Mälardalen University
M.Sc.Eng. Dependable Aerospace Systems
Västerås, Sweden

Project Course in Dependable Systems
22.5 credits

System Design Description

Responsible
Andrea Haglund
ahd20002@student.mdu.se

Contributors

Claire Namatovu
cnu21001@student.mdu.se

Emily Zainali
ezi21001@student.mdu.se

Esaias Målqvist
emt21001@student.mdu.se

Yonatan Michael Beyene
yme21001@student.mdu.se

Examiner: Luciana Provenzano

December 2, 2025

DOCUMENT APPROVAL

Name	Role	Version	Date	Signature
Esaias Målqvist	Safety Manager	1.0	2025-11-16	
Yonatan Michael Beyene	Q&C Manager	1.0	2025-11-16	

DOCUMENT CHANGE RECORD

Version	Date	Reason for Change	Pages / Sections Affected
0.1	2025-11-12	Version for internal review	
0.2	2025-11-13	Version for review	
1.0	2025-11-16	Version for public release	All
1.1	2025-11-30	Version for review	All
1.2	2025-12-04	Version for review	All
2.0	2025-12-07	Version for public release	All

Contents

List of Figures

1 Introduction

The Intelligent Replanning Drone Swarm (IRDS) protocol module is a decentralised coordination and replanning system for Unmanned Aerial Vehicle (UAV) swarms in Search and Rescue (SAR) missions. Each UAV runs an instance of the protocol, allowing the swarm to continue operating when individual agents degrade or fail, without relying on a central controller. This document gives a conceptual design of the protocol module: it describes the context and goals of the system, the architectural structure, the Rumour Mill consensus mechanism, the Sector Allocation Market, the Agent Task & Role Allocation layer, and the safety and fault-tolerance model.

1.1 Purpose

This document describes the conceptual design of the IRDS protocol module and serves as a reusable technical specification for future work. It explains:

- How the system is architecturally structured.
- How information is exchanged and agreed upon using the Rumour Mill.
- How search work is distributed using the Sector Allocation Market.
- How individual agents choose roles and behaviours using Task & Role Allocation.
- How safety and fault tolerance are achieved at the swarm level.

1.2 Intended Audience

This document is intended for:

- Project groups continuing the development of the IRDS.
- Researchers studying decentralised UAV coordination.
- Designers developing UAV swarm-level logic, interfaces, or simulations.
- Supervisors and examiners evaluating design decisions against course requirements.

The reader is expected to have basic familiarity with UAV systems, distributed coordination concepts, and general principles of software and system architecture.

1.3 System Overview

The IRDS protocol module is a decentralised coordination and replanning system for UAV swarms performing Search and Rescue (SAR) missions. Each UAV hosts an instance of the protocol module, which enables the swarm to assign work, react to degraded agent health, and maintain mission continuity without relying on a single central controller. The protocol interacts with a SwarmInterface component that exposes high-level commands from the Search Manager (such as mission start, search area, and hot regions) and reports back swarm status and key events.

Conceptually, the protocol module is organised into three functional layers. The Rumour Mill layer provides decentralised consensus on mission state and significant events. The Sector Allocation Market layer uses this shared state to distribute search sectors and rebalance work when agents degrade or fail. The Agent Task & Role Allocation layer translates sector ownership and agent health into concrete roles and tasks that each UAV executes through its onboard flight systems. This document specifies the behaviour and architecture of the protocol module at this conceptual level, while treating low-level flight control, detailed sensing, and physical communication mechanisms as external to the system.

1.4 Novel Contributions

The IRDS protocol module provides the following main contributions:

- A decentralised protocol architecture for UAV swarms that maintains mission continuity without a single point of failure.
- A rumour-based consensus mechanism tailored to SAR missions, using neighbourhoods and simple message types to achieve eventual consistency.
- A sector allocation market that distributes and rebalances search work based on agent health, workload, and sector value.
- An agent-level task and role model that links swarm-level decisions to concrete UAV behaviour.
- A safety and fault-tolerance model that combines local flight-system safeguards with swarm-level reasoning about degraded or faulty agents.

1.5 Document Structure

The remainder of this document is structured as follows:

- Section 2 - Goals & Requirements Overview
  Summarises the project goals and layered requirements (swarm, protocol, and sub-modules) and shows how they guide the design.
- Section 3 - System Context
  Describes the SAR environment, external actors (Search Manager, SwarmInterface), system boundaries, assumptions, and constraints.
- Section 4 - System Overview
  Summarises the main functional areas (Rumour Mill, Sector Allocation Market, Task & Role allocation) and presents high-level swarm workflows and scenarios.
- Section 5 - System Architecture
  Describes the structural decomposition of the IRDS system, including agent architecture, neighbourhood structure, communication model, and data structures.
- Section 6 - Rumour Mill Protocol (Consensus Layer)
  Defines the rumour message types, lifecycle, trigger events, and consensus thresholds. This section formalises what is meant by ”Rumour Mill”.
- Section 7 - Sector Allocation Market (Workload Layer)
  Explains how sectors are valued, how agents budget and bid for sectors, and how sector ownership is updated via the Rumour Mill.
- Section 8 - Agent Task & Role Allocation (Behaviour Layer)
  Describes how agents choose roles (search, relay, return, etc.) based on health status, sector ownership, and commands, and how this is implemented as a state machine.
- section 9 - Safety & Reliability
  Presents fault model, pulse-based failure detection, Byzantine fault tolerance assumptions, and safety-related behaviour.
- Section 10 - Future Work
  Outlines recommended directions for further development and research, such as implementation, security, and large-scale validation.
- Appendix
  Provides supporting details such as hardware diagrams.

2 Goals & Requirements Overview

This section summarises the goals and requirements that drive the IRDS protocol module. Detailed specifications are kept in four artefacts (goals, drone_swarm_requirements, system_requirements, subsystem_requirements) [2]; only the most relevant elements are repeated here. The aim is to show what the system must achieve at the project, swarm, protocol, and sub-module levels, and how these requirements shape the architectural and behavioural design choices in the rest of this document.

2.1 Goals & Requirements Overview

The goals and requirements are maintained in the following artefacts:

- goals - project goals derived from stakeholder needs and the KAOS method.
- drone_swarm_requirements - swarm-level behavioural and safety requirements.
- system_requirements - requirements allocated to the IRDS protocol module as a system.
- subsystem_requirements - requirements allocated to the internal sub-modules of the protocol.

2.2 Project Goals

The project goals in goals capture what the IRDS system of systems is intended to achieve at a high level. Key examples include:

- G-01 - Assign an appropriate task to each agent at the start of the mission.
- G-02 - When an agent fails, automatically reassign its tasks to other functional agents when resources permit.
- G-03 - Maintain correct swarm behaviour despite failure or inconsistent output from single agents.
- G-04 / G-05 - Reach collective decisions for tasks that require coordination and ensure that the search area is distributed among agents.
- G-08 / G-09 - Maintain mission continuity and recover from agent failures under the chosen protocol.
- G-11 / SG-01 - Allow agents to make individual decisions when appropriate while maintaining staff safety in all operational phases.

The IRDS protocol module contributes to these goals primarily through its three main functional areas: Rumour Mill (decentralised consensus on mission state and events), Sector Allocation Market (distribution of search sectors based on mission priorities and agent state), and Task & Role Allocation (mapping sector ownership and health status into concrete agent behaviour).

2.2.1 Requirements Structure

The requirements are organised in three main layers that correspond to different responsibility levels in the architecture:

- Swarm-level requirements (drone_swarm_requirements.csv)
  These requirements (e.g. SW-01-SW-07, SR-02-SR-07) describe how the swarm as a whole shall behave. Examples include:
  - Assigning tasks to all agents when a mission starts (SW-01).
  - Reallocating work when agents degrade or fail (SW-02, SW-03).
  - Employing a consensus mechanism that maintains correct behaviour even when at least one agent provides incorrect data (SW-04, SW-05).
  - Coordinating agents so that they cover different parts of the Search Area (SW-06).
  - Allowing agents to make independent local decisions when immediate action is required (SW-07).
  - Maintaining safety distances and buffer zones around agents and staff (e.g. SR-02-SR-07).
- System-level protocol requirements (system_requirements.csv)
  These requirements (PM-01-PM-xx) define what the IRDS protocol module as a system shall do, for example:
  - Define procedures to keep all agents aware of the prevailing mission state (PM-01).
  - Define procedures for sharing the mission plan and assigning tasks to agents based on the plan and their capabilities (PM-02, PM-03, PM-06).

    - Define how agents communicate their capabilities and the message formats used for this communication (PM-04, PM-05).
- Subsystem-level protocol requirements (subsystem_requirements.csv)
  These requirements are allocated to internal sub-modules of the protocol (Consensus, Information, Messaging, State Instruction, etc.). Examples include:
    - CON-01 / CON-02 - the Consensus sub-module shall specify how agents confirm agreement on the prevailing mission state and how they reach a common understanding of the mission plan.
    - INF-01-INF-03 - the Information sub-module shall define the content and structure of mission state data and the mission plan, and how agents retrieve this information.
    - MSG-01 / MSG-02 - the Messaging sub-module shall define communication rules for transmitting mission state and mission plan data between agents.
    - SIS-01 - the State Instruction sub-module shall define how instructions are derived from the current mission state and issued to agents.

In the architectural view adopted in this document, the IRDS protocol module is primarily responsible for satisfying the system-level and subsystem-level requirements, and for contributing to the swarm-level requirements that concern coordination, consensus, and task allocation. Low-level flight behaviour, obstacle avoidance, and detailed motion control requirements are delegated to the UAV flight systems (e.g. FCU, ODU, CAU), which are treated as external components.

2.2.2 Traceability

A detailed mapping from goals and requirements to design elements is maintained in the project's requirements artefacts and supporting traceability tables. At a high level:

- Swarm-level requirements such as SW-01-SW-07 and SR-02-SR-07 are addressed by the swarm context (Section 2), the system overview (Section 3), the Sector Allocation Market (Section 6), and the Agent Task & Role Allocation layer (Section 7)
- System-level protocol requirements (PM-xx) are mainly realised by the architecture described in Section 4, the Rumour Mill protocol in Section 5, and the Sector Allocation Market and Task & Role Allocation layers in Sections 6 and 7.
- Subsystem-level requirements (CON-xx, INF-xx, MSG-xx, SIS-xx) correspond to the internal sub-modules of the protocol module described in Sections 4 and 5.

The full traceability matrix is maintained outside this document and can be consulted for detailed one-to-one mappings between individual requirements and specific design artefacts.

Table 1 provides a small sample of the full requirements-design traceability matrix, illustrating how selected goals and requirements are realised by specific architectural elements and sections in this document.

Requirement ID	**Short description**	**Realised by design elements / sections**
SW-01	All agents shall be assigned a task at mission start.	3 System Overview; 4.3 Agent Architecture; 6 Sector Allocation Market; 7 Agent Task & Role Allocation
SW-07	Agents shall be able to take local decisions when necessary.	3.2 UAV Platform and Flight Systems; 4.2 External Components; 8.3 Safety and Local Emergency Behaviour
SR-03	The swarm shall maintain separation and safety buffer zones.	2.3 Swarm Context; 3.2 UAV Platform and Flight Systems; 8.3 Pulse Safety Mechanism and Buffer Zones
PM-01	Protocol shall keep all agents aware of mission state.	4 Protocol Module Architecture; 5 Rumour Mill Protocol; 5.4 Event Dissemination and Convergence
PM-03	Protocol shall support task assignment based on capabilities.	4.3 Agent Architecture; 6 Sector Allocation Market; 7 Task & Role Allocation
CON-01	Consensus sub-module shall define agreement on mission state.	4.4 Consensus Sub-module; 5.2 Rumour Model; 8.5 Formal Fault Tolerance Model
MSG-02	Messaging shall provide reliable delivery of critical updates.	4.6 Communication Model; 5.3 Message Types; 5.5 Reliability and Acknowledgement Strategy

Table 1: Sample requirements-design traceability.

3 System Context

This section describes the operational context, external actors, system boundaries, assumptions, and constraints of the IRDS system.

3.1 Operational Environment

The IRDS system is intended for SAR missions in which a swarm of UAVs is deployed to locate a missing Subject within a geographical area. Typical characteristics of this environment include:

- Uncertain and dynamic environments, including forests, mountain regions, or water.
- Limited or unreliable communication caused by obstacles, distance, or environmental interference.
- Decentralisation requirements, since persistent connection to a ground station or ”mother drone” cannot be guaranteed.
- Safety and time constraints, where prompt detection of the health of degraded agents is essential to maintain the integrity of the mission.

The above conditions motivate a decentralised, fail-operational swarm: The system must be able to continue a mission even if some agents become degraded, fail, or temporarily lose communication.

3.2 Actors & External Entities

The protocol module interacts with several external entities, and by understanding these actors, it clarifies the boundaries between the protocol module, the UAV hardware, and operational human control.

3.2.1 Search Manager (Human Operator)

The Search Manager is responsible for the strategic oversight of the mission.

Responsibilities include:

- Defining and adjusting the Search Area.
- Marking Hot Regions on the Search Area.
- Initialisation of swarm deployment.
- Issuing high-level operational commands (start search, pause, adjust Search Area, abort mission).
- Receive status updates through the SwarmInterface.
- Preparing interventions such as battery replacements when an agent returns home.

The Search Manager does not directly coordinate agent-level behaviour. Instead, all commands are processed through the protocol at swarm level.

3.2.2 SwarmInterface (External System)

The SwarmInterface serves as the communication layer between the Search Manager and the swarm. The interface may be implemented as software running on a laptop, tablet, or portable ground station.

Responsibilities include:

- Allowing a Search Manager to draw Hot Regions on a map.
- Dividing a Search Area into rectangular sectors.
- Assigning probability values to sectors based on marked Hot Regions.
- Sending high-level commands to the swarm (e.g. mission start, pause, abort mission).
- Receiving and displaying aggregated swarm status such as:
  - Degraded health events,
  - agents returning for battery replacement,
  - Subject found,
  - emergency landings,
  - membership status of agents.
- Ensure that messages follow the protocol module’s formatting rules, such as message IDs and data structure.

The SwarmInterface is included as a conceptual component in this iteration and the IRDS protocol module is designed to integrate with it. The SwarmInterface is intended to be developed in future work.

3.2.3 Agents (UAVs)

Agents are the autonomous UAVs that execute the protocol module.

Responsibilities include:

- Participate in consensus through the Rumour Mill.
- Perform health-dependent behaviour.
- Maintain knowledge of its neighbourhood.
- Broadcast and receive pulses (heartbeat messages).
- Perform tasks or secondary roles according to their state.
- Maintain an internal AgentInfo list describing known agents.
- Maintain a SectorInfo list describing known sectors.
- Form neighbourhoods with geographically closest agents.
- Report its health status to the swarm.
- Respond to Search Manager commands through the protocol module.

Agents are individually responsible for maintaining internal consistency and are cooperatively responsible for maintaining swarm-wide agreement. Agents can join, rejoin, or leave the swarm at any time.

3.2.4 Fault Management Unit

Each agent contains a Fault Management Unit (FMU) that monitors the health status of hardware and software.

Responsibilities include:

- Detect degraded health states.
- Publish health information to the agent.
- Trigger the creation of Origin rumours.
- Initiate fault isolation actions.

The protocol module relies on FMU health status outputs and does not modify the FMU itself.

3.2.5 Environment

The environment influences communication, agent behaviour, and mission performance. Relevant environmental factors include:

- Radio communication range and interference.
- Physical obstacles (terrain, forest canopy, urban structures).
- GPS conditions.
- Wind, rain, or weather effects on flight stability.

The protocol module assumes that while communication may be unreliable, rumour propagation is eventually possible.

3.3 System Boundaries

The protocol module defines behaviour at the swarm coordination level. The following responsibilities lie within the scope of the protocol module:

- Consensus formation (rumour lifecycle) for event interpretation.
- Sector allocation and bidding.
- Broadcasting health status
- Task allocation and reallocation
- Neighbourhood formation and recalculation.
- Transitions in agent behaviour state.
- Propagation of Search Manager commands.
- Track dynamic information about agents and sectors.
- Pulse monitoring.

The following responsibilities fall outside the scope of the protocol:

- Low-level flight control (managed by the FCU).
- Navigation and GPS fusion.
- Obstacle detection and avoidance (ODU / CAU).
- Motor control.
- Physical take-off, landing, and stability control.
- Fault detection logic inside the FMU.
- Cryptographic security or authentication mechanisms.

3.4 Assumptions

The protocol module design relies on the following assumptions:

- Swarm size: 4–255 agents
  - A minimum of four agents is required to tolerate one Byzantine fault per neighbourhood.
  - The maximum is capped by 8-bit (one byte) AgentID space.
- Each agent knows its own ID and can uniquely identify others.
- AgentID = 0 reserved for Search Manager.
- Neighbourhood definition:
  - Each agent forms a neighbourhood using the three geographically closest agents, determined by initial sector assignment.
  - Neighbourhoods overlap.
- Neighbourhood recalculation:
  - If an agent fails, leaves, or is kicked out of the swarm, all affected neighbourhoods must be recomputed to restore a 1 + 3 structure.
- Communication:
  - Agents exchange messages over unreliable wireless networks.
  - Message delays and drops are possible, but connectivity is eventually restored.
- Search Area:
  - The overall Search Area is defined by the Search Manager via the SwarmInterface.
  - The SwarmInterface divides the Search Area into sectors.
  - Sectors are rectangular (defined by NE/NW/SE/SW points).
  - The SwarmInterface assigns a value to each sector based on the Hot Regions marked by the Search Manager.
  - That value is used as part of the sector cost in the Sector Allocation Market.
- FMU health statuses are trustworthy, meaning that the FMU is assumed to operate correctly and FMU faults are not considered.
- Agent dynamics:
  - Agents are airborne during operation except during emergency landings or returns.
  - Agents may join, rejoin, or leave the swarm at any time.
- Security:
  - No encryption, authentication, or integrity protection is implemented in the current design.
  - All agents are assumed to be authorised participants.
  - Security is treated as future work.

