Autonomous Drone Engineer

B4 – Software Architecture

Paul.Guermonprez@intel.com
Autonomous Drone Solutions Architect

Basic Drone Control

Compute Board / Flight Controller

The Intel Aero RTF Drone has two parts:

- A Flight Controller (FC), taking care of the stabilization, compass, GPS and other simple sensors: **Intel Aero Flight Controller**
- A Compute Board running linux, taking care of the high level mission and complex sensors like Intel RealSense: **Intel Aero Compute Board**

The two components are connected over a **serial port**.
The protocol used is **MAVLINK**, an industry standard.

Most drone software developers only need to develop software for the Compute Board running linux.

To pilot the drone, you send MAVLINK messages.

Serial Port or Server Socket?

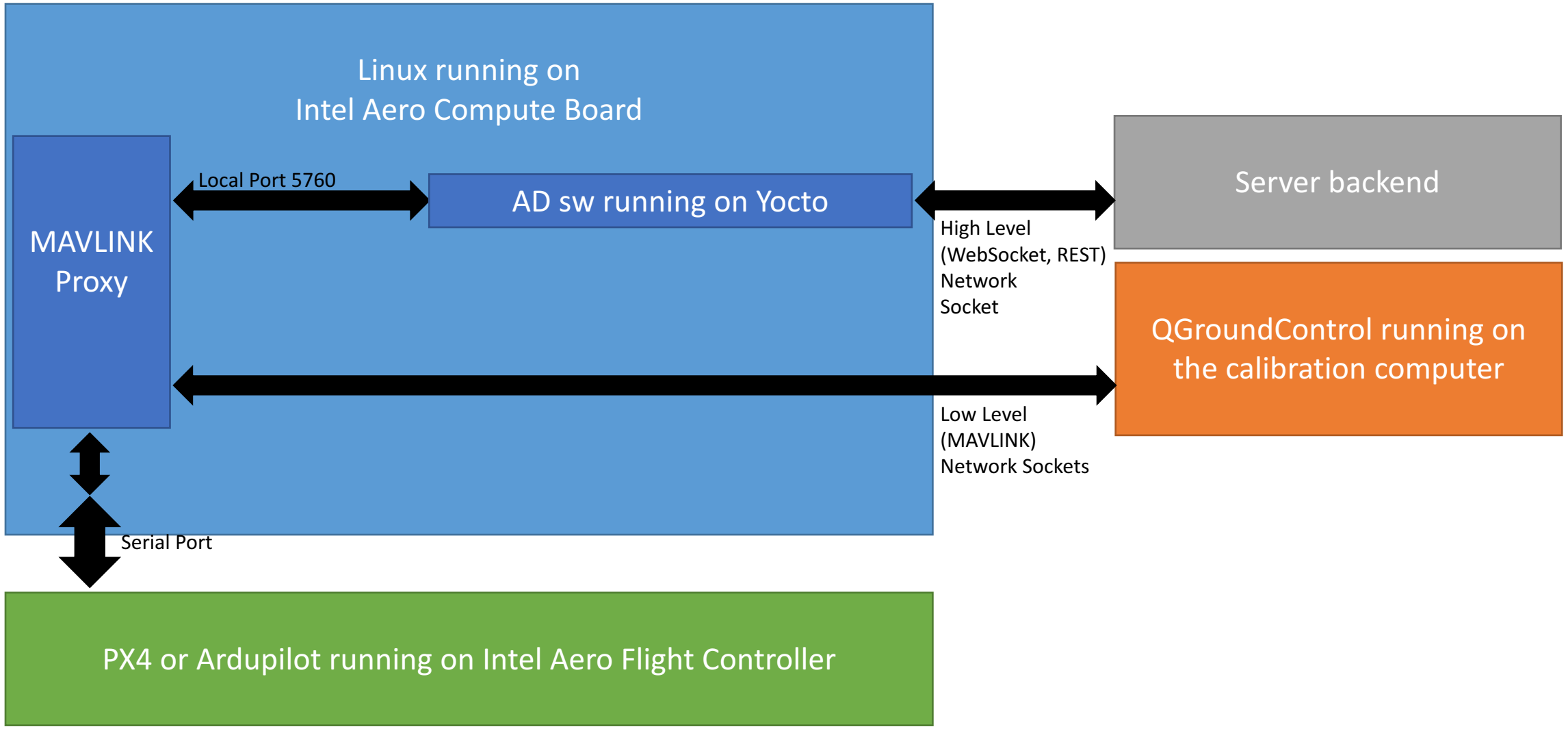
The default Linux build shipped with Intel Aero comes with a proxy software exposing the serial port as a network socket (**port 5760**) and sending MAVLINK messages over the network.

