–	–	–
			Limited	None	–	–	–	–	–	–

swarm's evolutionary performance and perhaps an extension of this work should investigate the impact of exchanging partial genomes and observe its effect on non-elitist GAs. We suspect that increasing the message size beyond the limits of the ESPNOW 250 bytes and having to parse the message will create blottlenecks on the agents processing ability as the buffers would need to grow (to prevent packet losses) and the compute requirements will also increase.

Energy consumption matters. Note that the author of this study is currently employed at Unilode, a company that currently operates the world's largest deployment of IoT gateways dedicated to aviation supply chain logistics. Perhaps one of the key barriers we've observed has been the battery life of sensors, such is its importance in logistics that energy usage will be prioritised over other QoS metrics as asset maintenance is extreamly complex when assets are globally sparse and there is a reliance on third parties to transport these to repair stations. Perhaps this is where we start seeing limitations with the ESPNOW data link which operates $100mA$ compared to BLuetooth's (BLE) $15mA$ during transmission. Figure 18 shows the CPU utilisation of the communication core, showing higher resource usage for broadcast policies. Furthermore, utilisation in Core 1 increases with an increasing number of peers this is because the buffer queue growths as more messages are exchange. Anecdotally, during experiments we observed faster battery drains during flooding experiments. Limiting the number of links via *COMM AWARE* would lead into longer battery savings over time.

Finally on the use of ESPNOW, though out of scope from this study we acknowledge the importance of message encryption for some swarm application. In our implementation one key point that differs from traditional swarms is that each agent needs to be aware of other agents that are part of the same group by directly updating the firmware with a list of MAC addresses, the ESP32 supports up to 20 registered peers for encrypted communication having said that it can expand beyond this limit up to 250 peers without encryption. We do not think this is a single point of failure for the scaling of swarms that use this type of data link, as **??** suggests it is possible to run OTA updates by introducing a peer with an updated firmware update.

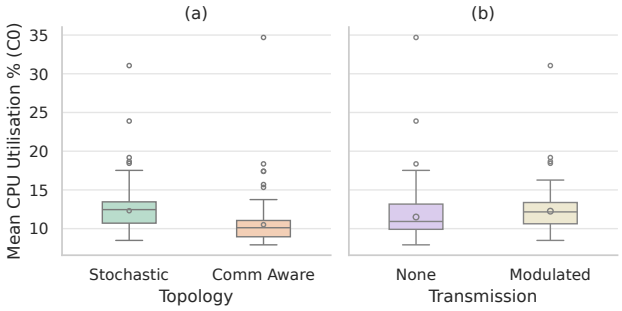

Fig. 18. Core 0 utilisation by communication policy, excluding message limited experiments

A. Limitations

The resolution of the results is affected by drift in the devices RTCs. We measured that between peers there is typically a drift of approximately $\pm 1s$ per device, meaning that message timestamps are not perfectly synchronised. As a result, time-based metrics such as jitter and adaptation rate are subject to a small margin of error. To mitigate this, we grouped the data into two-second buckets when analysing temporal patterns, with the exception of the link formation results shown in Figure 16. The other factor impacting the reliability of the study is the envrionment in which the measurements were taken, although they were conducted in the same room, the experiments were conducted over a period of multiple weeks and we cannot guarantee that variables such as channel interference can be ruled out, having said that we set the ESPNOW data link to operate in channel 6 instead of the default 11 which other access points use.

VIII. Conclusion

A. Future Work

References

[1] Heiko Hamann. *Swarm Robotics: A Formal Approach*. Cham: Springer International Publishing, 2018. ISBN: 978-3-319-74526-8 978-3-319-74528-2. DOI: 10.1007/978-3-319-74528-2. URL: http://link.springer.com/10.1007/978-3-319-74528-2 (visited on 02/15/2024).

[2] Marco Dorigo et al. "Ant algorithms and stigmergy". In: *Future Generation Computer Systems* 16.9 (June 1, 2000). MAG ID: 2151758915, pp. 851–871. DOI: 10.1016/s0167-739x(00)00042-x.

[3] Tan Jian Ding et al. "Advancements and Challenges of Information Integration in Swarm Robotics". In: *2023 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*. 2023 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM). ISSN: 2326-8239. June 2023, pp. 89–95. DOI: 10.1109/CIS-RAM55796.2023.10370011. URL: https://ieeexplore.ieee.org/document/10370011?denied= (visited on 02/25/2024).

[4] Xing An et al. "Multi-Robot Systems and Cooperative Object Transport: Communications, Platforms, and Challenges". In: *IEEE Open Journal of the Computer Society* 4 (2023). Conference Name: IEEE Open Journal of the Computer Society, pp. 23–36. ISSN: 2644-1268. DOI: 10.1109/OJCS.2023.3238324. URL: https://ieeexplore.ieee.org/document/10023955 (visited on 02/26/2024).