3.5 Constraints

The following constraints affect the design and behaviour of the protocol module:

- Energy constraints:
  - Battery level influences the budget for sector bidding and mobility.
- Computational limitations:
  - Agents may run on microcontrollers with limited memory and CPU.
- Bandwidth limitations:
  - Message sizes must remain small; rumour structures must be lightweight.
- Latency:
  - Rumour consensus must function under delayed or intermittent communication.
- Real-time constraints:
  - The protocol module must not delay or interfere with safety-critical flight control routines.
- Regulatory constraints:
  - UAV operations must comply with applicable aviation rules and safety procedures.

3.6 Dynamic Deployment

The protocol module supports dynamic and staggered agent deployment to support real-world scenarios that involve battery cycles, repairs, and communication degradation.
Dynamic deployment means that agents may:

- Launch at different times.
- Join the swarm after the initial deployment.
- Disconnect temporarily due to communication loss.
- Rejoin after recovery.

A **returning** agent A must perform a registration procedure:

1) Announce its presence through an Origin rumour to agents that were considered neighbours by A.
2) Receive Source rumours from those neighbours concerning if A has received membership.
3) Integrate into neighbourhood structures.
4) Receive information from neighbours to update its AgentInfo and SectorInfo lists.
5) Request sector allocation.
6) Synchronise with the swarm state.

A **new** agent B must perform a registration procedure:

1) Before deployment, receive information from the SwarmInterface to populate its AgentInfo and SectorInfo lists, and receive information on possible neighbours.
2) Announce its presence through an Origin rumour to agents the SwarmInterface assigned as possible neighbours.
3) Receive Source rumours from those possible neighbours concerning if B has received membership.
4) Integrate into neighbourhood structures.
5) Receive information from neighbours to update its AgentInfo and SectorInfo lists.
6) Request sector allocation.
7) Synchronise with the swarm state.

3.7 System Context Diagram

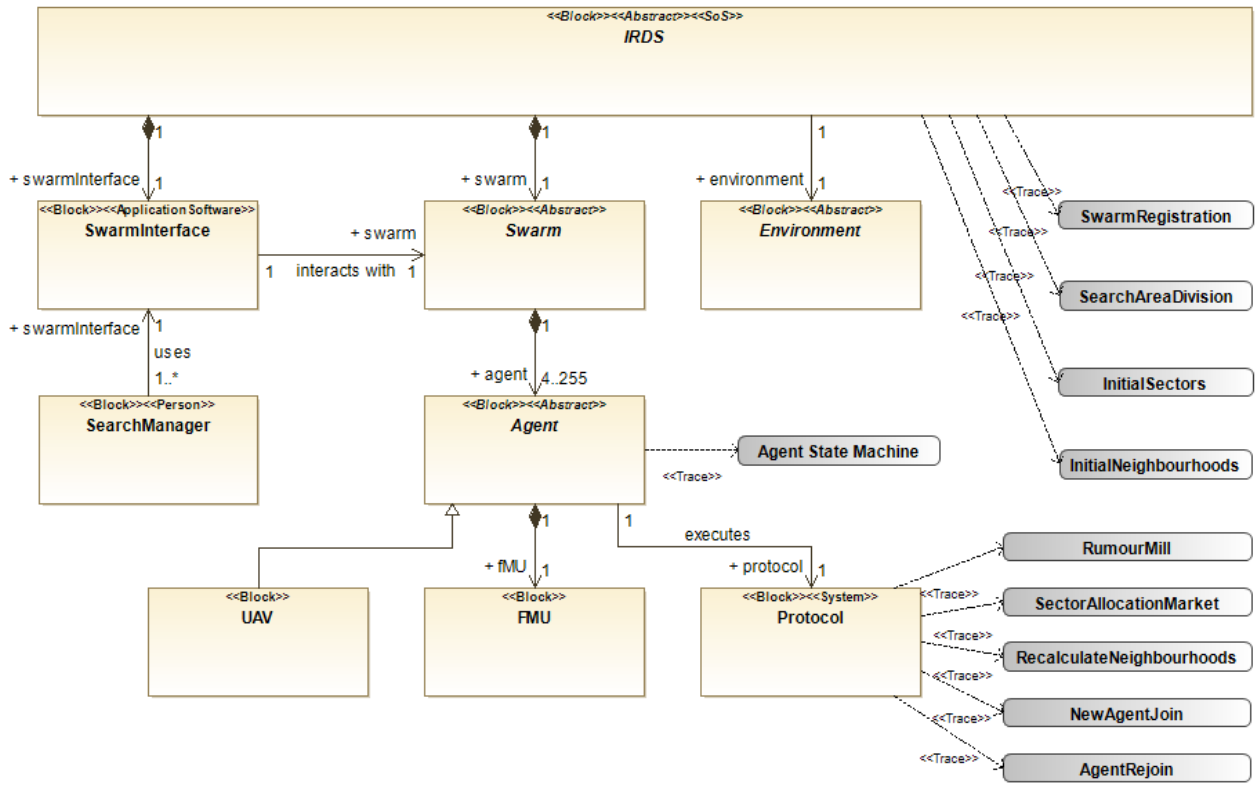

Figure 1: *(SD-01)* Block Definition Diagram (BDD) of which systems the protocol is related to. One swarm contains four to 255 agents, and one agent contains one FMU. UAVs inherit from agent, meaning that UAVs must contain FMUs. Each agent executes the protocol. The protocol carries out a set of behaviours. Search Managers use a Swarm Interface to interact with a swarm.

4 System Overview

This section provides a high-level behavioural overview of the IRDS protocol module and how it coordinates the swarm during a mission. It explains the design philosophy behind the protocol, introduces the three core functional layers (Rumour Mill, Sector Allocation Market, and Agent Task & Role Allocation), and describes the overall operational workflow at an intuitive level. Detailed structural decomposition and low-level protocol mechanics are intentionally deferred to Sections 5–8; the focus here is on how the main concepts fit together to produce the desired swarm behaviour.

4.1 Design Philosophy

The protocol module is designed around three guiding principles: Decentralisation, collective adaptation, and fault tolerance.

4.1.1 Decentralisation

The protocol module is designed to operate without any central controller or privileged ”mother drone”. Instead, every agent executes the same protocol module, makes decisions based on its own state and received messages, and participates equally in consensus and allocation. Those factors remove single points of failure: If any specific agent fails, the rest of the swarm continues to operate and make decisions.

4.1.2 Collective Adaptation

The swarm must adapt when conditions change. Examples include:

- An agent's health degrading.
- An agent running low on battery.
- Communication loss.
- Discovery of the Subject.
- Changes in the Search Area commanded by the Search Manager.

Rather than adapting locally in an ad-hoc way, the protocol module coordinates reactions collectively; Events are shared via rumours, decisions are agreed via the Rumour Mill, sectors are reallocated through the Sector Allocation Market, and agents update their roles accordingly.

4.1.3 Fault Tolerance

Using neighbourhoods, the swarm can tolerate Byzantine faults within each neighbourhood to ensure that no single faulty agent can corrupt group decisions or destabilise the mission.

The protocol module uses the following to tolerate Byzantine faults:

- Neighbourhoods (each agent + its three closest neighbours) as basic rumour propagation groups.
- A rumour lifecycle that
  - requires three independent Source rumours before an agent A interprets the event from agent Y through internal voting, and
  - after event interpretation, agent A possibly broadcasts an Opinion, where
  - agent Y (the originator) receives Opinions up to a certain consensus threshold α before making a final decision.

This design localises the impact of faulty agents and supports safe mission continuation despite individual failures.

4.2 Core Functional Layers

The protocol can be decomposed into several core functional layers:

4.2.1 Rumour Mill (Consensus Layer)

The Rumour Mill is the decentralised consensus mechanism used by the swarm to reach shared agreement about mission events (e.g. “Agent A is degraded”, “Sector S is completed”) and the decisions that follow from them (e.g. “Agent A should become relay”, “Agent A owns sector S”). Each agent broadcasts and interprets rumours about events, and converges to decisions based on multiple, independent confirmations from its neighbourhood. The full rumour types, message formats, and lifecycle are described in Section 6.

4.2.2 Sector Allocation Market (Workload Layer)

The Sector Allocation Market decides which agent is responsible for which sectors. It uses information about sector value and location, agent health and workload, and current sector ownership to distribute work and rebalance it when agents degrade or fail. The market operates on top of the Rumour Mill, so all changes in sector ownership are agreed via rumour-based consensus. The detailed market model and bidding process are described in Section 7.

4.2.3 Agent Task & Role Allocation (Behaviour Layer)

The Agent Task & Role Allocation layer determines what each agent actually does at any given time. It maps sector ownership (from the Sector Allocation Market), health status (from the FMU), and Search Manager commands (via the SwarmInterface) into roles and tasks such as searcher, relay, return, or failed. These roles drive the agent’s state machine and resulting flight behaviour. The state machine and role logic are described in detail in Section 8.

4.3 High-Level Operational Workflow

At a high level, the protocol module operates according to the following recurring loop:

1) Event occurs, such as:
   - FMU detects health degradation.
   - A sector is completed.
   - A new agent joins.
   - A Search Manager command is issued.
   - A pulse is missed.
   - The Subject is found.
2) Rumour Mill processes the event:
   - The event becomes an Origin rumour.
   - Neighbours create Source rumours interpreting the event.
   - Agents that have received three Source rumours interpret the event through internal voting.
   - Agents possibly create Opinion rumours after voting.
   - The originator collects Opinions until the consensus threshold α is reached and decides.
   - The decision is announced through a new Origin rumour.
3) Sector Allocation Market updates responsibilities:
   - Degraded agents release sectors.
   - Healthy agents bid on available sectors.
   - Ownership changes are confirmed through the Rumour Mill.
4) Agent Task & Role Allocation updates behaviours:
   - Based on health, sectors, and decisions, each agent updates its state and role.
   - Some agents continue searching, some become relays, some return to base, etc.
5) Swarm continues the mission:
   - Updated roles and sector allocations produce an adjusted search pattern.
   - The cycle repeats as new events occur.

4.4 Representative Operational Scenarios

The following scenarios illustrate how the layers work together.

4.4.1 Swarm Registration at Initial Start-Up

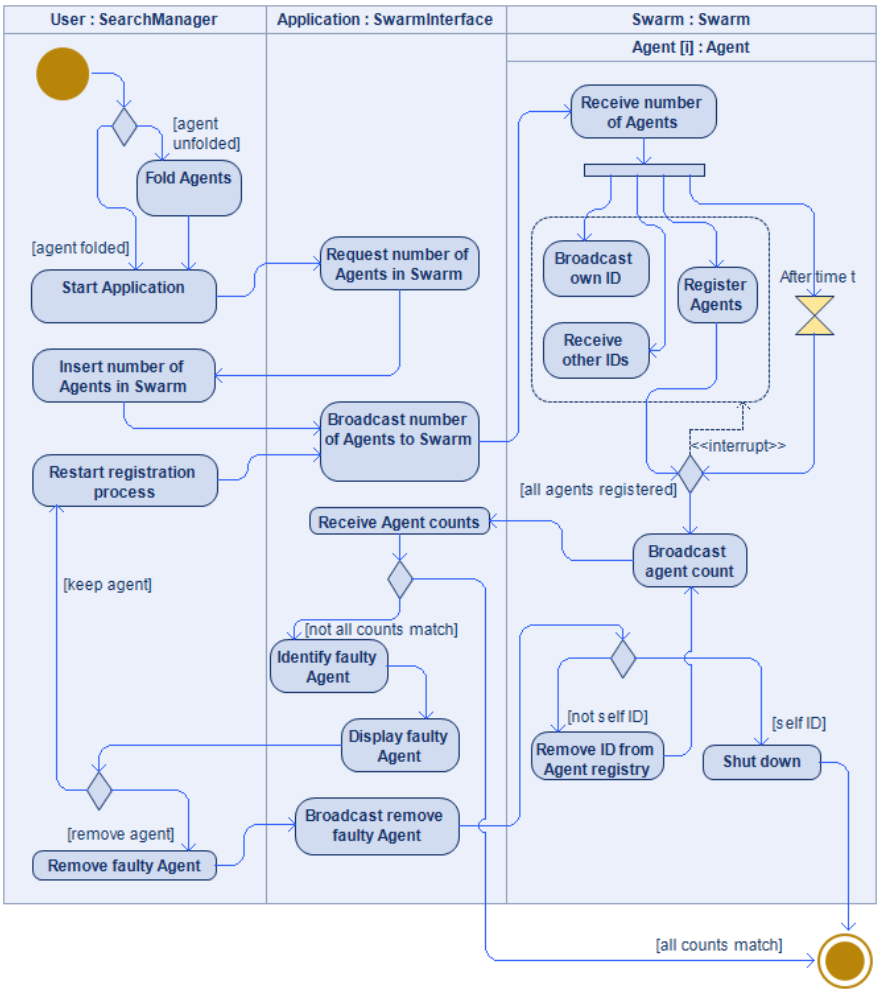

Figure 2: *(SD-03)* Activity Diagram of swarm registration at initial start-up. This activity diagram does not cover agents joining the swarm after mission start.

4.4.2 Search Area Division

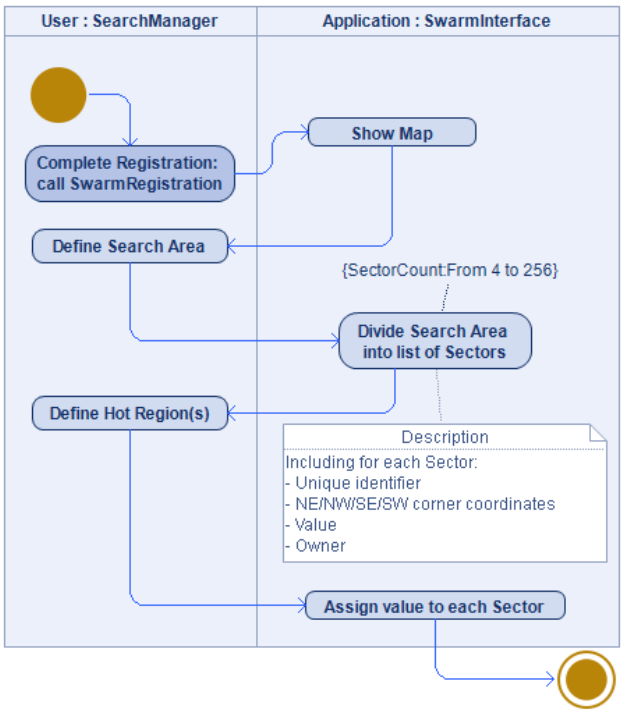

Figure 3: *(SD-04)* Activity Diagram of dividing a Search Area into sectors.

4.4.3 Initial Sectors Assigned

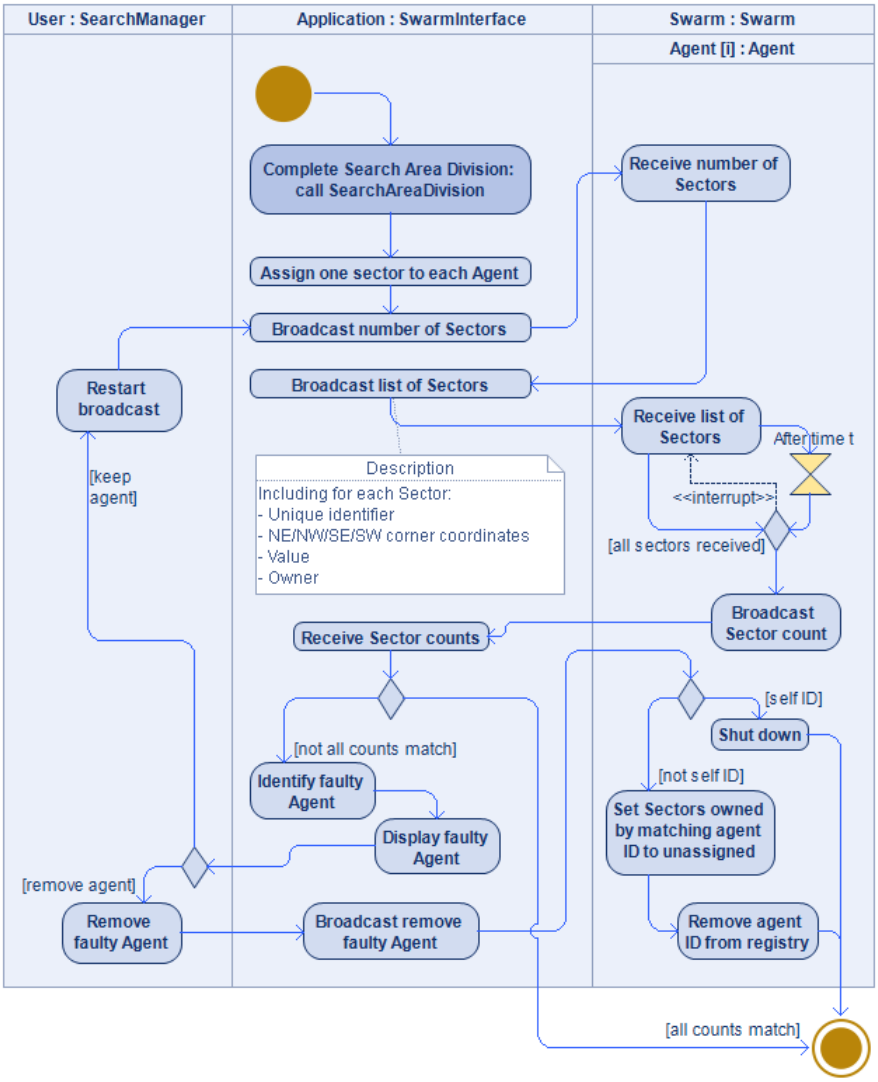

Figure 4: *(SD-20)* Activity Diagram of assigning initial sectors.