It is interesting for several reasons:

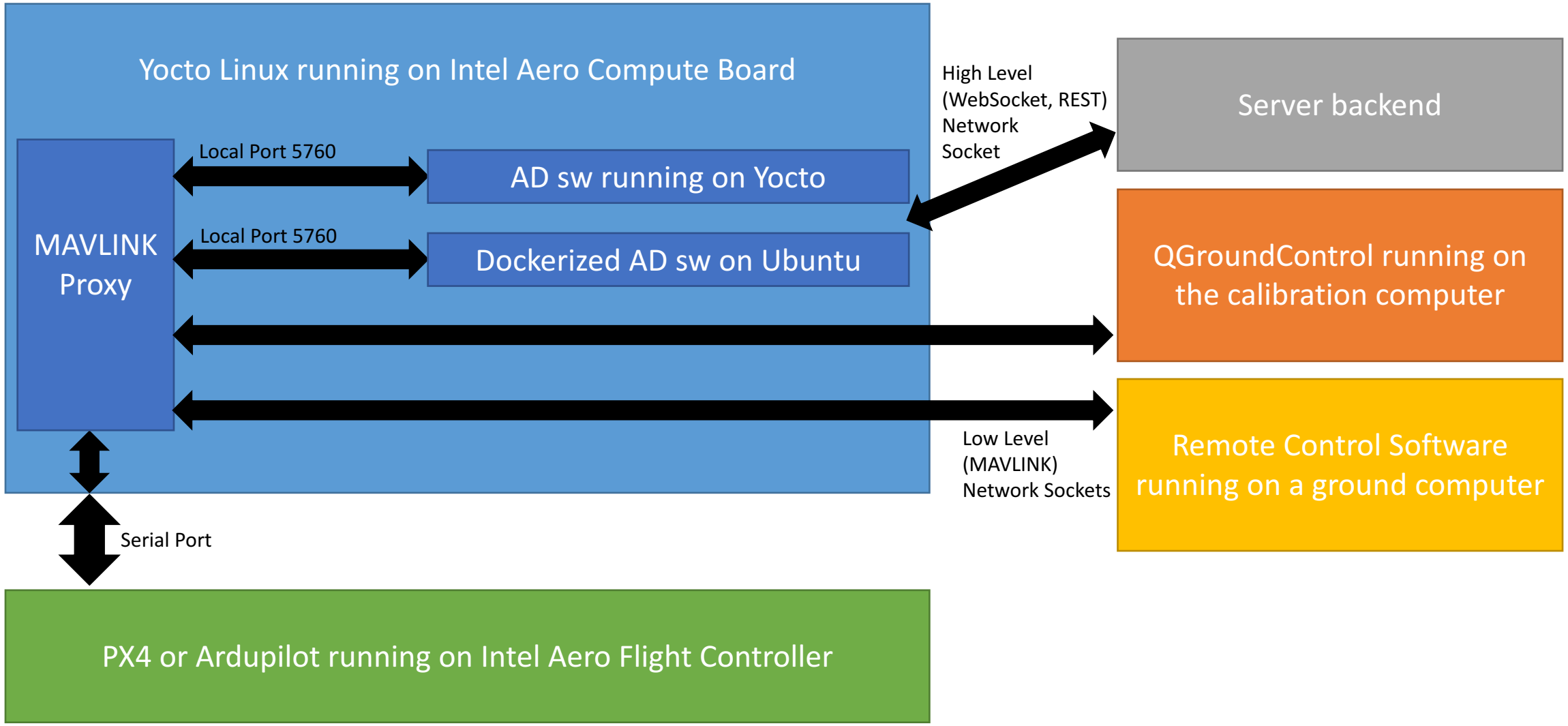
- The MAVLINK messages can be sent to the **calibration** computer over the network: You don't need a cable to calibrate your drone
- Accessing a network port is **easier** than a serial port
- Several software stacks can **share** the flight controller data and access

Conclusion: instead of sending MAVLINK messages over a serial port, we'll send **MAVLINK messages to the 5760 network port**.

