

Hybrid Communication Infrastructure: CubeSat Space Protocol and Space ROS Integration

Lab: Ω-Space Group, ORION Lab, National Technical University of Athens

Duration: 6 months

Expected Start: January 2026

Background

Reliable inter-subsystem communication is critical for CubeSat operations. This thesis focuses on implementing a hybrid communication infrastructure for the FlatSat testbed that combines proven satellite protocols (CubeSat Space Protocol over CAN bus) with high-bandwidth data paths (Gigabit Ethernet) for image and large data transfer.

This approach, inspired by JAXA's RACS [1] initiative, prioritizes flight-proven CubeSat protocols for the satellite bus control messaging (C&DH ↔ Payload commands/telemetry) while implementing Gigabit Ethernet for high-bandwidth data transfer (images, AI results, large data products). The payload subsystem uses SpaceROS [2] internally for its AI processing pipeline. This dual-layer architecture demonstrates the practical integration of traditional satellite protocols with modern high-bandwidth requirements for AI and imaging payloads.

Objectives

Develop a hybrid communication infrastructure that:

1. Implements CubeSat Space Protocol (CSP) over CAN bus for control messaging
2. Enables reliable command/telemetry exchange between C&DH and Payload subsystem
3. Implements CSP addressing, routing and packet management
4. Implements GigE interface for high-bandwidth data transfer (images, large AI results)
5. Supports payload's internal use of Space ROS (ROS2 DDS) for AI processing
6. Provides monitoring, debugging and diagnostic capabilities for both networks

Scope of Work

Phase 1: CSP/CAN Network Implementation

Physical Layer Setup

- Configure CAN bus interfaces on Raspberry Pi 4 (C&DH) and Jetson (Payload)
- CAN bus configuration with error handling
- Test physical connectivity with CAN tools (e.g. candump, cansend)

CSP Protocol Implementation

- Deploy libcsp (CubeSat Space Protocol library) on all modules
- Configure CSP addressing scheme
- Implement CSP over CAN driver
- Configure CSP routing tables
- Implement CSP services: ping, connection handling, buffer management

Message Type Definitions

Define standard message types for inter-subsystem communication (following CubeSat conventions)

- Telecommands: C&DH → Payload (mode changes, configuration, control)
- Telemetry: Payload → C&DH (housekeeping, status, events, small AI result summaries)
- Time synchronization: C&DH → all subsystems
- Ping/health check: bidirectional between all subsystems

C&DH Integration (Primary)

- Implement CSP interface in C&DH software
- Command routing: C&DH sends commands to Payload via CSP
- Telemetry collection: C&DH receives telemetry from Payload
- End-to-end validation: cmd injection → CSP transport → execution → telemetry return
- Error handling and retry logic

Payload Integration - CSP Interface

- Implement CSP interface module on Jetson
- CSP interface receives cmds from C&DH: START_APP, STOP_APP, REQUEST_STATUS, etc
- CSP interface sends telemetry to C&DH: status, resource usage, small result summaries
Note: Payload uses ROS2 DDS internally (camera → AI nodes)
- Test C&DH ↔ Payload communication via CAN/CSP

Phase 2: Gigabit Ethernet High-Bandwidth Data Path

Requirements & Design

Implement DDS-layer communication drawing inspiration from JAXA RACS Extended DDS pattern

- Purpose: Transfer images, processed images, detailed AI results, large data products
- Direction: Primarily Payload → C&DH (for eventual ground downlink via comms subsystem)
- Approach: DDS (Data Distribution Service) over Gigabit Ethernet
- Reference: JAXA RACS2 Extended DDS (github.com/jaxa/racs2_extended-dds)
- C&DH acts as minimal DDS subscriber (lightweight DDS client)
- Payload publishes to DDS topics (native from ROS2 nodes)
- Direct DDS-layer communication without ROS2 dependency on C&DH

DDS Implementation

- Select DDS implementation: e.g. CycloneDDS or FastDDS
- C&DH implements minimal DDS subscriber using DDS C or Python API
- C&DH does NOT need full ROS2 - just DDS library for subscribing
- Payload publishes data from ROS2 nodes to DDS topics
- Both sides use same DDS implementation for compatibility

Performance Analysis

- CSP/CAN performance: latency, throughput (commands/telemetry)
- GigE performance: bandwidth, latency (images/large data)
- Resource overhead: CPU/memory usage of both communication stacks
- Comparison: When to use CSP/CAN vs GigE based on data size
- Reliability metrics for both paths
- Bottleneck analysis and optimization recommendations

Documentation

- Network architecture diagrams (physical and logical)
- CSP addressing and routing documentation
- GigE protocol specification and message formats
- Integration guide for both communication layers
- Troubleshooting guide and common issues
- Performance benchmarks and recommendations
- Guidelines for choosing CSP/CAN vs GigE based on use case

Technical Requirements

- Operating Systems: Ubuntu 22.04 on Raspberry Pi 4 and Jetson, FreeRTOS on STM32
- Primary Protocol: CubeSat Space Protocol (CSP) over CAN bus for control messaging
- Secondary Protocol: ROS2 DDS over Gigabit Ethernet for high-bandwidth data
- Programming Languages: C/C++, Python
- CAN Bitrate: 250 kbps (TBD)
- Ethernet: Gigabit Ethernet
- Libraries: libcsp, python-can, socket libraries, ROS2 client libraries (TBD)

Deliverables

1. Working CSP/CAN network connecting C&DH and Payload subsystems
2. CSP implementation with addressing, routing and services
3. Working Gigabit Ethernet high-bandwidth data path (Payload → C&DH)
4. Full integration with C&DH subsystem (Thesis 1) - both CSP and GigE interfaces
5. Integration with Payload subsystem (Thesis 2) - both CSP and GigE interfaces
6. Basic network monitoring and diagnostic tools for both communication layers
7. Performance analysis for both CSP/CAN and GigE (latency, throughput, reliability)
8. Network architecture documentation and integration guides
9. Source code with documentation in GitHub repository

Technical Skills

- Python and C/C++, Cython
- Network protocols and communication systems
- CAN bus
- Embedded systems and real-time constraints
- Linux networking
- ROS2 / SpaceRos
- Space Systems and CubeSat protocols

Management

- Weekly coordination meetings ensure project tracking and knowledge sharing.
- All work is documented and shared via GitHub, following open-source guidelines and contributing to the open-source community.
- Master thesis can overlap with other theses and/or ongoing developments in the working group. Collaboration is encouraged but all topics have been designed to not introduce blocking points.

Contact

- Simon Vellas: svellas@mail.ntua.gr
- Alexis Apostolakis: alexis.apostolakis@gmail.com
- Giorgos Athanasiou: georgios.athanasiou.ntua@gmail.gr