



**POLITECNICO**  
MILANO 1863

**Politecnico di Milano**  
**Department of Aerospace Science and Technology**

## **STANDARD SELECTION FOR SIMULATOR INTEGRATION**

**Marek S. Łukasiewicz**

Integrate available resources into a flight simulation for the following projects:

- Rotorcraft-Pilot Coupling
- Manned-Unmanned Teaming
- G-Seat motion cueing
- *Future research, not yet defined*

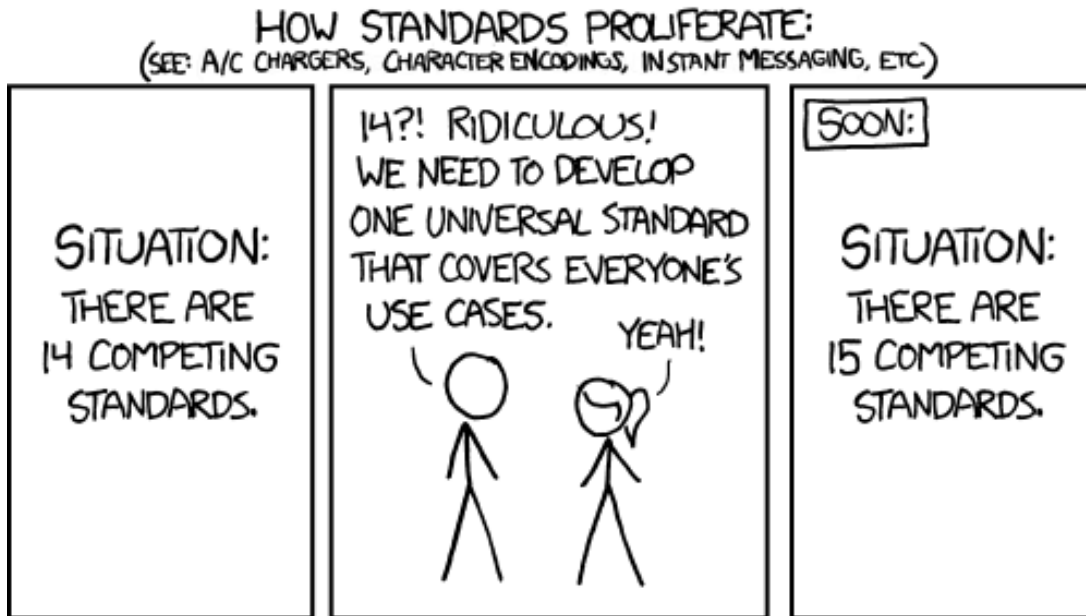


Figure: 'Standards' webcomic, reproduced from [xkcd.com](http://xkcd.com), CC-BY-NC 2.5

Summary of selected options from the previous presentation

Solution	Relevant Definitions	Ease of use	Software Constraints	Wider Applicability
Current	bad	neutral	good	very bad
HLA	very good	neutral	bad	neutral
DIS	bad	neutral	good	neutral
DDS	neutral	good	neutral	good
<b>MAVLink</b>	good	good	good	good
Mixed approach	neutral	neutral	good	neutral

Other solutions characterised, but omitted from presentation:

FMI (Functional Mock-up Interface), DCP (Distributed Co-Simulation Protocol), ZeroMQ, MessagePack, Capnproto/Protobuf, Web (JSON+HTTP+WS), MQTT (Message Queue Telemetry Transport), ACMI Tacview

On the basis of online documentation for each considered option, **MAVLink** protocol was chosen as the most promising option.

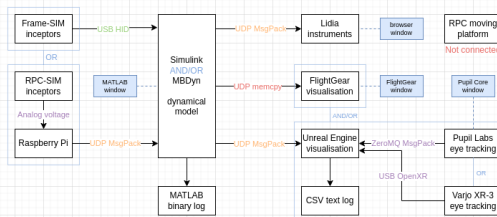
On that assumption, more detailed architecture proposal was made (shown on right), and sent out with request for comments to colleagues from academia, working in UAV or manned simulation, and publicly in ArduPilot forum (UAV software, one of main MAVLink dependents).

## Simulator architecture comparison

A proposal for reworking research flight simulators Rotorcraft-Pilot Coupling and Frame-SIM.

Marek S. Łukasiewicz (marek.łukasiewicz@polimi.it), 2024-01-29  
 Department of Aerospace Science and Technology (DAER) of Politecnico di Milano  
 © 2024. This work is openly licensed via CC-BY 4.0

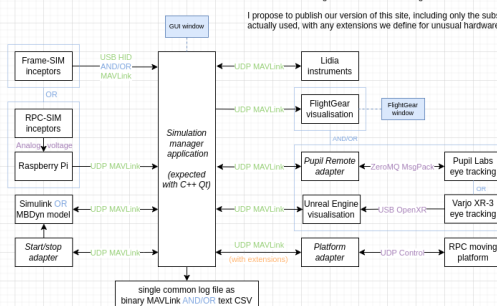
### Current setup



### Diagram legend



### Proposed MAVLink rework



Read more about MAVLink in its online documentation, which includes a large set of common messages. I propose to publish our version of this site, including only the subset actually used, with any extensions we define for unusual hardware.