4.4.4 Agent Join After Mission Start

Activity diagram showing the process of agents joining after mission start

4.4.5 Agent Rejoin

Activity diagram showing the process of agents re-joining after mission start

4.4.6 Agent Health Degrades

1) FMU on Agent A reports degraded health (e.g. sensor fault).
2) A releases its sectors and broadcasts an Origin: ”I am degraded (bad_sensor)”.
3) Neighbours interpret the Origin rumour and broadcast Source rumours about the degraded state.
4) Agents that have received three Source rumours interpret the event through internal voting.
5) Agents broadcast Opinion rumours; A receives enough Opinions and decides to switch to a secondary role.
6) A broadcasts a new Origin announcing its decision.
7) Its sectors re-enter the Sector Allocation Market; healthy agents bid and take over.
8) A moves to a secondary role, such as a communication relay.

4.4.7 Sector Completed

1) Agent A finishes searching sector S.
2) A creates an Origin rumour: ”I (A) have completed sector S.”
3) Neighbours interpret the Origin rumour and broadcast Source rumours confirming that S is completed.
4) Agents that have received three Source rumours interpret the event through internal voting.
5) Agents update their SectorInfo.
6) The budget of A increases by the cost of S, and A bids for a new sector.

4.4.8 Subject Found and Confirmed

1) Agent A detects the Subject in sector S.
2) A broadcasts an Origin rumour: ”Subject found at location L.”
3) Neighbours go to L to confirm Subject existence at L.
4) Neighbours broadcast Source rumours based on their own observations.
5) Agents that have received three Source rumours interpret the event through internal voting.
6) A decides from Source rumours that the Subject is confirmed or that A should move on to other sectors.
7) If Subject is confirmed found through internal voting, all agents eventually shift to return state.

4.4.9 Agent Returns for Battery Change

1) FMU on Agent A reports low battery.
2) A releases its sectors and broadcasts an Origin rumour: ”Returning for battery change”.
3) Neighbours interpret the Origin rumour and broadcast Source rumours: ”A no longer member of swarm”.
4) SwarmInterface displays to the Search Manager: ”Agent A returning – prepare battery replacement.”
5) Agents that have received three Source rumours interpret the event through internal voting and update their AgentInfo and SectorInfo lists.
6) A comes back and requests membership.

5 System Architecture

This section describes the structural architecture of the IRDS protocol module and its surroundings. It identifies the main components, their internal sub-modules, and the interfaces through which they interact with one another and with external systems such as the SwarmInterface and the UAV flight stack. The section also introduces the communication model and information model used by the protocol. Together, these views provide the static foundation on which the dynamic behaviours in Sections 6–8 are defined.

5.1 Architecture Context

At the top level, the IRDS architecture consists of:

- A Swarm of 4–255 agents, each running the IRDS protocol module.
- A SwarmInterface that relays high-level inputs from the Search Manager and displays swarm status.
- An Environment that influences communication and flight.

The System Context Diagram (fig. 1) shows this context:

- The Search Manager interacts only with the SwarmInterface.
- The SwarmInterface exchanges messages with the swarm as a whole.
- The swarm is modelled as a composition of agents.
- Each agent contains an FMU and the IRDS protocol module.

Therefore, the protocol module is a swarm-level coordination layer that sits on top of the UAVs' flight and navigation systems.

5.2 Decentralised Swarm Structure

5.2.1 Swarm & Neighbourhood Composition

The swarm is decentralised:

- There is no central controller or privileged leader.
- All agents run the same protocol module.
- Each agent maintains its own local view of the swarm (AgentInfo, SectorInfo, neighbourhoods).
- Swarm-wide behaviour emerges from repeated local interactions and the Rumour Mill.

If an individual agent fails or leaves, the remaining agents:

- Changes its membership status in their AgentInfo lists.
- Recompute affected neighbourhoods.
- Release and reallocate its sectors via the Sector Allocation Market.
- Continue the mission without requiring any central reconfiguration.

Each agent maintains a neighbourhood of exactly four agents in total consisting of itself and its three geographically closest agents (where "closest" is defined using the centres of each agent's initial sectors).

5.2.2 Neighbourhood Formation at Mission Start

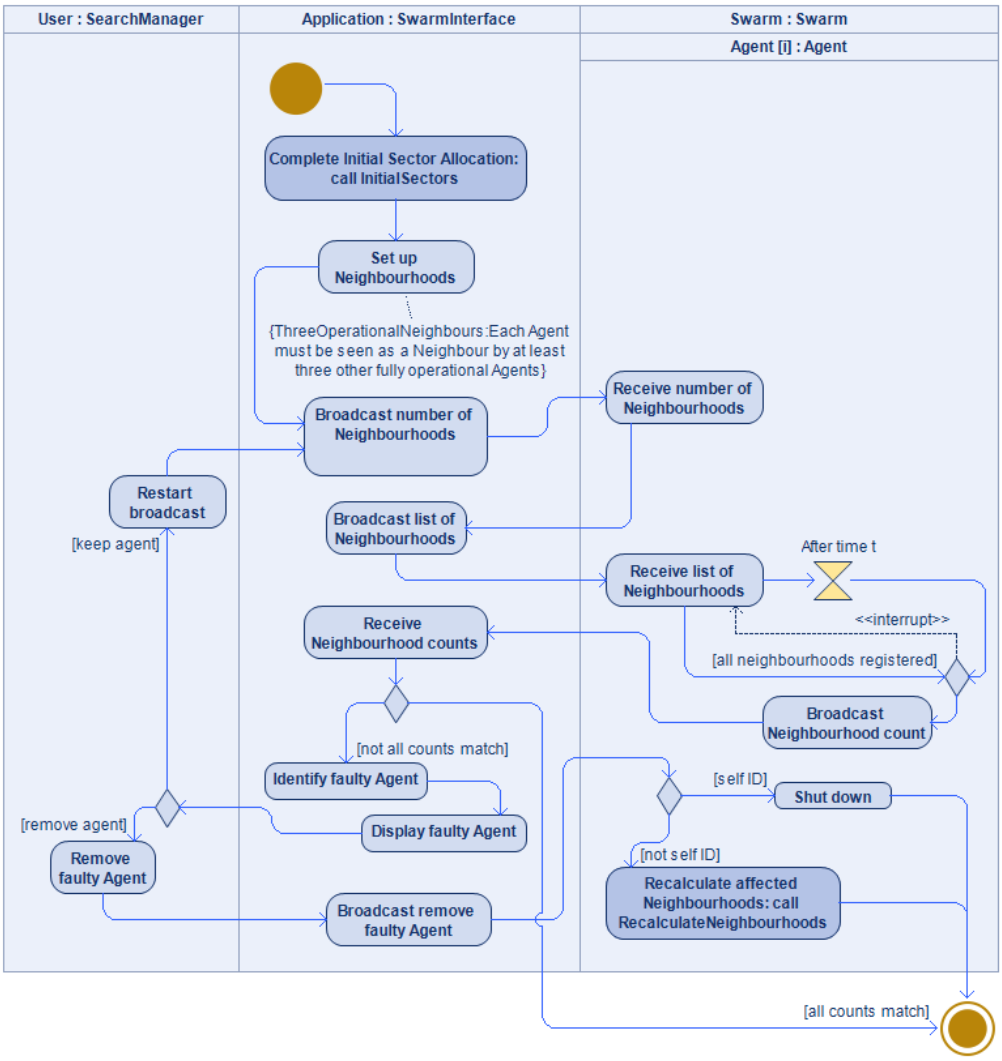

Figure 5: *(SD-24)* Activity Diagram showing the process of neighbourhood formation after initial sectors have been assigned.

5.2.3 Neighbourhood updates after join/re-join

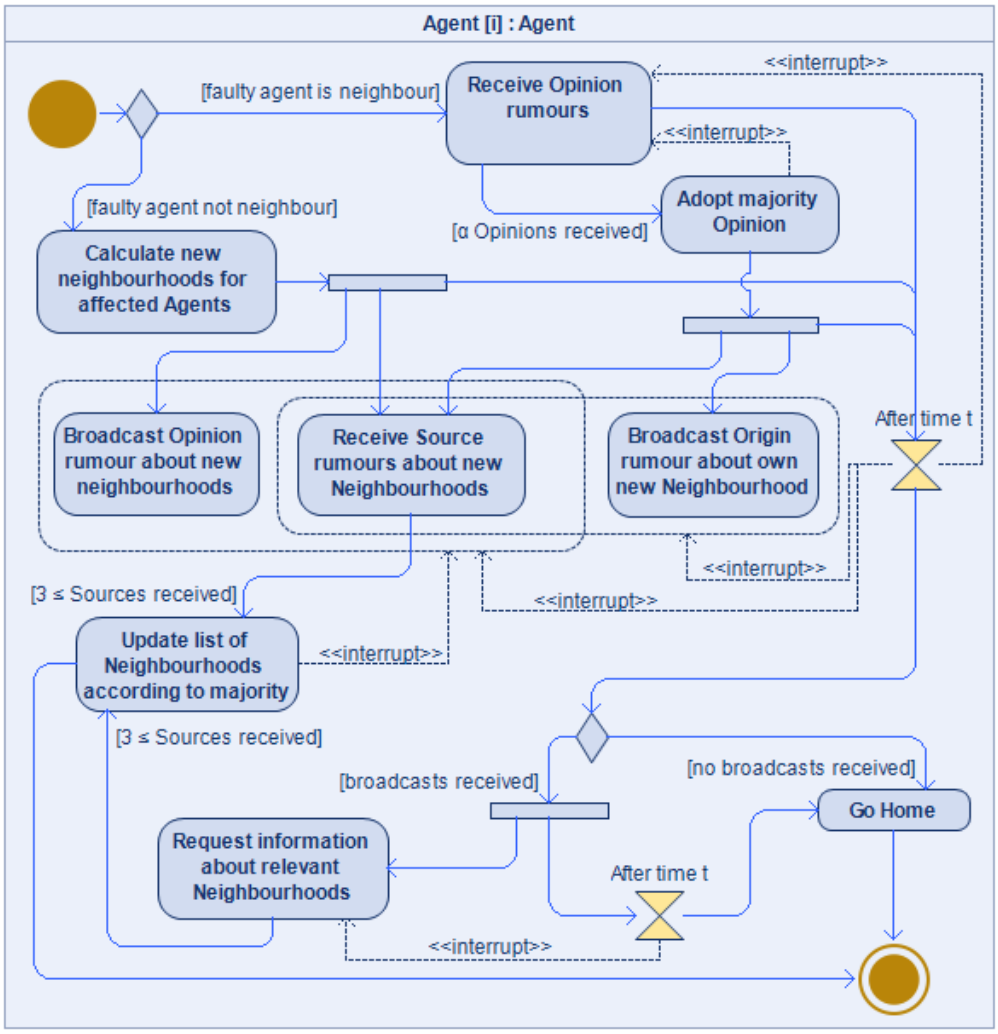

Figure 6: *(SD-25)* Activity Diagram showing the process of neighbourhood updates after agents join or re-join after mission start.

5.2.4 Overlapping Neighbourhoods

Neighbourhoods overlap, meaning that an agent is part of its own neighbourhood (as the centre agent), and it also appears as a neighbour in the neighbourhoods of other agents.

This overlap creates a connected graph of small local groups throughout the swarm. Origin rumours are broadcast to neighbours, neighbours broadcast Source rumours to their neighbours, which enables the Source rumours to propagate through the swarm; Opinion rumours then trigger the originator to make a decision.

Neighbourhoods do not perform any internal voting. Each neighbour independently:

- Receives the Origin,
- interprets it using local information,
- broadcasts a Source.

Each agent forms a neighbourhood with its three geographically closest peers (based on assigned initial sectors). Neighbourhoods overlap, provide multiple independent Source rumour for each Origin rumour, and serve as the initial ”cross-check” group for validating events.

5.2.5 Neighbourhood Recalculation on Failure or Leave

Neighbourhoods must be updated when:

- An agent leaves the swarm intentionally (e.g. battery replacement).
- An agent fails or is declared missing due to lost pulses.
- A new agent joins the swarm.
- An agent rejoins the swarm.

In these cases, each affected agent recomputes its three closest neighbours based on the latest available sector or position information. This ensures that neighbourhoods maintain their size (1 + 3) and that cross-checking coverage remains consistent.

Neighbourhoods do not vote as a group; Each neighbour independently creates Source rumours. The requirement to receive three Source rumours before interpeting an event and possibly broadcasting an Opinion gives each neighbourhood a built-in tolerance to some faulty behaviour, provided that no neighbourhood contains too many faulty agents.

5.3 Agent Architecture

Each agent is a UAV equipped with standard flight systems (example found in fig. 22) and a protocol module for swarm-level coordination. A simplified Internal Block Diagram (IBD) (fig. 7) models the internal structure relevant for the protocol module.

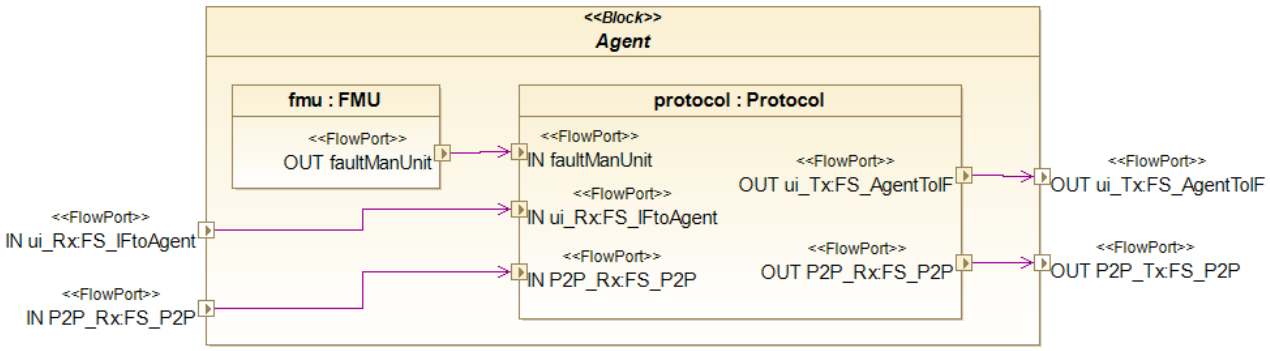

Figure 7: *(SD-10)* IBD of agents' internal structure. Even though flowports are named P2P_Rx / P2P_Tx in the diagram, they conceptually represent broadcast send/receive on a shared channel. Flow specifications for the flowports can be seen in fig. 11.

The protocol module never overrides flight-critical behaviour and does not replace any of an agent's systems; The protocol module reads inputs (from the FMU and swarm communication module) and outputs high-level decisions (e.g. "search sector S", "return home", "relay from position P") that are then translated into flight commands by the existing control stack.

Moreover, inside each agent, the protocol module contains:

- the Rumour Mill logic (Origin/Source/Opinion handling),
- the Sector Allocation Market logic (sector cost, budgets, bidding),
- the Task & Role Allocation logic (agent state machine),
- local information storage (StaticInfo and DynamicInfo).

The protocol module communicates with the FMU (to receive health states), the swarm communication module (to send and receive messages), and a local interface to flight/navigation systems (to request high-level actions).

5.4 Protocol Module Structure

A simplified architecture of the protocol is defined by fig. 8.