Serial Port or Server Socket?



Serial Port or Server Socket?



MAVLINK Abstraction?

MAVLINK is an industry standard, APIs are available in various languages like C/C++ and Python. Drone developers should know the basics of MAVLINK. But it's low level and not very friendly.

Several frameworks propose higher level abstractions and interfaces:

- **DroneCore**, part of the DroneCode initiative. http://dronecore.io Designed for PX4.
- MAVProxy, in Python http://ardupilot.github.io/MAVProxy/
  Mostly works with Ardupilot Flight Controller Stack.
- Module for **ROS**, the Robotic Operating System.
  More on ROS in the next slides. http://wiki.ros.org/mavlink
- **Dronekit**, originally developed by 3D Robotics.
  Not very active lately, but still interesting. http://dronekit.io
- Several startups developed **friendly proprietary stacks** on top of MAVLINK.
  You may encounter their stack if you buy one of their drones.
  Examples: DroneSmith, Drotek, OttoFly, Flylab …
  https://github.com/intel-aero/meta-intel-aero/wiki/80-Intel-Aero-Ecosystem

Summary

Propositions:

- Use QGroundControl to calibrate your drone and monitor the flight during development. It will use MAVLINK messages over the network. It is a **convenient debug and development mode**.
- **Learn** the basics of MAVLINK.
- Use MAVLINK **abstractions** to make the development of your autonomous drone logic simpler.
- Avoid using the serial port directly, use port **5760**.

Autonomous Drone Development

Sensors and Network

With MAVLINK over the port 5760, we can now write a software running on the Compute Board that will control the drone flight controller. Why do we want to do that?

- Go beyond the simple GPS path: *define/adapt the **mission on the fly***
- Interact with the network (**Wifi or LTE**): *receive instructions, traffic information, give feedback, upload data …*
- Interact with **smart sensors** like Intel RealSense, make decisions locally and quickly: *collision avoidance, SLAM, follow target …*

How to use Intel RealSense from the RTF?

RealSense is not just a pair of cameras. It is a smart 3D sensor returning the fully computed 3D depth matrix to the Compute Board.

- You can access this **depth matrix** directly with libRealSense: https://github.com/IntelRealSense/librealsense
- If you want to use RealSense to **avoid collisions**, you can test this library and code your own library adapted to your specific needs: https://github.com/01org/collision-avoidance-library
- To perform **SLAM**, you can use this library (but it will require a different RealSense camera: model ZR300): https://software.intel.com/sites/products/realsense/slam/developer_guide.html

Computer Vision

The video devices included in the RTF Drone are accessible as standard **V4L2** video devices (/dev/video*). You can access them by using the V4L2 library from the language of your choice or from the command line with **gstreamer**. Video devices are also exposed as **RTSP streams**. Typical usages:

- Video recording and streaming: **gstreamer, RTSP**
- Simple image analysis: **OpenCV**
- Vision based 3D positioning and virtual servoing: **VISP, ARToolkit**
- Reading QR Codes: **zbar**
- Your custom code, using the library of your choice …

ROS – Robotic Operating System

It may be tricky to integrate several libraries like MAVLINK, VISP, RealSense together. **ROS** is a way to unify the interfaces and simplify the integration of components coming from various sources.

ROS is not an operating system, it's a stack running on top of the Linux OS (Yocto or Docker-Ubuntu in our case).

ROS has modules such as:

- MavROS for MAVLINK http://wiki.ros.org/mavros
- VISP http://wiki.ros.org/visp
- OpenCV http://wiki.ros.org/vision_opencv
- RealSense http://wiki.ros.org/RealSense

ROS.org

Summary

Once you understood the basics of drone control over MAVLINK, you have a vast choice of libraries at your disposal.

- Some come from Intel, like the RealSense related libraries.
- Other from the Open Source ecosystem,
  like Computer Vision libraries VISP, ARtoolkit, OpenCV.
- And you can always code your own,
  using the language of your choice.

Intel Aero is an open platform, the limit is your imagination.

Thanks

Paul.Guermonprez@intel.com

https://intel-aero.github.io