Question asked on 2024-02-01 in public ArduPilot Discord, channel #research, quoted verbatim:

*I'm seriously considering (ab)using MAVLink to serve as the backbone of a crewed aircraft simulator. For research and teaching purposes, we're building it as loosely coupled modules e.g. the dynamic model takes controls and weather info, and publishes new aircraft attitude.*

*Even if I **ignore** all benefits for drone-helicopter collaboration, it seems to check all the boxes:*

- *free, open source*
- *already has well-documented definitions for all the aeronautical data I need*
  - ▶ *also microservices like heartbeat, params, command ack*
  - ▶ *I can easily extend it if I miss something specific like eye tracking*
- *could be useful for graduating students, instead of them figuring out my custom in-house developed mess*
- *doesn't force me to rewrite the whole software execution model, just share data*

*I'd really appreciate your feedback, I'm afraid that this seems such a silver bullet only because I have been using MAVLink for too long*

Received 9 replies overall to the open question. None were negative, four were decidedly positive.

Specific advantages pointed out:

- Using an established solution is really important to not depend on the author to on-board every user (3 responses). Also online search and chatbots will work much better
- Defining requirements through user stories was done well (2 responses)
- Widely used in the UAV environment, with the largest documentation
- Connecting all application elements with **one** standard is a valuable goal
- Very easy to integrate in a MUM-T scenario
- Can be sent over any transport medium, also has provisions for tunneling other packets through

Specific disadvantages pointed out by respondents, *with rebuttal below*:

- Payload is limited to only 256 bytes (2 responses). Simple fields not suited for arbitrarily complex data structures (eg. animation state of an airplane). *A JSON-like communication was already tested with lidia. In practice there was push for constant, well defined and described messages in the team. Efficient encoding makes it a viable format for direct logging to binary file.*
- Has some other features for lossy links that aren't needed in the context of local network, incl. sequencing packets and CRC. *To have simpler, stateless connections with less latency, UDP will be used which is lossy. CRC is handled by libraries, not adding any work for user.*
- Using ZeroMQ or MQTT would provide publish and subscribe control and other conveniences, once the admittedly complex setup is done. *MAVLink defines node discovery and requesting data stream. Will use software tools developed by autopilot authors for debugging.*
- Some devices would not be covered by the standardized messages. *Same person later added that custom protocol would copy ca. 80% anyway*



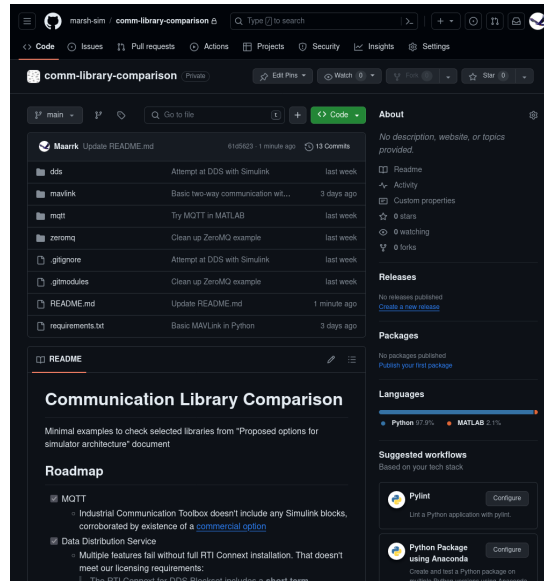
Another round of comparison was made, adding alternatives suggested by respondents:

- MQTT
- Data Distribution Service
- ZeroMQ
- MAVLink

An attempt was made at creating a minimal demo for each, integrating them with the following tools used in the simulator facility:

- Python
- MATLAB Simulink

*Notably, C and C++ were omitted. Each of these solutions is already implemented in either of these languages, so easy integration is expected.*



- MQTT features a centralised architecture where messages are routed by a single broker instance, well suited for single simulation manager
- Example in the repository uses Mosquitto broker, with `paho-mqtt` for Python subscriber, and ‘Industrial Communication Toolbox’ for MATLAB subscriber
- Found out that ‘Industrial Communication Toolbox’ doesn’t include any Simulink blocks, corroborated by existence of a commercial package specifically for that use case