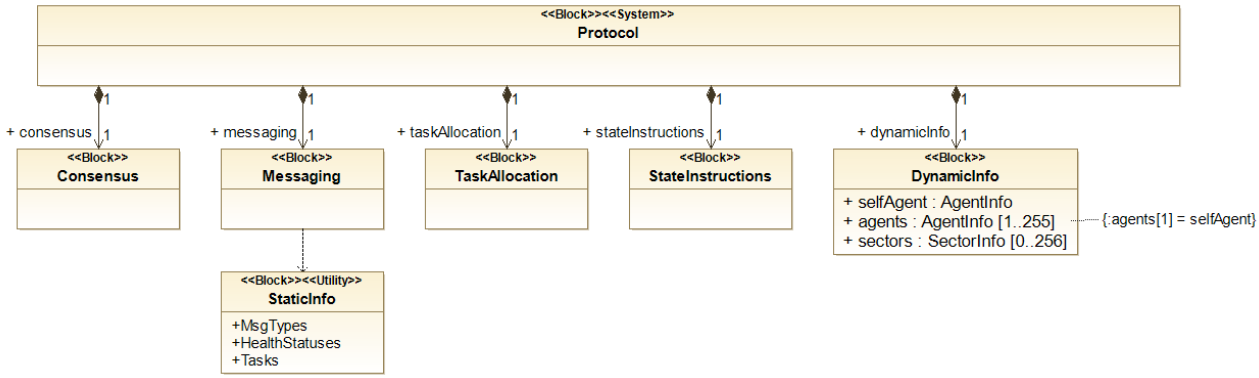

Figure 8: *(SD-13)* BDD of which main parts the protocol module contains. DynamicInfo holds lists containing information about agents in the swarm (where the first index is always the agent itself) and about sectors. StaticInfo holds enumerations of message types, health statuses, and tasks.

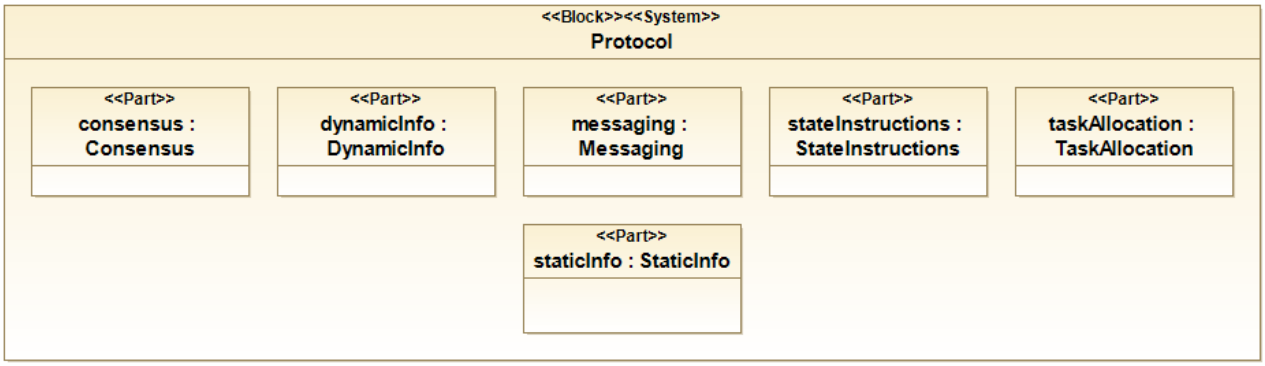

Figure 9: *(SD-16)* IBD of the protocol module's main parts.

5.4.1 Sub-modules

The protocol module consists of several submodules that relate to each other through shared memory and message flows:

- Messaging, responsible for:
  - Receiving messages from the FMU.
  - Receiving all incoming broadcasts (pulses, Origins, Sources, Opinions).
  - Mapping msgID values to internal handlers.
  - Discarding malformed or obviously stale messages.
  - Build and send outgoing messages with correct msgID and data types.
- Consensus, responsible for:
  - Tracking active Origin rumours and their Source rumours and Opinion rumours.
  - Determining when enough Source rumours exist to interpret the event and to possibly form an Opinion rumour.
  - Tracking Opinion rumours and detect when the chosen consensus threshold α (e.g. 70% in examples) is reached for a given Origin.
    Consensus handles events such as:
  - "Three Sources reached for rumour R" $\rightarrow$ trigger to vote on truth, and create an Opinion or update local information.
  - "Consensus threshold reached for rumour R at originator" $\rightarrow$ trigger to make a decision.
  - Applying ownership changes when Opinions are announced.
- TaskAllocation, responsible for:

- Initiating bids for sectors (creating appropriate Origins).
- Evaluating other agents' bids when forming Sources.
  TaskAllocation uses the Rumour Mill for all sector ownership changes to ensure consistent views of who owns which sector.

- StateInstructions, responsible for:
  - Implementing the agent's state machine (start, task, search, found, confirmed, bad_sensor, secondary_task, return, failed).
  - Mapping health status, sector ownership, and commands into roles (search, relay, return, etc.).
  - Triggering sector releases and task changes when roles change.
- DynamicInfo, responsible for
  - Maintain AgentInfo list: members, neighbours, positions, health statuses, tasks, subjectFound, subjectLocation.
  - Maintain SectorInfo list: owner, corners, value, searched flag, cost.
  - Maintaining the agent's budget (maxBudget, budgetRoof, currentBudget).
- StaticInfo, responsible for:
  - Enumerations and constants such as MsgID, Task, HealthStatus, etc.
  - Possibly constant timeouts (pulse timeout, auction wait times) and global configuration.

5.4.2 Internal Interfaces & Flowports

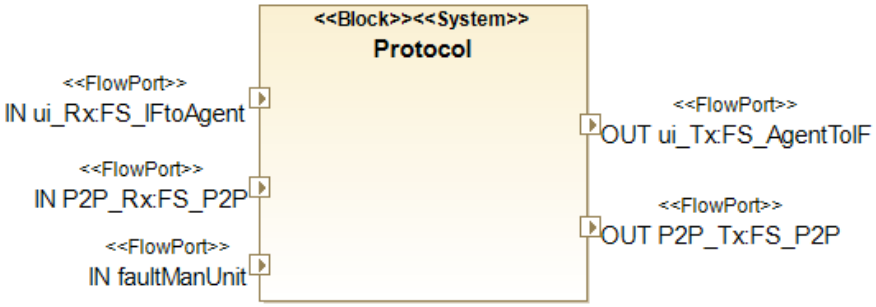

Figure 10: *(SD-15)* Blackbox view of the protocol. Flowports named ui_Rx and ui_Tx are to receive broadcasts from and broadcast to SwarmInterface. Flowports named P2P_Rx and P2P_Tx are to receive broadcasts from and broadcast to other agents. The flowport named faultManUnit is for receiving messages from the FMU. Flow specifications for the flowports can be seen in fig. 11.

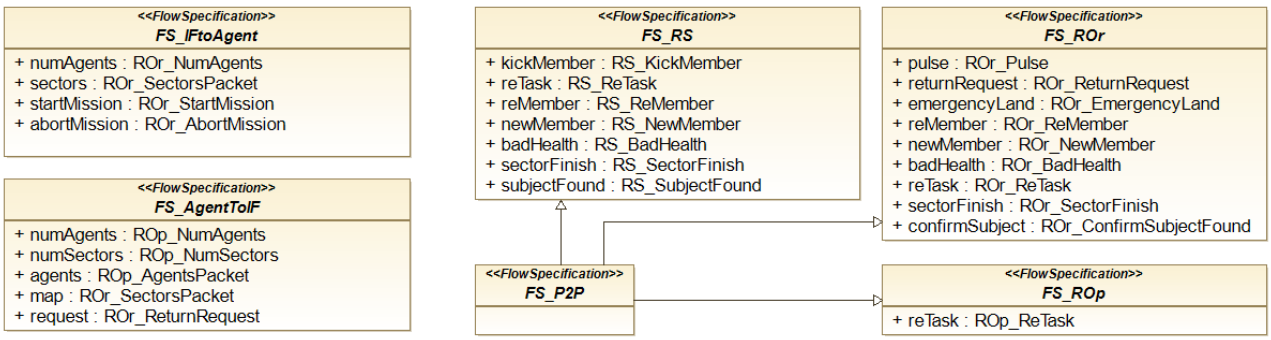

Figure 11: *(SD-09)* Flow specification for flowports. ROr means Rumour Origin, RS means Rumour Source, and ROp means Rumour Opinion.

5.5 Communication Model

The protocol module uses a broadcast-based wireless communication model, which means that when an agent broadcasts a message, all agents within range can receive it, assuming no interference or packet loss.

There is no concept of fixed point-to-point links or explicit routing within the protocol module. Instead, to spread information throughout the swarm, the swarm relies on broadcasts from agents, redundancy (retransmissions over time), and overlapping radio ranges.

A graphical representation of incoming and outgoing broadcasts can be seen in fig. 10, and message types can be seen in fig. 11.

5.5.1 Agent ↔ Agent Communication

Agents exchange information by broadcasting messages on a shared channel. Typical broadcast messages include:

- Pulses (heartbeats).
- Origin rumours, such as:
  - Sector bidding.
  - Sector completion.
  - Confirm Subject found.
  - Bad health.
- Source rumours, such as:
  - Originator no longer a member of the swarm.
  - Join/rejoin announcements for an Originator.
  - Subject found.
- Opinion rumours, such as:
  - Agent X won bid for sector Y.
  - Originator should become a communication relay.
  - The number of members in the swarm.

All messages include the sender's AgentID, so receivers can update their local AgentInfo and justify who sent what. Any agent within range may receive the message; agents outside range may miss it.

The protocol module does not assume that every broadcast reaches every agent, instead:

- Messages may be dropped due to interference or distance.
- Different agents may hear different subsets of transmissions.
- Eventual propagation is achieved because agents repeat or rebroadcast information over time and because neighbourhoods overlap spatially.

5.5.2 Agent ↔ SwarmInterface Communication

The SwarmInterface participates in the same broadcast channel as the swarm, broadcasting and receiving messages such as:

- Initialisation data (e.g. initial sector map, intended swarm size).
- High-level commands (start mission).
- Swarm status updates (health changes, return requests, Subject found).
- Perceived number of agents in swarm.
- Perceived number of sectors.

From the protocol module perspective, these messages are just additional broadcasts with specific msgIDs that indicate that they originate from the SwarmInterface.

Most operational commands injected by the SwarmInterface are turned into Origin rumours and processed by the Rumour Mill. A small number of initialisation messages (see Section 6.5.2) are treated as trusted inputs and do not trigger the rumour lifecycle.

Note that the communication between the swarm and SwarmInterface is conceptual in the current version but included in the architecture.

5.5.3 Pulse Mechanism

Each agent periodically broadcasts a pulse containing their AgentID, position, and timestamp of when that position was measured.

Any agent within range can receive the pulse and update its AgentInfo. Pulses are used to:

- Detect the presence or disappearance of agents.
- Update positions.
- Maintain neighbourhood membership.

If an agent does not receive pulses from a neighbour within a configured timeout, it initiates a Rumour Mill event (a Source stating that the neighbour is missing). This may eventually lead to the neighbour being considered no longer a member of the swarm, its sectors being reallocated, and neighbourhoods recalculated.

5.6 Static vs. Dynamic Information

Each agent maintains two main categories of information:

- Static information – predefined and identical for all agents, such as message IDs, task IDs, health stauses, and any constant parameters.
- Dynamic information – updated at runtime, including:
  - Known agents and their statuses (AgentInfo),
  - neighbourhood memberships,
  - known sectors, their owners, and cost (SectorInfo),
  - budget
  - sector bids,
  - current state and role.

Static information defines the ”vocabulary” and structure of the protocol module; Dynamic information records the current situation.

5.6.1 Static Information

Static information is defined before the mission, is identical for every agent, and does not change at runtime. Static Information includes:

- Enumerations of message IDs (msgID).
- Enumerations of task IDs (taskID).
- Enumerations of health statuses (supplied by the FMU).
- Role identifiers.
- Configuration parameters (e.g. pulse intervals, timeouts).

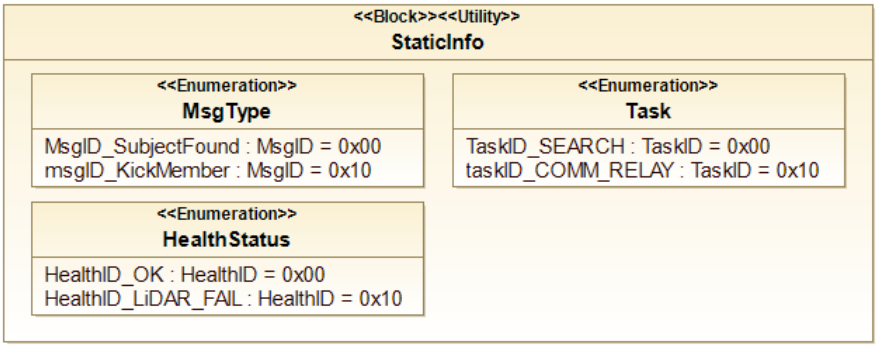

Figure 12: *(SD-14)* Examples of what StaticInfo contains.

5.6.2 Dynamic Information

Dynamic information is updated throughout the mission and stored locally on each agent, and ensures that each agent can make decisions based on its own up-to-date local view, while the Rumour Mill works to keep these local views consistent across the swarm.

Dynamic information includes:

- AgentInfo (fig. 13) entries (per known agent):
  - agentID: Unique ID of agent X.
  - member: If agent X is considered a member of the swarm.
  - neighbour: If agent X is considered a neighbour by the owner of the list.
  - position: Last known latitude, longitude, and altitude of agent X (received through Pulse).
  - posTimestamp: Time that agent X broadcasted their position (received through Pulse).
  - posReceived: Time that the owner of the list received the broadcast from agent X.
  - healthStatus: Last known health status of agent X.
  - task: Last known task of agent X.
  - subjectFound: If agent X is the one that found the Subject (received by Source rumour).
  - subjectLocation: Latitude and Longitude of the Subject's position (received through Source rumour).
- SectorInfo (fig. 13) entries (per sector):
  - sectorID: Unique ID of sector S.
  - ownerID: ID of agent (or 0 for unassigned) that has been allocated sector S.
  - nw: Latitude and longitude of north-west corner of sector S.
  - ne: Latitude and longitude of north-east corner of sector S.
  - sw: Latitude and longitude of south-west corner of sector S.
  - se: Latitude and longitude of sector X's south-east corner.
  - value: Value based on the probability that the Subject will be found in sector S. Probability is based on Hot Regions.
  - cost: Cost of the sector.
  - searched: TRUE if ownerID has finished searching sector S, otherwise FALSE.
- Budget:
  - maxBudget: Static value of the max value of the owner's budget.
  - budgetRoof: Maximum spending allowance for the owner, based on the owner's battery level.
  - currentBudget: Current value of the owner's budget.
- SectorCost entries (per sector):
  - sectorID: Unique ID of sector S.
  - value: Value based on the probability that the Subject will be found in sector X. Probability is based on Hot Regions.
  - initSector: Latitude and longitude of the first initial sector of the owner.
  - cost: Cost (f(distance, value)) of sector X based on value and initSector. Calculated by TaskAllocation.

5.7 Information Model & Data Types

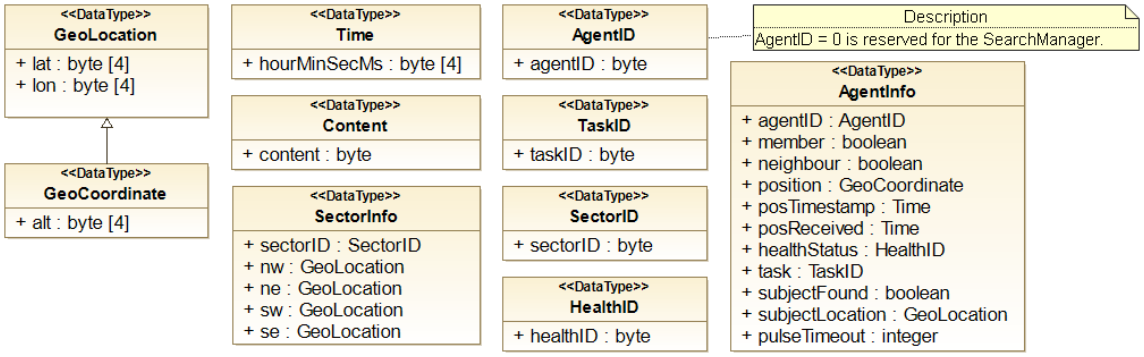

Figure 13: *(SD-05)* Datatypes used in the system.

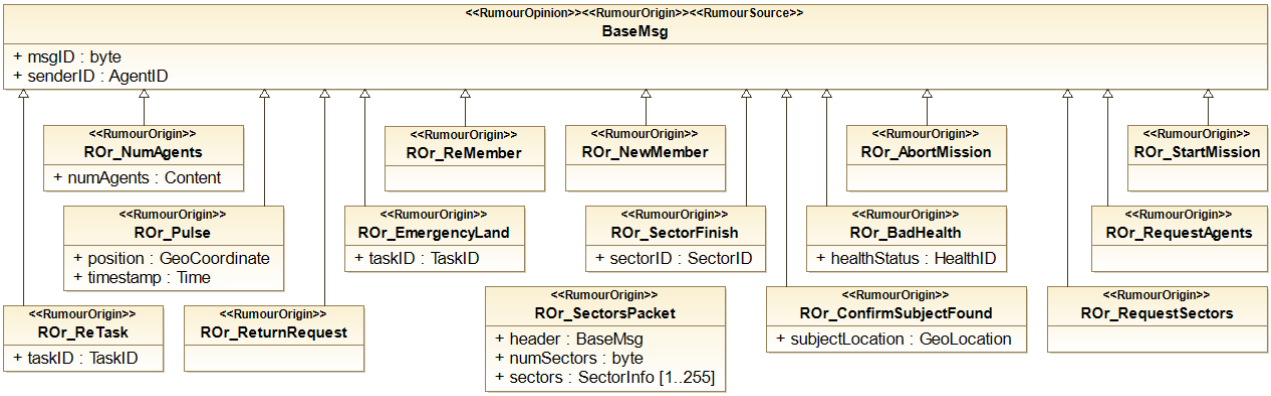

Figure 14: *(SD-06)* Contents of Origin Rumours.

