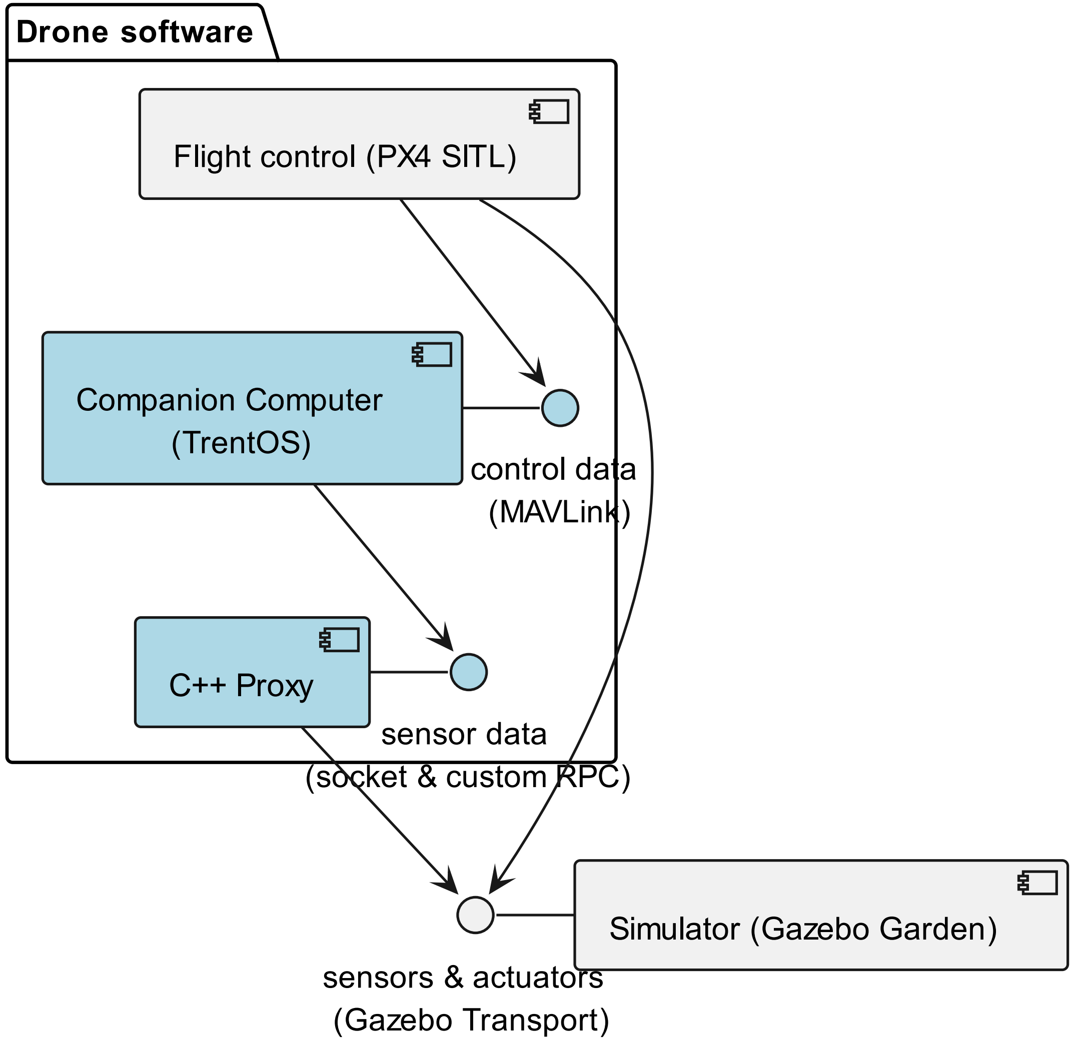
Operating Systems - seL4 & TRENTOS

Drone Simulator

Project Overview

System Architecture

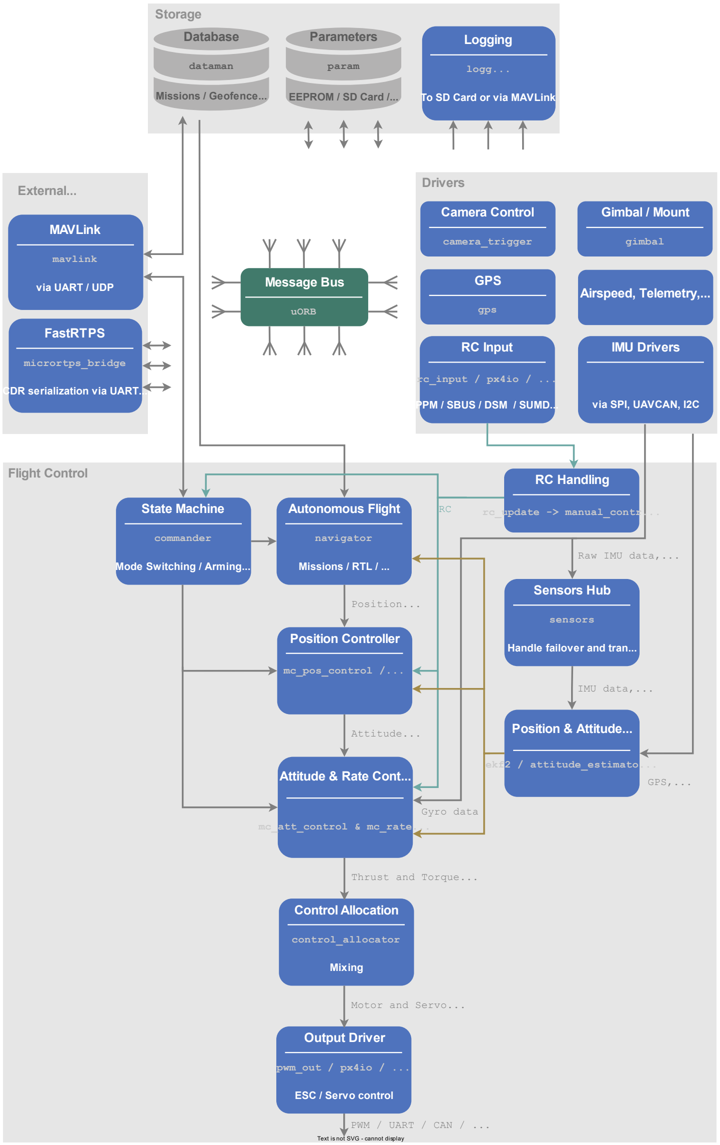


- **Gazebe Garden (Simulator)**
- **Flight control app (PX4)**
- **C++ Proxy**
- **TrentOS-based Companion Computer**

Our goal:

Implement flight task on CompanionComputer (TrentOS) that sends actuators MAVLink messages to PX4 in order to fly the simulated drone to a predefined destination guided by GPS and altitude sensor data from Gazebo.

PX4 SITL (flight control app)



- **Q: What is PX4?**
- **"The brain of a drone"**
- **Architecture: *concurrent* modules that communicate *asynchronously* via uORB message bus**
- **Communication with external world through MAVLink**
- **Q: How to communicate?**
- **A: [PX4 startup] and [predefined MAVLink channels]**
- **Q: Semantics of communication: what to communicate?**
- **A: Flight modes, among which offboard mode. Standard MAVLink messages More details later :)**

Patches to PX4

- Already has everything we need???
- No GPS and altitude sensor on default drone model from PX4 for Gazebo
- Models are specified using a XML-based format called SDF format.
- Patch PX4-Autopilot repository
- Custom drone model
- Custom world mode
- Custom (minimal) MAVLink message channel

C++ Proxy

Two Major Questions:

- How to get sensor data from Gazebo
- How to communicate with TrentOS

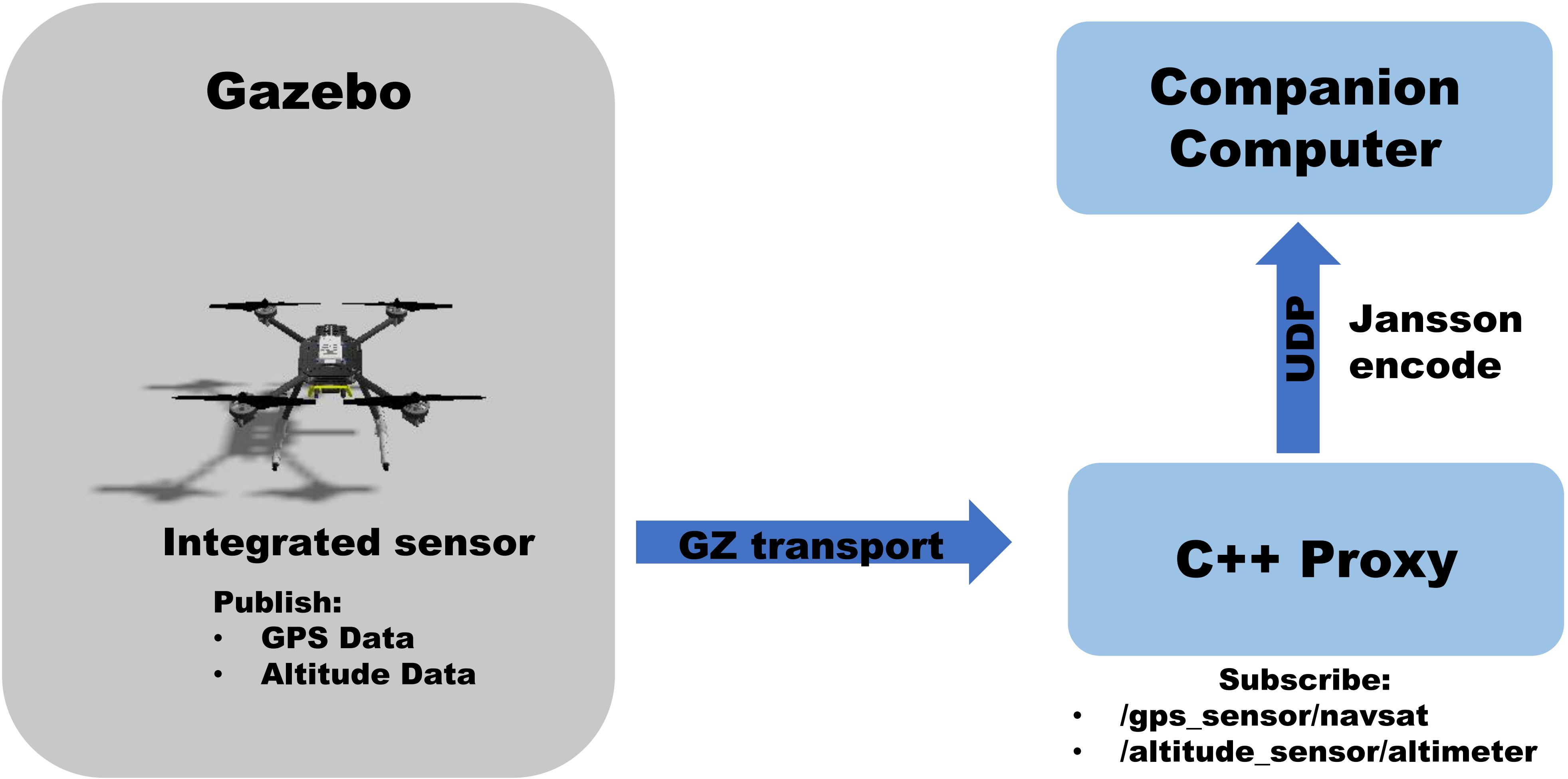
Gazebo

- **3D robotic simulator**
- **Plugin system for modifying models**
- **Pub-Sub system communication**