[5] Motoaki Hiraga et al. "When Less Is More in Embodied Evolution: Robotic Swarms Have Better Evolvability with Constrained Communication". In: *Journal of Robotics and Mechatronics* 35.4 (Aug. 20, 2023), pp. 988–996. ISSN: 1883-8049, 0915-3942. DOI: 10.20965/jrm.2023.p0988. URL: https://www.fujipress.jp/jrm/rb/robot003500040988 (visited on 07/30/2025).

[6] Sabine Hauert, Jean-Christophe Zufferey, and Dario Floreano. "Evolved swarming without positioning information: an application in aerial communication relay". In: *Autonomous Robots* 26.1 (Jan. 1, 2009), pp. 21–32. ISSN: 1573-7527. DOI: 10.1007/s10514-008-9104-9. URL: https://doi.org/10.1007/s10514-008-9104-9 (visited on 02/15/2024).

[7] Dimitri Perrin and Hiroyuki Ohsaki. "Decentralised Communication in Autonomous Agent Swarms". In: *2012 26th International Conference on Advanced Information Networking and Applications Workshops*. 2012 26th International Conference on Advanced Information Networking and Applications Workshops. Mar. 2012, pp. 1143–1146. DOI: 10.1109/WAINA.2012.201. URL: https://ieeexplore.ieee.org/document/6185403 (visited on 02/19/2024).

[8] Mohamed S. Talamali et al. "When less is more: Robot swarms adapt better to changes with constrained com-

munication". In: *Science Robotics* 6.56 (July 28, 2021). Publisher: American Association for the Advancement of Science, eabf1416. DOI: 10.1126/scirobotics.abf1416. URL: https://www.science.org/doi/10.1126/scirobotics.abf1416 (visited on 02/25/2024).

[9] Till Aust et al. "The Hidden Benefits of Limited Communication and Slow Sensing in Collective Monitoring of Dynamic Environments". In: *Swarm Intelligence*. Ed. by Marco Dorigo et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 234–247. ISBN: 978-3-031-20176-9. DOI: 10.1007/978-3-031-20176-9_19.

[10] Konstantinos I. Tsianos and Michael G. Rabbat. *The Impact of Communication Delays on Distributed Consensus Algorithms*. July 24, 2012. DOI: 10.48550/arXiv.1207.5839. arXiv: 1207.5839[cs]. URL: http://arxiv.org/abs/1207.5839 (visited on 08/16/2025).

[11] Stefan M. Trenkwalder et al. "SwarmCom: an infra-red-based mobile ad-hoc network for severely constrained robots". In: *Autonomous Robots* 44.1 (Jan. 1, 2020), pp. 93–114. ISSN: 1573-7527. DOI: 10.1007/s10514-019-09873-0. URL: https://doi.org/10.1007/s10514-019-09873-0 (visited on 12/11/2023).

[12] Mahmoud Almostafa Rabbah et al. "Real Time Distributed and Decentralized Peer-to-Peer Protocol for Swarm Robots". In: *International Journal of Advanced Computer Science and Applications (IJACSA)* 12.11 (Jan. 30, 2021). Number: 11 Publisher: The Science and Information (SAI) Organization Limited. ISSN: 2156-5570. DOI: 10.14569/IJACSA.2021.0121131. URL: https://thesai.org/Publications/ViewPaper?Volume=12&Issue=11&Code=IJACSA&SerialNo=31 (visited on 12/10/2023).

[13] Vivek Shankar Varadharajan et al. "Over-the-Air Updates for Robotic Swarms". In: *IEEE Software* 35.2 (Mar. 2018), pp. 44–50. ISSN: 0740-7459, 1937-4194. DOI: 10.1109/MS.2018.111095718. URL: https://ieeexplore.ieee.org/document/8255775/ (visited on 08/03/2025).

[14] M. Ruciński, D. Izzo, and F. Biscani. "On the impact of the migration topology on the Island Model". In: *Parallel Computing*. Parallel Architectures and Bioinspired Algorithms 36.10 (Oct. 1, 2010), pp. 555–571. ISSN: 0167-8191. DOI: 10.1016/j.parco.2010.04.002. URL: https://www.sciencedirect.com/science/article/pii/S0167819110000487 (visited on 12/10/2023).

[15] ESP-BOARDS. *ESP-IDF (IoT Development Framework) vs Arduino Core in 2023*. URL: https://www.espboards.dev/blog/esp-idf-vs-arduino-core/ (visited on 06/03/2024).