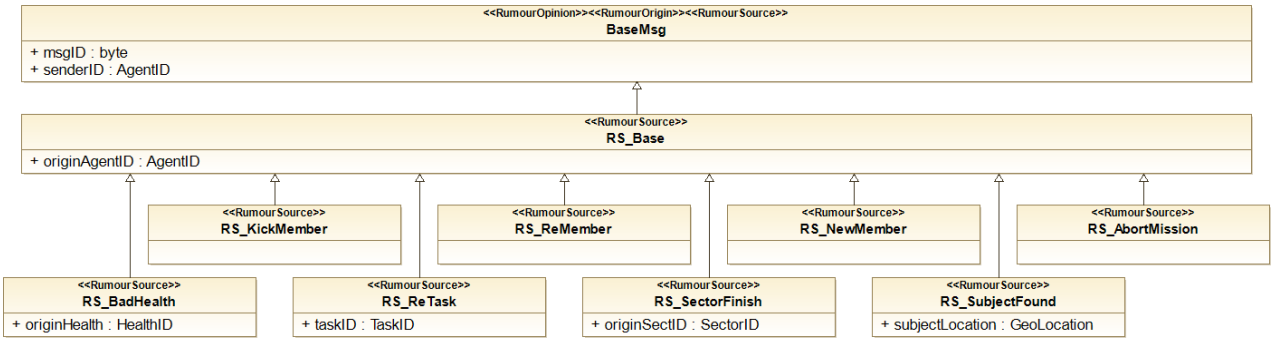

Figure 15: *(SD-07)* Contents of Source Rumours.

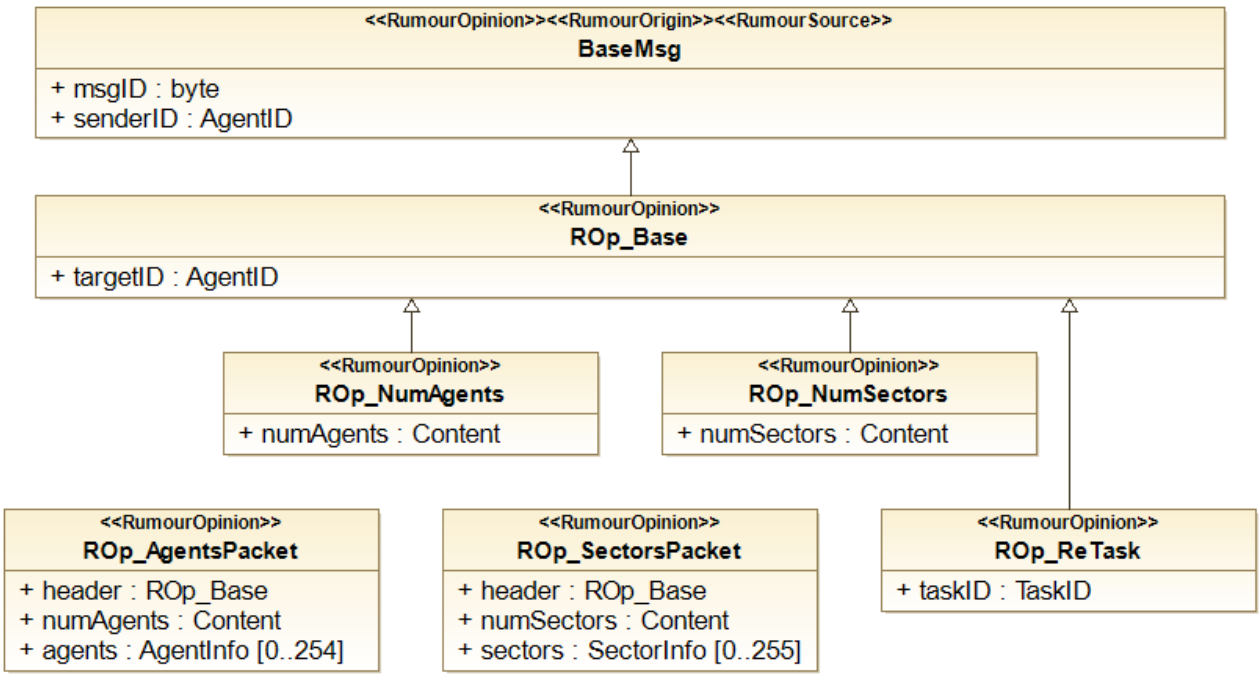

Figure 16: *(SD-08)* Contents of Opinion Rumours.

5.8 External Interface Architecture

The Search Manager interacts with the swarm through the SwarmInterface, where the core interactions are:

- Input to the swarm:
  - Initial Search Area.
  - Hot Regions.
  - Mission start / pause / abort commands.
  - (potentially) mission updates.
- Output from the swarm:
  - Health status summaries (e.g. ”Agent A emergency landed at location L”).
  - Return requests (battery changes).
  - Sector completion status.
  - Subject found.

The External Interface Architecture defines how the protocol module interacts with external systems, primarily the SwarmInterface.

The SwarmInterface is modelled as a distinct external boundary component. This design decouples human-driven command logic from decentralised swarm behaviour and supports future expansion (e.g., GUI, map view).

Note that only the high-level interface between SwarmInterface and the protocol module is in scope; concrete message encoding and RF communication details remain out of scope for this project.

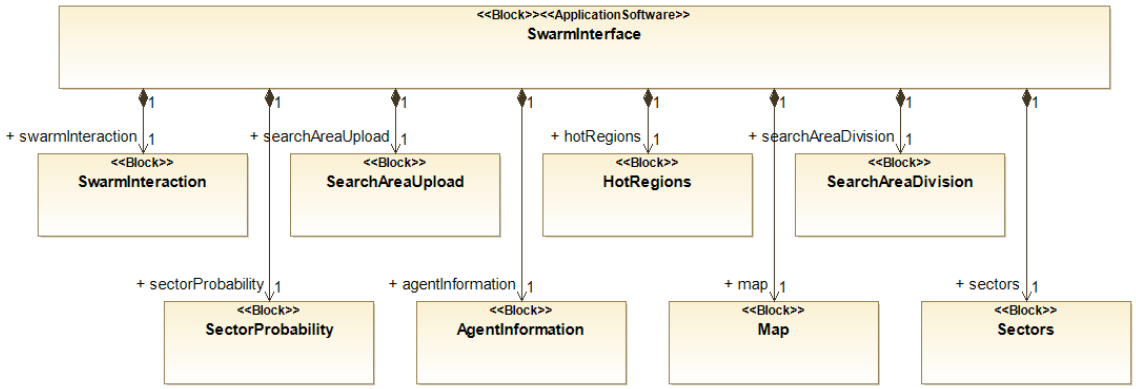

Figure 17: *(SD-11)* BDD of a conceptual design of the SwarmInterface.

Figure 18: *(SD-12)* Blackbox view of the SwarmInterface.

The SwarmInterface aggregates protocol module events into human-readable information, such as ”Agent A requests battery replacement” or ”Agent B degraded”, to allow the Search Manager to prepare appropriate actions (e.g. swapping batteries when an agent returns to base).

From the protocol module perspective, the SwarmInterface provides an input channel for high-level commands and mission configuration (initial sector map, Hot Regions), and an output channel for aggregated swarm status and event reports.

5.9 Communication Context

The communication context summarises how all the previously described elements fit together:

- Agents communicate with one another by broadcasting messages on a shared wireless channel that carry pulses and Rumour Mill traffic.
- Agents communicate with the SwarmInterface on the same shared wireless channel to receive mission configurations, updates, and to report events.

Neighbourhoods provide the local structure that seeds rumour propagation.

Rumours ensure that important events and decisions are eventually known by all agents.

The Sector Allocation Market and Task & Role logic run on top of this communication fabric to allocate work and determine agent behaviours.

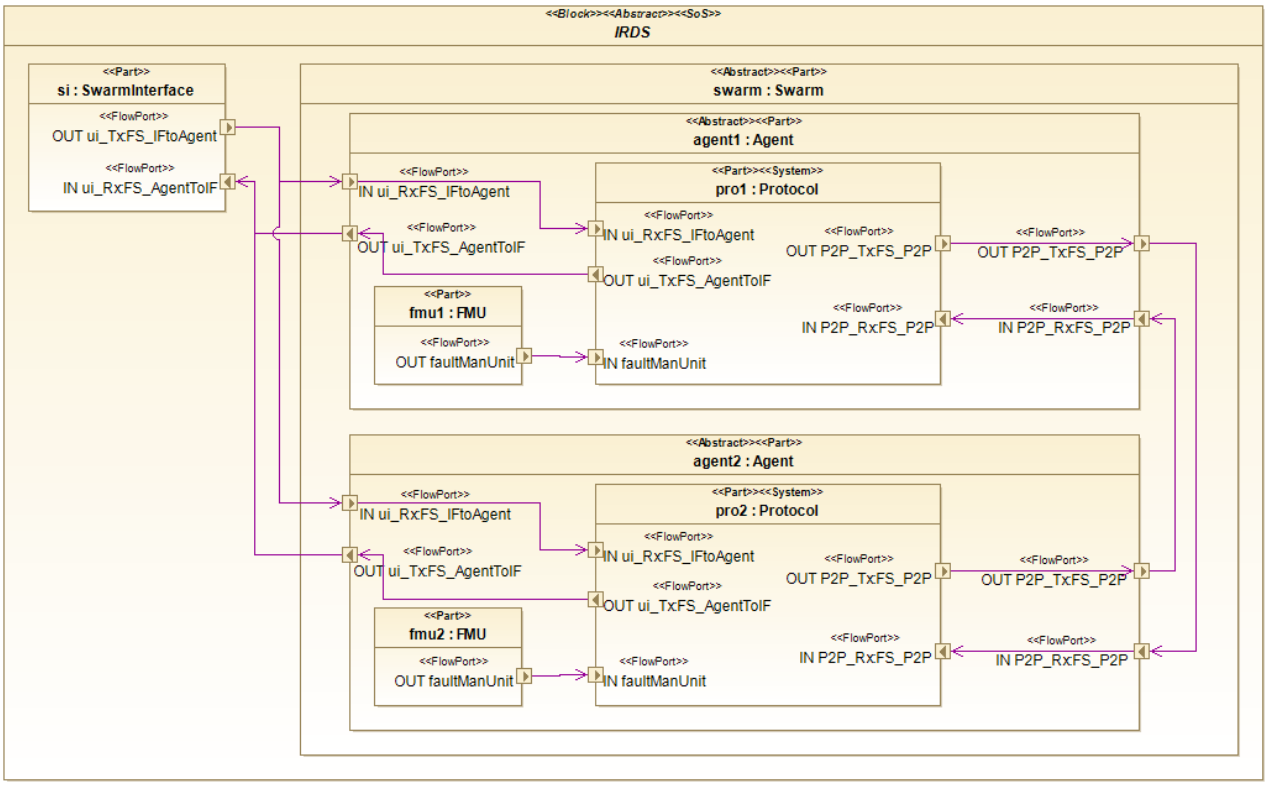

Figure 19: *(SD-21)* IBD over communication context. Even though flowports are named P2P_Rx / P2P_Tx in the diagram, they conceptually represent broadcast send/receive on a shared channel. Flow specifications for the flowports can be seen in fig. 11.

6 Rumour Mill Protocol (Consensus Layer)

The Rumour Mill is the core consensus mechanism of the IRDS protocol module and is responsible for ensuring that all agents eventually agree on how to interpret important events (such as degraded health, sector completion, or Subject detection) and on the decisions that follow from those events.

In this work, the combination of Origin rumours, Source rumours, Opinion rumours, and their lifecycle is referred to as the **Rumour Mill**. To the best of the project group's knowledge, using this particular rumour structure and lifecycle as a decentralised consensus mechanism for UAV SAR swarms is a novel concept introduced by this project.

The Rumour Mill is fully decentralised: Each agent executes the same logic, uses only locally stored data and received messages, and no agent is given special decision authority.

6.1 Purpose & Scope

The Rumour Mill serves two main purposes:

1) Shared interpretation of events
   When an event occurs (e.g. "Agent A has degraded health"), the swarm needs a consistent answer to "what actually happened?".
2) Shared decisions linked to those events
   Once the event is interpreted, the swarm needs to decide how to react (e.g. "Agent A should become relay").

The Rumour Mill is used for almost all operational decisions that require swarm-wide agreement. A small set of initialisation-time inputs from the Search Manager are treated as trusted and do not use the full rumour lifecycle (see Section 6.5.2).

The Rumour Mill itself does not decide how to value sectors or how to map decisions into specific roles. Instead, it provides a consensus layer that the Sector Allocation Market (Section 7) and Task & Role Allocation (Section 8) build on top of.

6.2 Message Model & Message IDs

All protocol module messages are broadcast over a shared wireless channel, meaning that when an agent broadcasts, any agent within range may receive the message. Messages have a common structure that includes, at minimum:

- msgID: Type identifier for the message.
- senderID: ID of the agent that broadcast the message.

Additional fields depend on the rumour type and event type (e.g. sectorID, healthStatus).

The msgID field is especially important as it tells the protocol which internal handler to call and therefore what to do with the message, for example, different msgIDs may distinguish between:

- Health-related Origins (e.g. degraded, bad_sensor).
- Sector-related messages (bids, releases, and completions).
- Subject-related messages (found, confirmation).
- Search Manager commands injected through the SwarmInterface.

This design keeps the message parsing logic simple and makes it easy to extend the protocol by defining new msgID values and handlers without changing the overall architecture.

6.3 Rumour Types

The Rumour Mill uses three types of rumours:

- Origin Rumour: Created by an agent that directly observes or generates an event.
- Source Rumour: Created independently by agents that receive an Origin and interpret it.
- Opinion Rumour: Created by agents that have received enough Sources to summarise their view.

Together, these three types of rumours and their lifecycle implement the Rumour Mill.

6.3.1 Origin Rumours

An Origin rumour represents a local observation or decision made by an agent. Examples include:

- ”I have degraded health” (triggered by FMU).
- ”I have completed sector S.”
- ”Subject found at location L.”
- ”I want to purchase sector S.”
- ”Search Manager issued command C.”

The agent that detects or generates the event (the originator) creates the Origin rumour and broadcasts it to its neighbours.

6.3.2 Source Rumours

When a neighbour receives an Origin rumour, it independently evaluates the event using its local information and creates a Source rumour. A Source rumour expresses that neighbour’s interpretation of the Origin rumour, for example:

- ”Agent A is no longer a member of the swarm” (triggered by a lack of Pulse).
- ”I confirm that Agent A appears degraded.”
- ”I agree that Agent A has completed sector S.”
- ”I consider Agent A’s bid on sector S valid.”
- ”I consider Agent A’s bid on sector S invalid (conflict).”

Each Source rumour includes the ID of the originator. Source rumours are broadcast to the swarm so that other agents can accumulate multiple interpretations of the same event. An agent that holds three Source rumours can broadcast all of them to other agents.

6.3.3 Opinion Rumours

When an agent receives **three** Source rumours referring to the same Origin (i.e. three independent interpretations for that event), it votes internally on how to interpret the event, and then **possibly** creates an Opinion rumour.

An Opinion rumour expresses a recommended action or conclusion, for example:

- ”Agent A should change to secondary_task (relay).”
- ”Agent A should return to base.”
- ”Agent A is now the owner of sector S.”

Opinion rumours are broadcast to the swarm. They summarise the local view built up from multiple Sources and are used by originators to make a final decision.

6.3.4 Rumour Mill as Named Mechanism

In this document, the term Rumour Mill specifically refers to the set of rumour types (Origin, Source, Opinion), and the full lifecycle described in Section 6.4.

All consensus-related behaviour in the protocol module is built on this mechanism.