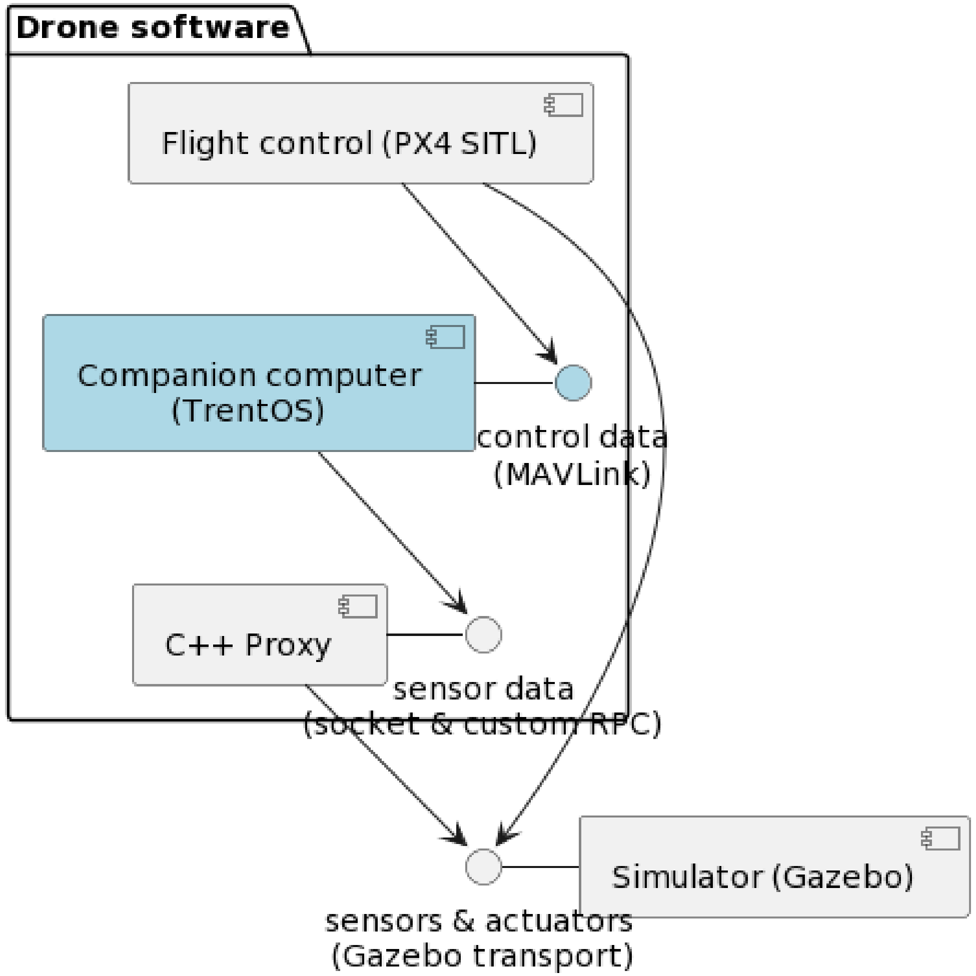
What we need to do?

1. **Sensor Integration**
2. **World modification**

C++ Proxy


Gazebo
Integrated sensor
Publish:
• GPS Data
• Altitude Data
GZ transport
C++ Proxy
Subscribe:
• /gps_sensor/navsat
• /altitude_sensor/altimeter
UDP
Jansson encode
Companion Computer