6.4 Rumour Lifecycle

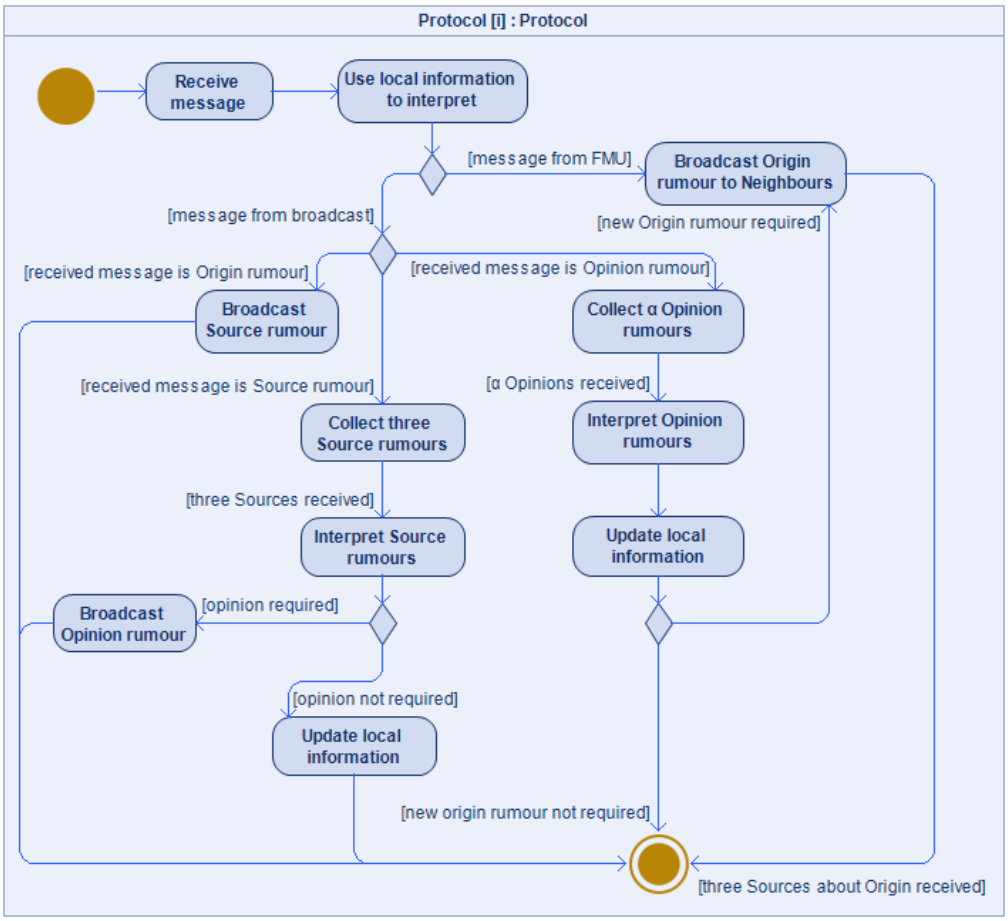

Figure 20: *(SD-17)* Activity Diagram showing the sequence of actions a protocol module takes upon receiving or generating rumours.

The lifecycle of rumours allows the swarm to converge on consistent decisions without any agent having special authority. The complete lifecycle of a rumour-based decision consists of five phases:

1) Event → Origin
   a) An agent detects or generates an event (e.g. FMU health change, sector completion, subject found).
   b) It creates an Origin rumour and broadcasts it to its neighbours.
2) Origin → Sources
   a) Each neighbour independently interprets the Origin.
   b) It creates a Source rumour and broadcasts it to the swarm.
3) Sources → Opinions
   a) Any agent who has received three Source rumours (from three distinct agents) considers that it has enough information to interpret the event.
   b) It makes a decision, and possibly creates an Opinion rumour and broadcasts it.
   Note that any agent (not just the originator or its direct neighbours) can reach the ”three Sources” threshold, come to a decision, and possibly broadcast an Opinion rumour.
4) Opinions → Decision at the Originator
   a) The originator collects Opinions broadcast by the swarm.
   b) Once the originator has received Opinions from at least a certain fraction of the swarm (the consensus threshold, see Section 6.6), it makes an internal decision.

c) The decision may include changes such as:
   - ”I will switch to secondary_task (relay).”
   - ”I will return to base.”
   - ”I am now owner of sector S.”

   The decision is made locally at the originator; no single Opinion decides the outcome, and no external authority is required.

5) Decision → New Origin (Decision Announcement)
   a) After deciding, and if needed, the originator broadcasts a new Origin that describes the decision (e.g. ”I will return home”, ”I have switched to secondary_task”).
   b) Other agents update their local state (AgentInfo, SectorInfo, state machine, etc.) according to the decision.
      This closes the rumour cycle. All agents now have a consistent view of the event and its outcome, subject to message delivery.

6.5 Trigger Events

6.5.1 Events Using the Full Rumour Lifecycle

The Rumour Mill is used for almost all operational events that require swarm-wide agreement. Typical trigger events include:

- FMU-driven health events:
  - Degraded health (e.g. bad_sensor).
  - Low battery (when it implies behaviour changes).
  - Critical failure (where possible).
- Search-related events:
  - Sector completed.
  - Sector cannot be searched (e.g. blocked, unreachable).
- Subject-related events:
  - Subject found.
  - Subject confirmation.
- Agent membership events:
  - New agent join.
  - Agent rejoin.
  - Agent missing (detected through pulse timeout).
- Sector Allocation Market-related events:
  - Bid to purchase a sector.
  - Release of sector ownership.
- Operational commands from the Search Manager (e.g. start, pause, abort, change Search Area), as delivered via the SwarmInterface.

All of these are represented as Origins (except for the detection of a missing agent, which is handled as a Source rumour) and processed through the full rumour lifecycle.

6.5.2 Initialisation Events Bypassing Rumour Mill

Most commands are translated into Rumour Mill events so that the entire swarm can respond consistently. A small set of initialisation-time commands from the Search Manager are treated as trusted configuration for the mission and do not use the full Rumour Mill process:

- The initial intended number of agents in the swarm.
- The initial Search Area and sector division (the sector map).

After initialisation, all further operational updates and decisions use the Rumour Mill.

6.6 Consensus Threshold & Parameters

Interpretation of and decisions based on Source rumours are made once three Source rumours have been received and an internal vote has been done. This is based on the principles of triple modular redundancy.

The originator decides once it has collected enough Opinion rumours for a particular Origin rumour. The required fraction of Opinion rumours is denoted as α, where:

$$0.5 < \alpha < 1$$

In this document, 70% is used as an illustrative example, meaning that the originator decides once it has received Opinions from at least 70% of the agents it currently believes are in the swarm. This 70% value is not fixed by the protocol, it is a tunable design parameter, and the best value will depend on:

- Communication reliability and message loss.
- Swarm size.
- The expected number of faulty or unreachable agents.
- How quickly the swarm must respond to events.

Determining an appropriate threshold α for deployment is left for future work and should be investigated through simulation or analysis. The examples in this document use 70% purely for concreteness.

6.7 Properties of the Rumour Mill

The Rumour Mill provides several desirable properties for the protocol module:

- Decentralisation:
  - There is no central coordinator or leader.
  - All agents run identical logic and participate symmetrically.
  - Interpretations are made locally once three Source rumours have been received.
  - Decisions are made locally by the originator based on Opinion rumours collected from the swarm.
- Eventual Consistency:
  - Messages are broadcast in a lossy network; not all agents hear all messages immediately.
  - However, Origins, Sources, and Opinions are rebroadcast and spread over time.
  - Provided that communication is occasionally available, all non-faulty agents eventually receive the decision-Origin and converge on the same view of the event.
- Fault Tolerance:
  - By requiring multiple independent Sources before interpreting and possibly broadcasting an Opinion rumour, the protocol module reduces the influence of isolated faulty interpretations.
  - By requiring a consensus threshold α of Opinion rumours before deciding, the originator avoids being swayed by a small number of faulty agents.
  - Neighbourhoods and overlapping broadcast ranges ensure that events are cross-checked by multiple agents.
  - A more detailed fault-tolerance analysis (including Byzantine assumptions and neighbourhood constraints) is provided in Section 9.
- Extensibility and Implementation Friendliness:
  - New event types can be added by defining new msgID values and handlers without changing the basic Origin/Source/Opinion structure.
  - Messages remain small and simple, suitable for microcontrollers and limited-bandwidth radio links.
  - The algorithm uses only basic counting and storage, making it practical for embedded systems.

6.8 Example Rumour Cycles

This section gives brief examples of how the Rumour Mill works in typical situations.

These scenarios show how the Rumour Mill links local observations to swarm-wide, consistent decisions without a central controller.

6.8.1 Degraded Health Event

1) FMU on agent A detects bad_sensor.
2) A releases its sectors and broadcasts an Origin: ”I (A) am degraded: bad_sensor.”
3) Neighbours of A interpret it and broadcast Sources confirming the degraded state.
4) Any agent (including neighbours of A) that receives three Sources updates A’s sectors to unassigned, and creates an Opinion rumour (e.g. ”A should release its sectors and become relay”).
5) A collects Opinion rumours; Once Opinion rumours from $\geq \alpha$ fraction of agents have arrived, A internally decides to switch to secondary_task (e.g. relay).
6) A broadcasts a decision-Origin announcing its new role.
7) The Sector Allocation Market (Section 7) reassigns A’s sectors to other agents through bidding.

6.8.2 Subject Found Event

1) Agent A detects the Subject at location L and broadcasts an Origin: ”Subject found at L.”
2) Neighbours receive the Origin rumour and go to L to confirm that the subject is at L.
3) Neighbours broadcast Source rumours based on their findings (e.g. ”Subject not found at L” or ”Subject found at L”).
4) Once any agent has three Sources (including originator and neighbours), it interprets the event by either continuing its current task (”Subject not found at L”) or updating their state machines to enter roles appropriate for mission completion (usually return to base).

7 Sector Allocation Market (Workload Layer)

The Sector Allocation Market determines which agent is responsible for which sectors at any given time by distributing search workload across the swarm using simple market-like rules:

- Sectors have costs.
- Agents have budgets.
- Agents "buy" sectors they can search efficiently.
- Agents release sectors when they can no longer search reliably.

All changes in sector ownership are confirmed through the Rumour Mill (Section 6), so that all agents eventually agree on who owns which sector.

The Sector Allocation Market operates only on workload distribution (who searches which sectors). It does not itself decide the concrete behaviour of the agent, which is handled by the Task & Role Allocation layer (Section 8) based on sector ownership and health status.

7.1 Market Overview & Rationale

Instead of using a static or centrally assigned mapping of sectors to agents, the protocol module uses a distributed market approach where:

- Each sector has a **value** determined by its probability (importance).
- Each agent has a **budget** that reflects its energy levels and possible workload.
- Each agent evaluates the cost of a sector using a **personal cost function**.
- Sectors are bid for in order of importance (highest probability first).
- Agents bid for sectors they can search efficiently.
- Degraded agents release sectors that they can no longer search reliably.
- Healthy agents take over released sectors through new bids.
- Agents generate rumour events when they want to acquire or release sectors.
- Final ownership decisions are made through the Rumour Mill.

This approach allows the swarm to:

- Balance workload dynamically.
- Prioritise high-probability areas.
- Adapt to changing agent health and positions.
- Continue operating even when agents fail or leave the swarm.
- Tend to have sectors handled by agents that are nearby and have enough battery.
- Have no single assignment authority that can fail; all agents participate.

This Sector Allocation Market yields a simple but flexible mechanism that fits on top of broadcast communication and the Rumour Mill.

7.2 Sector Cost

Each sector S is assigned a cost that reflects both how important it is and how hard it is for a particular agent to search it.

The value of a sector is calculated by the SwarmInterface, where the SwarmInterface:

1) Receives Hot Regions from the Search Manager.
2) Divides the Search Area into rectangular sectors.
3) Assigns a probability to each sector based on the Hot Regions.

From the probability, each sector is then given a value where high-probability sectors receive a higher value, and low-probability sectors receive a lower value. This encourages the Sector Allocation Market to favour assigning high-probability sectors to agents.

Thereafter, for each agent A and sector S, an an additional cost component is computed based on the distance between the centre of A's initial sector and the centre of S. Agents use this cost when deciding which sectors to bid for.

Future work must decide the exact algorithm for this.

7.3 Agent Budgets & Budget Dynamics

Each agent A is assigned a budget B_A, which is primarily based on its battery level. The budget represents how many and which sectors the agent can afford to purchase. At mission start (or when joining), an agent's initial budget is set based on its available battery meaning that higher battery has a larger budget, and lower battery has a smaller budget.

Budget dynamics follow these rules:

- When agent A successfully purchases a sector S through the Sector Allocation Market, B_A decreases by the cost of S.
- When agent A finishes searching sector S and confirms completion via the Rumour Mill, B_A increases by the cost of S (but does not go over what is allowed by the battery level).
- Agent A may only bid on sectors if its remaining budget is sufficient to cover the sector's cost.

In the current design, it is assumed that agents behave non-maliciously with respect to their own budget variables, i.e., agents do not intentionally corrupt their budgets or attempt to bid on sectors they cannot afford. Handling behaviours that target budget manipulation is left as future work.

7.4 Initial Sector Allocation

The initial sector allocation aims to:

- Ensure that each agent has at least one sector to search.
- Provide good coverage in high-probability areas.
- Establish initial positions for neighbourhood computation.

The process consists of two phases: Guaranteed First Sector and Initial Auction Loop.

7.4.1 Initial Bidding Process

Activity diagram showing the process of bidding for sectors after initial sectors have been given

7.4.2 Guaranteed First Sector

Each agent is assigned at least one sector purchased at the sector's value, without going through the full bidding process. This initial mapping:

- Uses probability information (favour high-probability regions).
- Ensures that every agent starts with something to do.

The positions of these initial sectors are also used as the basis for neighbourhood formation (one agent + its three closest neighbours) and sector costs.

7.4.3 Initial Auction Loop

After the first sector, agents may attempt to purchase additional sectors using the Sector Allocation Market:

1) Each agent evaluates which sectors are attractive based on its cost function and remaining budget.
2) For each candidate sector S, the agent creates an Origin rumour expressing an intention to buy S (using an appropriate msgID).
3) Neighbours interpret this bid and broadcast Source rumours evaluating:
   - Whether S is currently unassigned or conflicts with another owner,
   - whether the bidding agent's budget appears sufficient,
   - any other validity conditions.
4) Once any agent sees three Source rumours for this Origin rumour, it creates an Opinion rumour summarising the interpretation (e.g. "bid valid" or "bid invalid").
5) The bidding agent collects Opinions until the consensus threshold α is met, then decides whether the bid succeeded.
6) The bidding agent broadcasts a decision-Origin announcing the outcome (e.g. "I own sector S" or "bid rejected").

This process repeats until agents run out of budget. Unassigned sectors keep ownerID = 0 and remain available for future bidding.

7.5 Ownership Changes via Rumour Mill

All changes in sector ownership (initial purchases, reassignments, and releases) are confirmed through the Rumour Mill to avoid conflicting views. A typical sequence for a successful purchase of sector S by agent A is:

1) Origin (bid)
   - A broadcasts an Origin: ”I (A) want to purchase sector S.”
   - This Origin uses a specific msgID that indicates a sector bid.
2) Sources (validation), i.e.,
   - Neighbours that receive the Origin evaluate the bid based on their local information:
     - Current owner of S (if any).
     - Any conflicting bids they have observed.
   - Each neighbour broadcasts a Source saying, for example, ”Bid from A on S is valid.”
3) Opinion (summary)
   - Once an agent has three Source rumours for this Origin rumour, it broadcasts an Opinion summarising the outcome (e.g. “A should own S” or “A should not own S”).
4) Decision at A
   - A collects Opinions until it has received Opinions from at least α fraction of agents (e.g. 70% in this example).
   - A makes a local decision: bid success or failure.
5) Decision-Origin (announcement)
   - A broadcasts a new Origin announcing the decision, where success $\rightarrow$ ownerID(S) := A and A’s budget is reduced, and failure $\rightarrow$ no change to ownerID(S).

All agents update their SectorInfo and budgets accordingly. Because the final ownership decision is always announced via a decision-Origin, all agents converge on a consistent view of sector ownership.

7.5.1 Avoiding Double Assignment

The use of Rumour Mill for each ownership change prevents double assignment (two agents believing that they own the same sector):

- Conflicting bids are detected in the Source stage when agents see multiple bids for the same sector.
- Opinions summarise the outcome of these conflicting interpretations.
- The originator’s decision is only accepted once it is announced in a decision-Origin that the swarm sees.

Because all agents eventually receive the same decision-Origin, they cannot permanently disagree about who owns a sector.

7.6 Reallocation Due to Degradation or Failure

When an agent becomes degraded or fails the Sector Allocation Market must quickly reassign its sectors to maintain search coverage.

7.6.1 Re-bidding after agent removal or completion

Activity diagram showing the process of sector bidding after agents have left swarm or have finished searching all their sectors

7.6.2 Sector Release from Degraded Agents

When an agent becomes degraded (e.g. bad_sensor state):

1) The agent’s FMU triggers a health event.
2) The agent releases its sector and starts a Rumour Mill event with an Origin describing its degraded state.
3) Neighbours of the agent interpret it and broadcast Sources confirming the degraded state

4) Any agent (including neighbours) that receives three Sources updates A's sectors to unassigned, and creates an Opinion rumour (e.g. "A should become relay").
5) The agent collects Opinion rumours; Once Opinion rumours from$\geq \alpha$ fraction of agents have arrived, the agent internally decides to switch to secondary_task (e.g. relay).
6) The agent broadcasts a decision-Origin announcing its new role.
7) The Sector Allocation Market (Section 7) reassigns A's sectors to other agents through bidding.

7.6.3 Sector Release from Failed or Missing Agents

If an agent A fails or disappears (e.g. pulse timeout):

1) Neighbours detect missing pulse and create Sources: "Agent A is no longer a member of swarm."
2) All agents, after receiving three sources, vote locally on whether Agent A should be considered a member or not.
3) If considered not a member, all agent A's sectors are treated as released.
4) SectorInfo entries for sectors formerly owned by A are updated to ownerID = 0.

Again, these sectors become candidates for new bids by healthy agents.

7.6.4 Re-bidding on Released Sectors

Agents monitor SectorInfo:

- When a sector's ownerID becomes 0, it is considered available.
- Agents with sufficient budget and good cost for that sector may initiate new bids using the process in Section 7.5.
- Over time, released sectors are redistributed to agents best able to search them.

This provides automatic replanning: the swarm reorganises its workload in response to agent degradation and failure without central control.

7.7 Failed Auctions & Recovery

Broadcast communication is lossy; messages may be dropped or delayed. As a result, a bidding agent might not receive enough Opinions within a reasonable time, or some agents may temporarily disagree about whether a bid is still active. To handle this, simple recovery strategies can be used:

- Timeout and retry – if the bidder does not receive enough Opinion rumours within a timeout, it may rebroadcast the bid Origin rumour.
- Backoff – after repeated failures, the agent may de-prioritise bidding for that sector and focus on others.
- Problem-sector flagging – sectors that repeatedly fail auctions can be flagged for special handling or manual inspection (future work).

These strategies do not change the conceptual design of the Sector Allocation Market, but are important for a robust implementation. The exact policies for handling such cases can be tuned in simulation and are considered an area for future refinement.

7.8 Summary of Market Properties

The Sector Allocation Market provides the following properties:

- Dynamic workload distribution: Sectors are continuously reassigned as agents join, degrade, fail, or complete tasks.
- Priority to important sectors: High-probability sectors carry higher values and tend to attract more bids.
- Energy awareness: Budgets and distance-based costs encourage agents to take on work they can realistically perform.
- Consistency: All ownership changes are confirmed via the Rumour Mill, avoiding double assignment.
- Decentralisation: No central scheduler is required; agents make local bidding decisions.
- Implementation simplicity: The mechanism uses simple arithmetic and broadcast messages, making it suitable for resource-constrained UAVs.

The output of this Sector Allocation Market—who owns which sectors—feeds directly into the Agent Task & Role Allocation layer (Section 8), which decides how each agent behaves based on its current responsibilities and health state.

8 Agent Task & Role Allocation (Behaviour Layer)

The Agent Task & Role Allocation layer determines what each agent actually does at any given time, based on which sectors it currently owns (from the Sector Allocation Market, Section 7), its health status (from the FMU), and high-level commands or mission status (via the Rumour Mill and SwarmInterface).

This layer is implemented as a state machine inside each agent. While the Sector Allocation Market answers ”who owns which sectors?”, the Task & Role layer answers ”what should this agent be doing right now?”.

8.1 From Sector Ownership to Tasks

For a healthy agent, its primary tasks are derived from the set of sectors it owns:

- Each owned sector S corresponds to a search task: fly to S, search S, and report completion.
- Tasks can be scheduled internally in some order (e.g. nearest-first or probability-weighted order).
- When a sector is fully searched, the agent:
  1) Broadcasts an Origin rumour indicating ”sector S completed”,
  2) participates in the Rumour Mill to confirm completion,
  3) updates its local SectorInfo (status = completed),
  4) receives the budget refund for S when the completion decision is confirmed (Section 7.3).

Thus, sector ownership drives the agent’s task queue. The Task & Role layer connects: SectorInfo to a set of search tasks, health and decisions to whether the agent should continue searching, release sectors, or take on a new role.

After completing or releasing sectors, the agent can use the Sector Allocation Market (Section 7) to bid for new sectors, creating new tasks.

8.2 Roles & Task Types

Agents may operate in different roles depending on their health status, sector ownership, and mission context. Roles are implemented as combinations of state machine states and allowed behaviours.

8.2.1 Primary Roles

Searcher

- The agent’s main role is to search assigned sectors.
- It selects a sector from its task queue, navigates there, and executes the search.
- It reports sector completion via the Rumour Mill and moves on to the next task.

Subject confirmer

- When an agent finds the Subject, it raises a ”Subject found” event (Origin).
- It participates in the subsequent Rumour Mill cycle to confirm or reject the detection.
  (Receiving neighbours go to the location and confirm the subject is there before broadcasting sources about subject found.)
- Once the Subject is confirmed found, its role typically transitions to returning to base.

8.2.2 Secondary Roles (for Degraded Agents)

Communication relay

- The agent positions itself to improve communication coverage between other agents and/or the SwarmInterface.
- It may hover in a location that bridges otherwise disconnected parts of the swarm.
- It continues to send and receive pulses and rumours, but does not perform primary search.

Secondary searcher

- If the agent is partially degraded but still somewhat usable, it may be assigned less critical search tasks (e.g. low-probability sectors).
- The policies for when to allow this are design choices and can be tuned in future work.

8.2.3 Terminal or Restricted Roles

Return

- The agent returns to base, typically due to low battery, mission completion, or explicit command.
- Before returning, it releases its sectors via the Sector Allocation Market, so that other agents can take over.

Failed

- The agent has suffered a critical failure or emergency landing and is effectively out of service.
- It is removed as a member of the swarm after a Rumour Mill decision based on missing pulses or explicit failure events.
- Its sectors are released and reallocated to other agents.

These roles govern which actions are allowed for the agent and how it contributes to the swarm. The exact mapping from health states to roles is configurable, but the protocol provides the mechanisms to implement these mappings in a decentralised way.

8.3 Role Selection Logic

Role selection is driven by three main inputs:

- Health status from the FMU (e.g. healthy, degraded, bad_sensor, failed),
- Sector ownership (how many sectors the agent owns and their status),
- External or global decisions (e.g. Search Manager commands, Subject confirmed found, swarm-wide abort).

The IRDS protocol module provides the mechanisms (Rumour Mill and Sector Allocation Market) to coordinate these inputs; specific role policies can be tuned, but typical examples include:

- Bad Sensor $\rightarrow$ Secondary Role + Sector Release
  1) FMU detects a bad sensor condition.
  2) Agent releases its sectors and raises an Origin: ”I have bad_sensor.”
  3) Neighbours broadcast Sources confirming degradation.
  4) Opinions form; once the consensus threshold is met, the agent decides to transition to a secondary role (for example, communication relay).
  5) A decision-Origin is broadcast to announce this outcome.
     Result: The degraded agent stops performing primary search but remains useful to the swarm.
- Low Battery $\rightarrow$ Return + Sector Release
  1) FMU detects that battery is below a safe threshold.
  2) Agent starts a Rumour Mill event indicating low battery and its intention to return.
  3) Sources confirm that returning is appropriate.
  4) The agent releases its sectors and transitions to the return role.
  5) The SwarmInterface can display this event so the Search Manager can prepare a battery change.
     Result: The agent returns safely without leaving unfinished sectors ”dangling”.
- Subject Found $\rightarrow$ Swarm-Wide Role Change
  1) An agent finds the Subject and broadcasts an Origin: ”Subject found at location L.”
     Neighbours go to the location and confirm the subject is there before broadcasting sources about subject found.
  2) The Rumour Mill confirms or rejects this event.
  3) All agents receive this decision and transition from search or task roles to return.
     Result: The entire swarm reacts consistently when the mission objective has been achieved.

These examples show that role decisions are local (each agent runs its own state machine) but are driven by global information provided by the Rumour Mill and Sector Allocation Market.

Future work may extend this role selection logic by including neighbourhood suggestions, for example, to choose optimal relay positions for degraded agents.

8.4 Agent State Machine

The Agent Task & Role Allocation layer is implemented as a state machine. Each agent transitions between states based on health events, sector allocation, and Rumour Mill decisions.

Each agent has the following key states:

- start
  - Agent inactive
  - Transitions to task upon initialisation or joining
- task
  - Receives assigned sectors
  - Participates in bidding
  - Waits for rumour confirmation

  task transitions to:
  - search when a sector becomes assigned
  - secondary_task if degraded
  - return if failed or commanded by Search Manager
  - failed if agent suffers critical failure
- search
  - Actively searches assigned sector(s)
  - Broadcasts completion via Origin
  - Updates position, health, and pulses

  search transitions to:
  - task when search is completed
  - found if Subject is detected
  - bad_sensor if FMU detects sensor degradation
  - return if commanded
- found
  - Occurs when the agent detects the Subject
  - Broadcasts Origin: ”Subject found”

  found transitions to:
  - confirmed when receiving sources from neighbours
- confirmed
  - All agents move to return state
  - Mission ends or transitions to rescue operations
- return
  - Agent returns to base
  - Human operator may prepare battery replacement
  - Full swarm may return (e.g., mission abort or confirmed subject found)
- bad_sensor
  - Triggered by FMU health output
  - Agent broadcasts degraded health
  - Drops sectors via rumour process

  bad_sensor transitions to:
  - secondary_task
  - or failed
- secondary_task
  Agent performs a non-critical role, such as:
  - Communication relay.
  - Secondary search.

secondary_search transitions to:
    - return if battery low
    - failed if further degradation occurs
- failed
    - Critical degradation or emergency landing
    - Agent informs neighbours (if possible)
    - Leaves swarm
    - Neighbourhoods recalculate
    - Sectors re-enter auction pool

Transitions between these states are triggered by:

- FMU health outputs (e.g. bad_sensor, low battery, failure),
- Rumour Mill decisions (e.g. Subject confirmed found, sector completion, global abort),
- Sector Allocation Market events (e.g. no more sectors owned, sectors reallocated),
- high-level commands via the SwarmInterface.

8.4.1 State Diagram

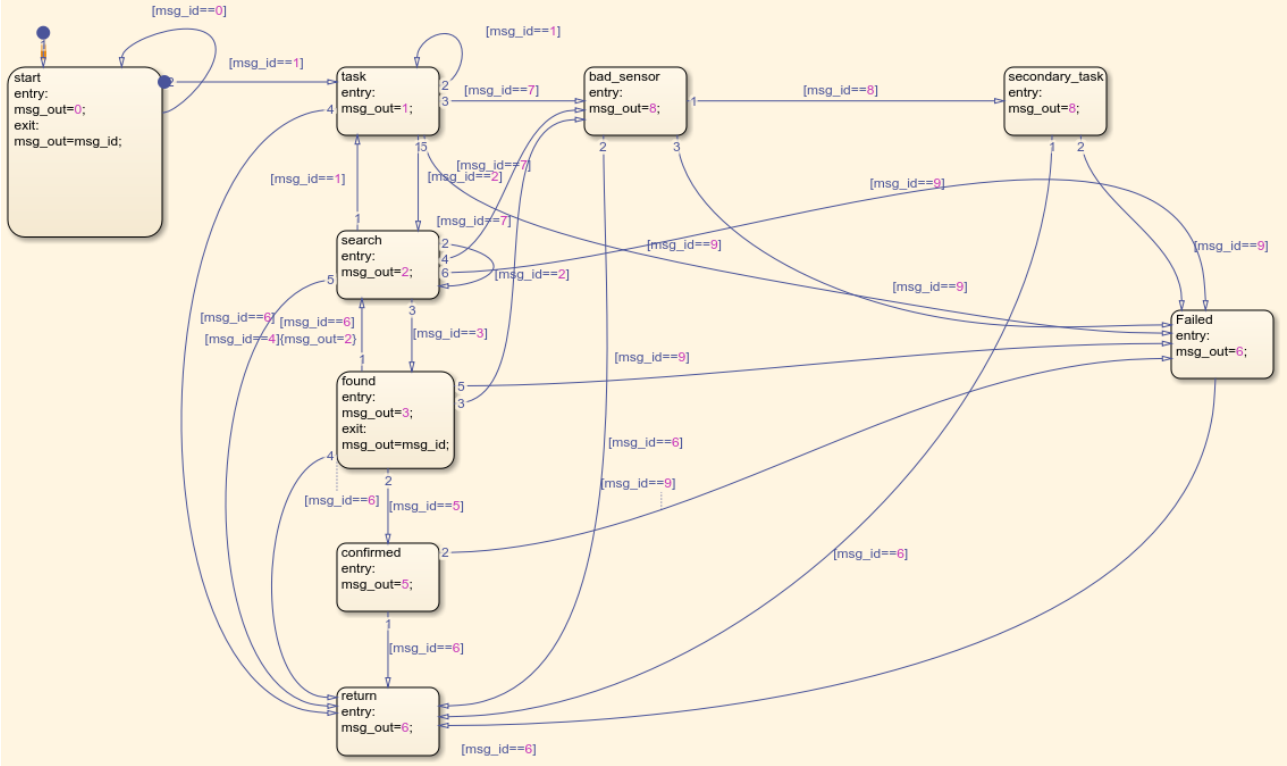

Figure 21: *SD-18* Stateflow of an individual agent.

8.4.2 Activity model: moving to the next sector

Activity diagram showing the process of an agent moving on to its next sector when it has finished searching a sector. Will include broadcast of "sector X finished"

8.5 Swarm-Level Behaviour from Local Roles

Although each agent runs its own state machine and makes decisions locally, the combination of globally consistent sector ownership (Section 7), globally consistent event interpretation (Section 6), and consistent role selection rules (this section) leads to coherent swarm-level behaviour.

Examples of emergent behaviours include:

- Collective Replanning:
  - When one agent degrades or fails, its sectors are released and reallocated.
  - Healthy agents take over these sectors by bidding in the Sector Allocation Market.
  - The swarm automatically reshapes its search pattern to compensate for the loss.
- Coherent Mission Termination:
  - When the Subject is confirmed found, Source rumours are broadcast.
  - All agents update their state machines to transition to return (or other appropriate end-of-mission roles).
  - The swarm thus terminates the search phase in a coordinated way without conflicting behaviours.
- Self-Healing After Failures
  - Missing pulses cause suspected failures to be raised via the Rumour Mill.
  - Once a failure is confirmed, the failed agent's sectors are released and reallocated.
  - Neighbourhoods are recomputed to maintain the 1 + 3 structure where possible.
  - Information about the failed agent (e.g. role, sectors, health) is cleaned up in AgentInfo.

As a result, the swarm can heal from local failures and continue its mission using only local state machines and broadcast rumour messages. This shows how the system-level properties described in the safety and reliability section are realised through local behaviours.

8.6 Safety-Oriented Role Restrictions

Role allocation is also used to enforce safety constraints, for example:

- Agents in bad_sensor or other degraded states cannot perform primary search of critical sectors.
- Agents in failed state must not rejoin the swarm without a proper registration sequence.
- Agents in return should not acquire new sectors until after maintenance and rejoin, if at all.

These policy-level restrictions, combined with the state machine, ensure that degraded or failed agents do not perform unsafe or misleading actions. Their impact on the mission is limited to safe behaviours (e.g. relay) or removal from the swarm.

Further safety analysis of these behaviours is provided in Section 9.

9 Safety & Reliability

The IRDS protocol module is designed to keep a SAR mission running safely and coherently even when individual agents degrade, fail, or behave unpredictably. This section describes:

- The safety objectives of IRDS
- The fault model and pulse-based failure detection.
- How the agent state machine enforces safe behaviour.
- The formal assumptions behind fault tolerance.
- How decentralisation contributes to mission-level reliability.

9.1 Safety Objectives

The protocol module aims to satisfy the following safety objectives:

- Mission continuity: The swarm must continue searching even if individual agents degrade or fail.
- Consistent decision-making: All non-faulty agents should eventually agree on the interpretation of critical events and the resulting actions.
- Prevention/Avoidance of contradictory actions: Conflicts over sector ownership, search direction, or state transitions must be prevented. The swarm should avoid unsafe conflicts such as two agents believing they own the same sector, or an agent both ”returning” and ”searching” simultaneously.
- Containment of individual faults: A malfunctioning or malicious agent must not compromise the swarm. The influence of a malfunctioning or malicious agent should be localised; the swarm as a whole should remain trustworthy.
- Graceful degradation: Agents must shift to safe secondary tasks or return-home behaviour when necessary. Degraded agents should move into safe secondary roles or return home, rather than silently producing bad data.
- Operator awareness: Critical events must be communicated to the Search Manager in a timely manner. Critical events (e.g. agent returning for battery, agent degraded, Subject found) should be communicated to the Search Manager via the SwarmInterface.

These objectives shape the design of the Rumour Mill, Sector Allocation Market, and Task & Role layers.

9.2 Fault Model

The protocol module assumes the following types of faults may occur:

1) Crash Faults
   The agent stops operating or lands unexpectedly (includes emergency landing), and may stop sending messages entirely.
   Examples: Hardware failure, sudden power loss, motor loss.
2) Omission Faults
   The agent intermittently fails to send pulses, or propagate rumour messages.
   This can be due to radio interference, range limitations, or transient internal issues.
3) Timing Faults
   Messages may be delayed or arrive out of order because of variable communication latency and local processing delays.
   The protocol does not assume strict timing guarantees; it aims for eventual consistency instead of synchronous behaviour.
4) Byzantine Faults
   An agent may behave arbitrarily, including sending incorrect, inconsistent, or misleading information. IRDS is designed to tolerate one Byzantine fault per neighbourhood (details in Section 9.5), provided faulty agents are not too clustered.

The protocol is designed to tolerate one Byzantine fault per neighbourhood and multiple crash or omission faults swarm-wide.

9.3 Pulse Safety Mechanism

Pulses provide a simple, continuous check on agent presence (health status is assumed healthy unless a rumour states otherwise). Pulse messages support safety by enabling:

- Agent presence detection.
- Neighbourhood integrity checks.
- Failure detection.
- Position updates for separation distance and collision avoidance buffers.

Content of a pulse can be seen in Section 5.5.3.

9.3.1 Pulse-Based Failure Detection

If an agent does not receive pulses from any of their neighbours for a configured timeout period, it:

- Suspects that the neighbour, agent A, has failed or is unreachable,
- creates a Source rumour: ”Agent A no longer member of swarm”,
- participates in the Rumour Mill to confirm or reject this Source rumour.
- Once the Rumour Mill confirms that Y is missing or failed:
  - Y is set as no longer a member in AgentInfo,
  - neighbourhoods are recalculated to exclude Y,
  - sectors previously owned by Y are considered released and re-enter the Sector Allocation Market (Section 7.6.3).