Companion computer & MAVLink



MAVLINK
MICRO AIR VEHICLE COMMUNICATION PROTOCOL

MAVLink

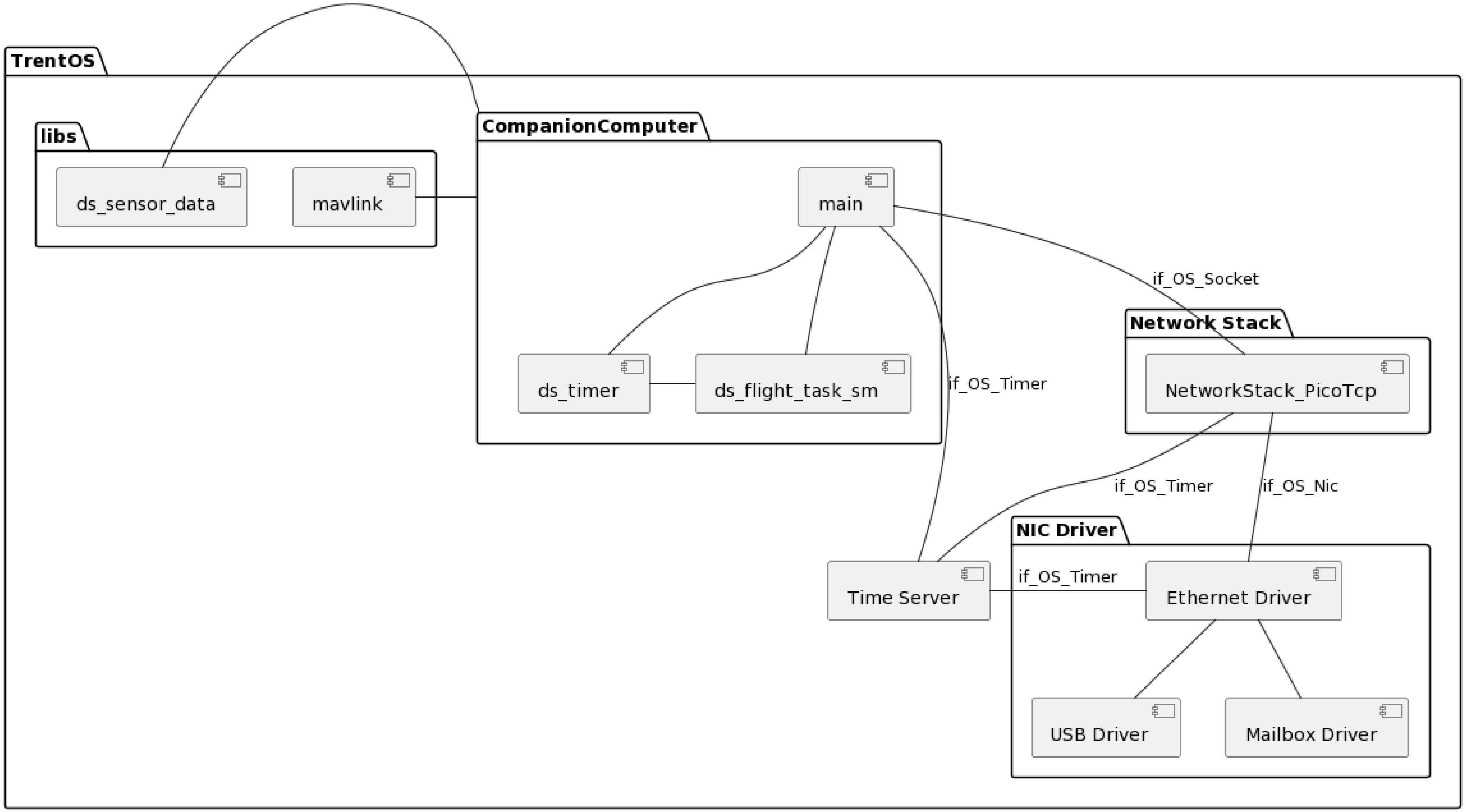
MICRO AIR VEHICLE COMMUNICATION PROTOCOL

- **Binary Telemetry Protocol**
- **Transport Agnostic Library**

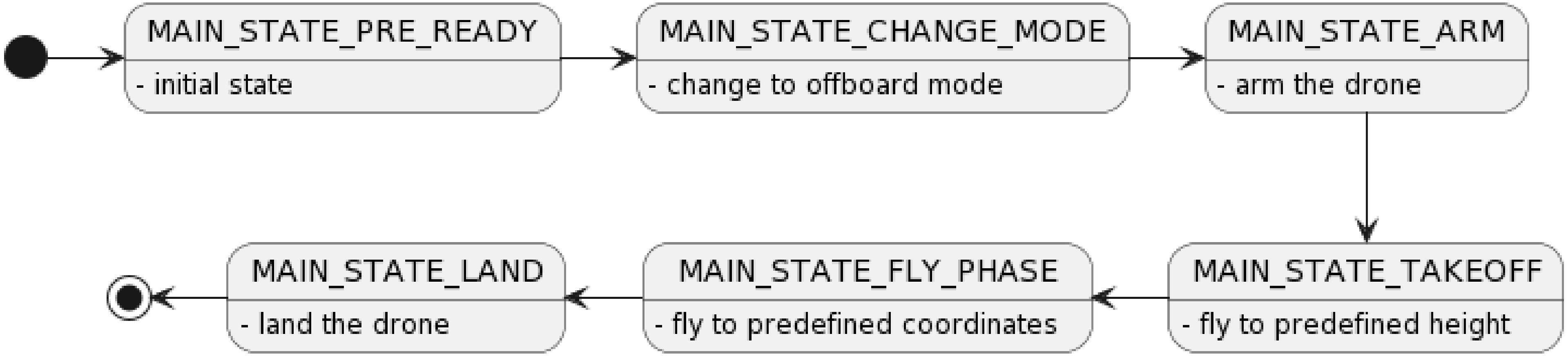
Some important Commands

- **MAV_CMD_DO_SET_MODE**
- **MAV_CMD_COMPONENT_ARM_DISARM**
- **SET_POSITION_TARGET_LOCAL_NED**
- **MAV_CMD_NAV_LAND**

Companion computer

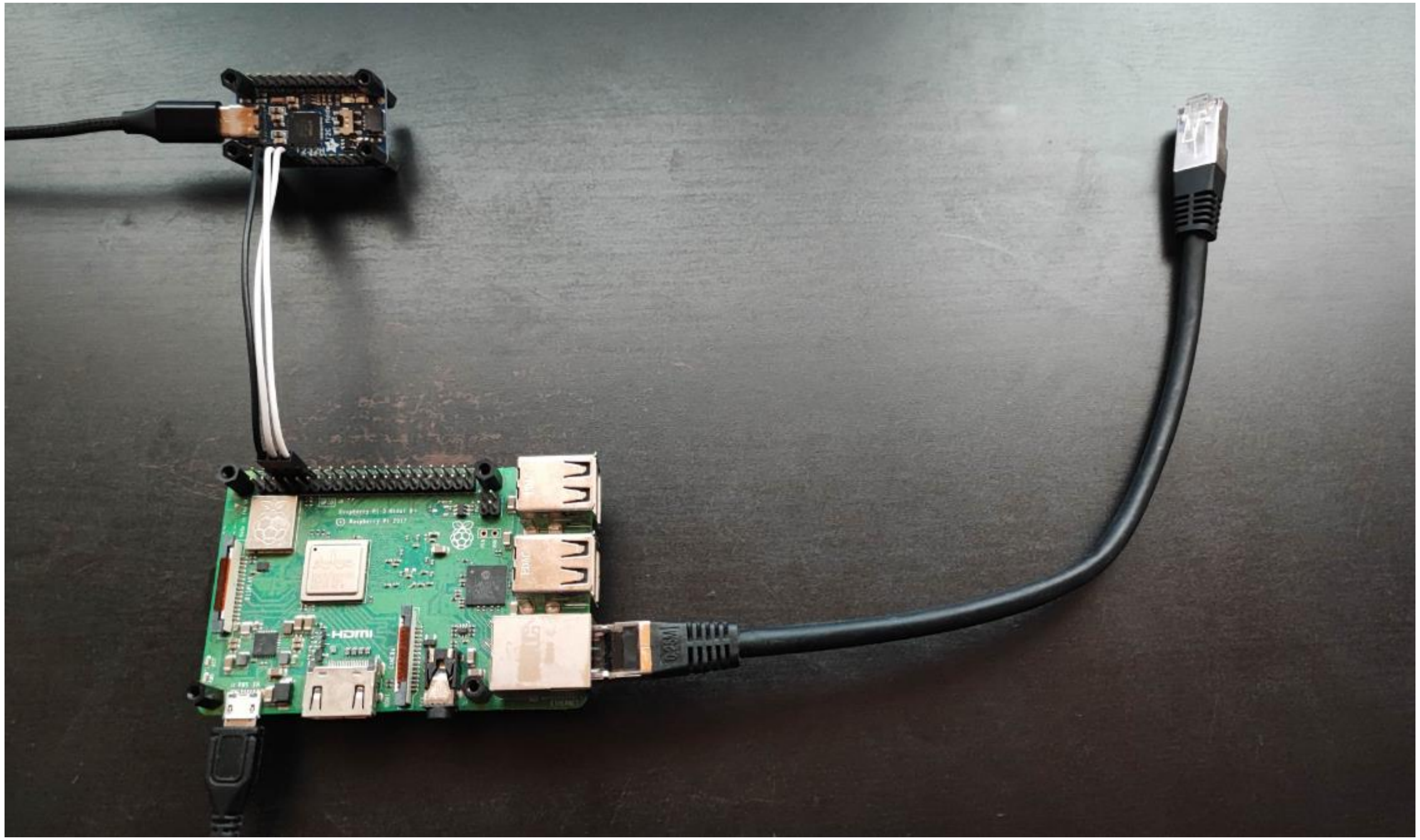


State Machine

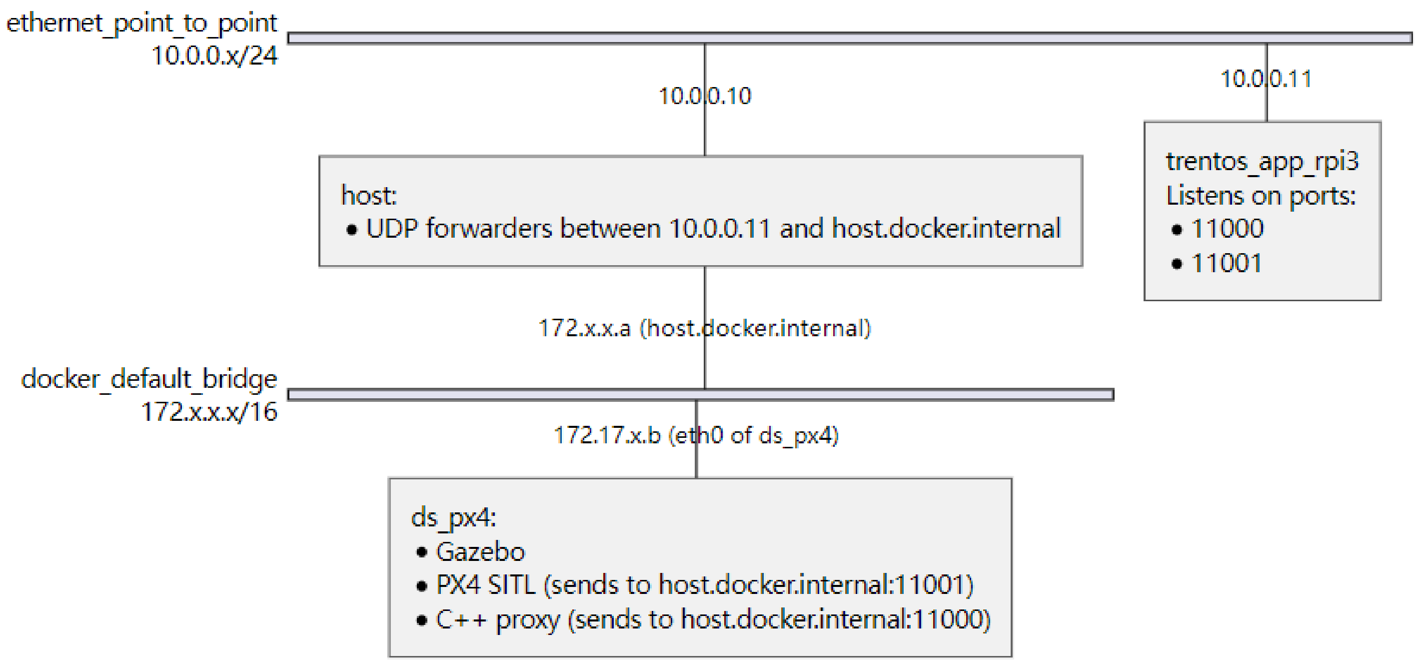


Final Setup

Hardware setup



Network Architecture



- **TrentOS on RPI3**

TrentOS application connected to host PC via Ethernet interface

Project Details

Problems faced

- Jansson (`sys_clock_gettime()` not implemented)
- Docker GUI (disable X11 authentication)
- Firewall on NixOS
- Socket can’t handle large traffic on QEMU
- `OS_Socket_recvfrom()` bug (or feature?): nothing received =>

  `srcAddr` set with last address from which something was received
- Can’t use more complex world (some models take a lot to load)

Ways to improve

UDP forwarding is a hassle

- Better utility scripts that sets up all the components of the system (instead of opening a lot of terminals)
- \`iptables\` in a production setting

Make flight task even more robust

- Handle scenario where PX4 and Gazebo are killed and restarted

Repository structure

```
.
├── Dockerfile          # Dockerfile for the `ds_px4` container
├── external            # Where external dependencies are placed
│   ├── mavlink/        # Pre-compiled MAVLink headers for C/C++
│   └── PX4-Autopilot/  # PX4 repository
├── Trentos             # TrentOS folder added manually
│   ├── docker/         # TrentOS docker containers
│   └── sdk/            # TrentOS sdk
├── px4.patch           # Patches to PX4-Autopilot
├── scripts/            # Utility script
└── src                 # Source code
    ├── apps
    │   ├── proxy/      # C++ proxy source code
    │   └── trentos/    # TrentOS application (companion computer) source code
    └── libs/           # Modules shared between proxy/ and trentos/
```