Thus, silent failures do not go unnoticed and their impact on search coverage is mitigated.

9.4 Safety-Driven State Behaviour

The agent state machine (Section 8.4) is designed with safety in mind, where key transitions include:

- Health degradation → restricted roles:
  - bad_sensor → transition from search to secondary_task, prevents faulty sensors from corrupting search results.
  - Release of sectors via the Sector Allocation Market.
  - Stop performing primary search tasks.
- Critical failure → failed:
  - Severe FMU errors or complete communication loss → failed.
  - Removal of the agent from neighbourhoods and SectorInfo after Rumour Mill confirmation.
- Low battery → return:
  - Low-battery event → sector release, then return.
  - Prevents agents from dying silently in the field with sectors assigned.
- Subject confirmation → return
  - Subject confirmed found → all agents transition towards return.

These transitions ensure that degraded agents limit their impact, failed agents are excluded from coordination, and resources and responsibilities are reallocated to healthy agents.

9.5 Formal Fault Tolerance Model

This subsection summarises the assumptions and reasoning behind the fault tolerance of the IRDS protocol module with respect to neighbourhoods and consensus.

9.5.1 Neighbourhood Size and Structure

Each agent forms a neighbourhood consisting of itself and its three geographically closest agents (based on initial sectors). Thus, each neighbourhood contains four agents in total (1 + 3). Each agent must be seen as a neighbour by at least three other agents.

Neighbourhoods overlap, meaning that a given agent is the centre of its own neighbourhood, and is one of the three neighbours in three or more other agents’ neighbourhoods.

9.5.2 Rumour Mechanics within a Neighbourhood

For a given event:

- The originator broadcasts an Origin rumour.
- Neighbour agents independently broadcast Source rumours interpreting that Origin rumour.
- Any agent that receives three Source rumours for this Origin rumour can interpret the event through voting, and possibly creates an Opinion.

Within the originator's neighbourhood, there are up to three neighbour Source rumours (from its three neighbours), and potentially additional Source rumours from other agents as rumours propagate. To interpret an event and possibly broadcast an Opinion rumour, an agent needs three Source rumours, and those Source rumours may come from neighbours and/or other agents in the swarm.

9.5.3 Tolerance of One Byzantine Neighbour

Consider a single neighbourhood of 4 agents (A = originator, plus neighbours B, C, D), and assume at most one of these neighbours is Byzantine:

- In the worst case, one neighbour (say B) may send misleading or malicious Source rumours.
- The other two neighbours (C and D) are non-faulty and produce correct Source rumours.
- Any agent forming an Opinion based on three Source rumours will see at least two honest Source rumours out of the three.

Since Opinions are derived from multiple independent Source rumours, a single Byzantine neighbour cannot, by itself, force a consistently incorrect Opinion if the Opinion logic respects the majority of honest Source rumours. Therefore with neighbourhood size 4 (1 + 3), the design can tolerate one Byzantine agent per neighbourhood without corrupting the consensus process, assuming that Opinion formation logic is dominated by honest Source rumours.

9.5.4 Overlapping Neighbourhoods and System-Level Bounds

Because neighbourhoods overlap, a single Byzantine agent appears as a neighbour in several neighbourhoods, but each neighbourhood still contains at most 1 Byzantine agent as long as faults are not clustered.

A key constraint is for each neighbourhood N_i, the number of Byzantine agents in that neighbourhood must be at most 1:

$$\forall N_i : |N_i \cap B| \leq 1$$

where B is the set of Byzantine agents.

If this constraint holds, then each neighbourhood's Opinions are dominated by honest Source rumours, swarm-wide consensus remains trustworthy, and the overall system remains safe, even with multiple Byzantine agents.

Roughly speaking (and ignoring clustering effects), with N agents and neighbourhoods of size 4, the swarm can tolerate up to on the order of N/4 Byzantine agents **provided they are spread out** so that no neighbourhood contains more than one Byzantine agent. This is an approximation, not a strict bound, and a precise analysis is left for future work.

9.5.5 Relationship to Consensus Threshold α

The consensus threshold α (Section 6.6) determines how many Opinions the originator waits for before making a decision.

- Larger $\alpha \rightarrow$ more tolerant to faulty or missing agents, but slower decisions.
- Smaller $\alpha \rightarrow$ faster decisions, but less robust if many agents are faulty or disconnected.

In this design, α is treated as a configurable parameter (70% used in examples). The optimal value depends on:

- Expected number and distribution of faulty agents.
- Communication reliability.
- Mission time constraints.

9.6 Prevention of Contradictory Actions

The combination of the Rumour Mill, the Sector Allocation Market, and state machine prevents several classes of contradictory or unsafe actions:

- Double sector ownership
  - Sector ownership changes are always confirmed through the Rumour Mill.
  - Bids are validated by Source rumours and Opinions before the originator announces a successful purchase.
  - All agents update their SectorInfo based on the decision-Origin, avoiding conflicting owners.
- Inconsistent health interpretations
  - Health changes (e.g. degraded, failed) are rumour events.
  - Agents eventually converge on whether an agent is healthy, degraded, or failed.
  - This avoids half the swarm thinking an agent is healthy while others think it has failed.
- Conflicting roles
  - The state machine defines allowed transitions (e.g. from search to return, not both at once).
  - Rumour Mill decisions (e.g. Subject confirmed, mission aborted) lead to consistent role changes across the swarm.

These mechanisms prevent the most dangerous inconsistencies during mission execution.

9.7 Eventual Consistency Guarantees

The protocol module assumes a lossy, broadcast communication channel, i.e., not every agent receives every message. However:

- Origins, Sources, Opinions, and decision-Origins are broadcast repeatedly over time,
- neighbourhoods overlap, creating multiple paths for information to propagate,
- agents maintain rumour state and can act on messages received later.

Under these conditions:

- all non-faulty agents will eventually receive the decision-Origin for each rumour,
- their local AgentInfo and SectorInfo will converge to the same values,
- their state machines will react consistently.

This eventual consistency is sufficient for SAR missions where real-time, perfectly synchronised decisions are not strictly required, but coherent behaviour over time is.

9.8 Safety Benefits of Decentralisation

The decentralised design of the protocol module offers several safety benefits:

- No single point of failure
  - No central controller whose failure would stop the mission.
  - No special ”leader” agent that, if compromised, could mislead the entire swarm.
- Local containment of faults
  - Faulty behaviour from one agent is limited by neighbourhood cross-checking and the need for multiple Source rumours and Opinions.
  - Degraded agents are moved into secondary or terminal roles.
- Graceful scaling
  - Adding more agents increases redundancy and potential coverage.
  - The same protocol logic applies regardless of swarm size (within design limits).
- Resilience to communication loss
  - Because logic is local and rumour-based, temporary disconnects do not permanently break the system; agents can rejoin and resynchronise through the registration procedure.

9.9 Mission-Level Reliability

At the mission level, the IRDS protocol module improves reliability by:

- Maintaining search coverage
  - When agents fail or degrade, their sectors are released and reallocated via the Sector Allocation Market.
  - The swarm continues searching instead of leaving ”holes” where failed agents were assigned.
- Keeping the operator informed
  - Return requests, degraded states, and mission milestones (e.g. Subject found) are surfaced to the Search Manager via the SwarmInterface.
  - The operator can prepare interventions such as battery changes or mission replans.
- Supporting long-running missions
  - Dynamic join/rejoin allows agents to be swapped in and out (e.g. for battery changes) without restarting the mission.
  - The protocol automatically integrates new agents and rebalances the workload.

Overall, the combination of Rumour Mill consensus, market-based sector allocation, and safety-aware role allocation results in a swarm that can tolerate a range of faults and still perform its SAR mission in a controlled and predictable manner.

10 Future Work

This section outlines future work to refine and implement the IRDS conceptual architecture, protocol specification, and behavioural model for a decentralised, fail-operational UAV swarm in SAR missions.

10.1 Protocol Implementation

The Rumour Mill, Sector Allocation Market, and Task & Role Allocation are currently specified conceptually and only partially explored in simulation. Implementing them will require:

- Translating the protocol logic into an embedded language (e.g. C/C++) suitable for UAV microcontrollers.
- Defining concrete message formats and encodings (bit-level layout for msgID, taskID, etc.).
- Implementing Origin/Source/Opinion handling and lifecycle on real hardware.
- Integrating the rumour lifecycle with periodic broadcast scheduling and pulse transmission.
- Implementing local storage for AgentInfo, SectorInfo, etc. within memory constraints.

Implementation work should also address:

- Handling message drops, duplicates, and stale rumours.
- Setting practical limits on the number of active rumours and stored Sources/Opinions.
- Memory- and CPU-efficient data structures for onboard use.

Fault injection and stress testing in simulation and on hardware will be important to verify real-time behaviour, robustness, and scalability.

10.2 SwarmInterface Development

In the current design, the SwarmInterface is a conceptual component. Future work should develop a concrete implementation, including:

- A graphical map-based interface for the Search Manager.
- Tools to draw Hot Regions and visualise sector boundaries and probabilities.
- Controls for sending high-level commands (start, pause, abort, area updates).
- Visualisation of swarm status:
  - Agent positions and health statuses.
  - Which sectors are assigned / in progress / completed.
  - Agents returning for battery change.
  - Subject found / confirmed events.
- Event logging, playback, and debugging tools.

Hardware requirements may include a laptop or tablet, and a ground radio (or equivalent) for broadcast communication with the swarm.

Designing this interface with usability in mind will make the system much more practical for real SAR operators.

10.3 Pathing & Navigation Integration

The protocol module currently assumes that agents can move between sectors when requested. Integration with navigation and path planning deserves further development:

- Computing efficient routes to visit multiple owned sectors (e.g. nearest-first, probability-weighted).
- Integrating with buffer-zone enforcement, and collision avoidance and obstacle detection subsystems.
- Dynamic replanning of paths when other agents move, new obstacles are detected, or sectors change ownership.
- Ensuring that role changes (e.g. becoming a relay) translate into stable, safe positions in space.

This may involve defining a clear interface between the Task & Role layer and the navigation stack, and experimenting with different path-planning strategies and measuring their impact on mission time and safety.

10.4 Expanded Secondary Tasks

Currently, the protocol module defines limited secondary tasks for degraded agents, such as communication relay and secondary search.

Future enhancements may include:

- Communication relay positioning:
  Selecting optimal relay positions based on neighbourhood geometry or sector layout.
- Search assistance:
  Using partially degraded agents to search sectors owned by healthy agents.

Additionally, neighbourhoods may vote (via independent Sources) on where a degraded agent should position itself as a communication relay, improving cooperative performance.

Task prioritisation rules may also be expanded to consider mission-critical objectives.

10.5 Security

Security is not implemented in the conceptual design, but future work should include an appropriate encryption scheme appropriate for UAVs.

Suggested starting point:

- AES-128 encryption with a shared operation-specific 16-byte key.
- Key derived via a UUIDv4 random generator for each mission.
  This is intended to ensure that only authorised participants can interpret messages and to provide basic protection against eavesdropping and simple spoofing.

Additional future enhancements may include:

- Replay protection to prevent old rumours from being re-injected.
- Secure registration for join/rejoin procedures.

Initial work could focus on low-overhead schemes suitable for microcontrollers and low-bandwidth radio links. More advanced work could study detection of malicious agents that send inconsistent Source rumours or Opinion rumours, and integration of security events into the fault model and Rumour Mill.

10.6 Scalability & Optimisation

The current design targets swarms of roughly 4–255 agents. Future work can investigate:

- How performance (message load, decision latency, convergence time) scales with swarm size.
- Alternative neighbourhood selection strategies (e.g. dynamic neighbourhood size, non-geographic metrics).
- Adaptive rebroadcast strategies to reduce redundant traffic while maintaining robustness.
- Optimisation of rumour timeouts and retry strategies.
- Techniques for prioritising critical rumours over less important ones.

Simulation with larger swarms (e.g. hundreds of agents) can help identify scaling bottlenecks and guide protocol optimisations.

10.7 Formal Verification & Validation

Once implemented (for future dependability or safety case work), the IRDS protocol module should undergo:

- Formal modelling using state machines or model-checking frameworks.
- Validation against SAR requirements and safety objectives.
- Verification of rumour convergence.
- Stress-testing under high fault loads.
- Analysis of worst-case message overhead.
- Verification that no inconsistent states can arise.
- Analysis of the impact of different consensus thresholds α under various fault scenarios.
- Verification that, under specified assumptions, the protocol tolerates one Byzantine fault per neighbourhood.

10.8 Physical Deployment (Long-Term)

Eventually, the protocol module may be tested on real UAVs. This will require:

- Hardware integration of the protocol module.
- Adapting communication models to real radio frequency hardware.
- Accounting for UAV delays, vibration, and sensor noise.
- Ensuring real-time constraints are met.
- Integration with flight controllers.
- Safety testing in controlled flight zones.

10.9 Summary

The IRDS protocol, as described in this document, provides a decentralised Rumour Mill for consensus, a Market for sector allocation, and a Task & Role state machine for agent behaviour. Future work can extend this foundation by:

- Further development of the conceptual designs.
- Implementing and optimising the protocol on real hardware.
- Building a practical SwarmInterface for SAR operators.
- Expansion of secondary tasks.
- Integrating advanced path planning and secondary tasks.
- Adding cryptographic security.
- Scaling and formally verifying the design.
- Eventually deploying and testing the system in real-world scenarios.

These directions provide a roadmap for future project groups and research efforts to evolve IRDS from a conceptual design into a robust, field-ready swarm system.

References

[1] L. Giacomissi, Z. Yigit, M. Shakarna, S. Saleemi, I. Tomasic, and H. Forsberg, “Design of a fail-operational swarm of drones for search and rescue missions,” *Not Published*, 2025.

[2] C. Namatovu, *Requirements Management Plan*, Intelligent Replanning Drone Swarm, Oct. 4 2025, Version 1.0.

Appendix

This appendix provides additional material to support the IRDS system design.

UAV Hardware

A BDD of the hardware inside a UAV was made to easier understand the design of the protocol. The BDD is based on the design found in [1] with alterations to titles and the split of the one unit for Object Detection & Collision Avoidance in to two units, as seen table 2.

Hardware Components Overview

The UAV architecture consists of the following major subsystems:

Title in [1]	**Abbr. in [1]**	**Title in fig. 22**	**Abbr. in fig. 22**
Fault Management System	FMS	Fault Management Unit	FMU
Flight Control System	FCS	Flight Control Unit	FCU
Navigation System	NAV	Navigation Unit	NAV
Motor Control	MC	Motor Control Unit	MCU
Power Management	PM	Power Management Unit	PMU
Swarm Coordination Module	SCM	Swarm Coordination Unit	SCU
Object Detection & Collision Avoidance	OD-&CAS	Object Detection Unit	ODU
		Collision Avoidance Unit	CAU

Table 2: Original titles and abbreviations in [1] compared to new titles and abbreviations in fig. 22.

These subsystems interact with various sensors, actuators, and communication modules to execute flight, navigation, and safety-critical functions.

UAV Hardware BDD

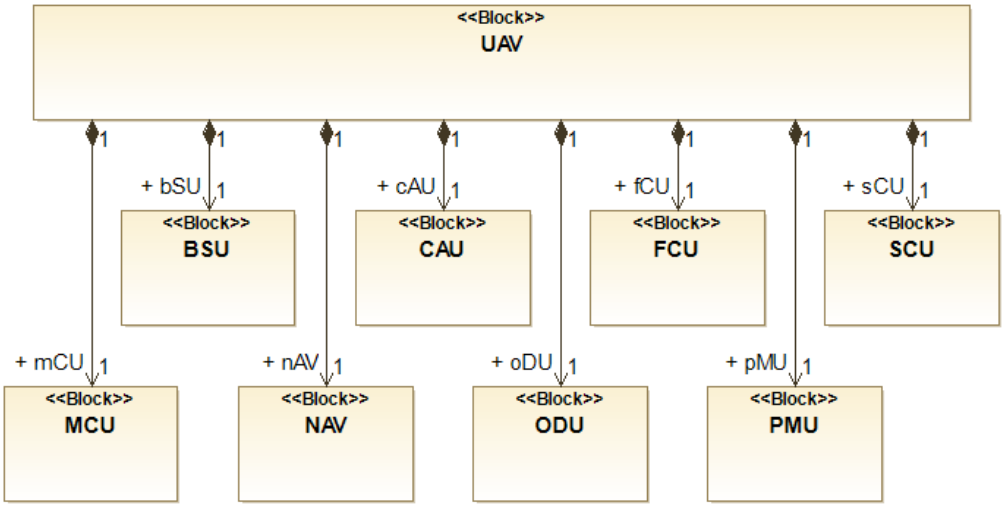

Figure 22: *(SD-02)* BDD of main hardware parts of a UAV, based on design found in [1].

This diagram shows:

- physical UAV components
- internal software modules
- their relationships and dependencies
- their structural boundaries relative to the IRDS protocol

The SCU (Swarm Coordination Unit) is the module that executes the IRDS protocol, interacting with the FMU and external communication interfaces.