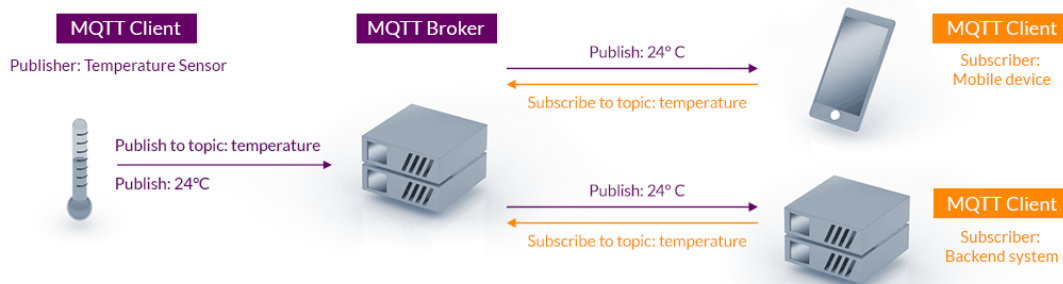


Figure: MQTT architecture, reproduced from [mqtt.org](https://mqtt.org) © 2022 MQTT.org

9

- DDS also offers publish/subscribe pattern, but doesn't have a central broker, and topic contents are described with XML files
- Example in the repository uses `cyclonedds` for Python writer and subscriber, and 'DDS Blockset' for MATLAB subscriber
- Found out that 'DDS Blockset' is dependent on commercial 'RTI Connex' software, and only includes a short term license. Also requires Simulink model to be run in Real-Time, which turned out too difficult to configure.

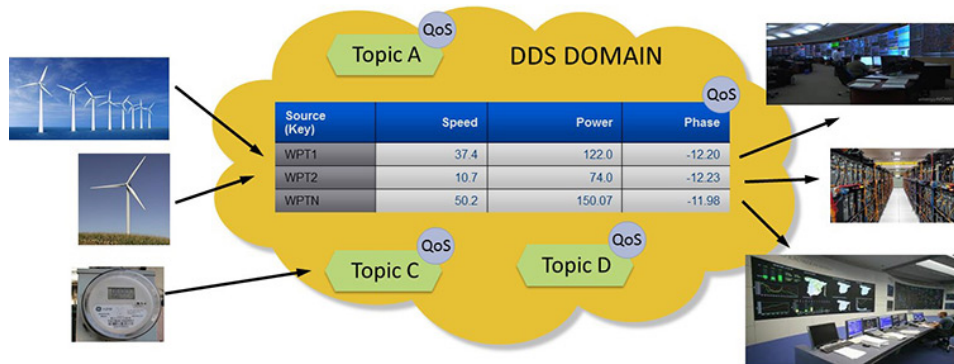


Figure: DDS infographic, reproduced from [dds-foundation.org](https://dds-foundation.org). © 2024 OMG, Inc.

- ZeroMQ also offers publish/subscribe pattern without central broker, is much more open and has multiple services standardised on top of this transport
- Python usage with `pymq` package was recently tested in [another project](#)
- Example for MATLAB was run following a [blog post on MATLAB Central by MathWorks staff](#), which involves local build of C and C++ libraries for ZeroMQ and loading them as Simulink extensions  
 Running the provided project required C++ compiler setup, installing external dependencies, checking out past versions of Git repositories, debugging dynamic library loading, while having virtually no information from Simulink error messages

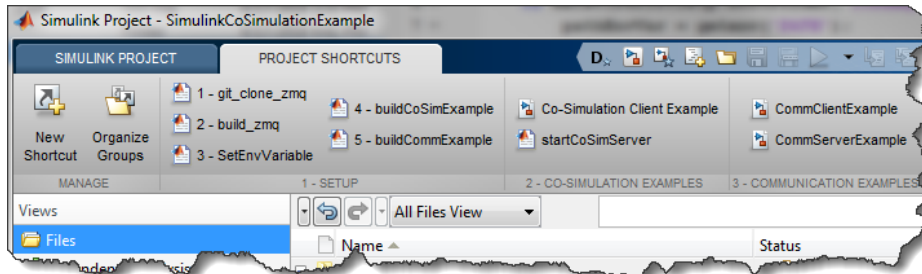


Figure: Simulink project steps, reproduced from [Communicating with an External Application for Co-Simulation](#) blog post mentioned above. © 2018 The MathWorks, Inc.

- Only specifies message contents, doesn't involve any specific transport
- Example in the repository uses `pymavlink` for Python server and node, 'UAV Toolbox' and 'Instrument Control Toolbox' for MATLAB node. Both include standard messages, and allow loading custom XML definitions.
- Broadcasting without a central node was successfully done in Python using IP Multicast, but in Simulink required the difficult 'Real-Time' setup
- A classical client-server architecture was quickly and easily implemented
  - ▶ Client sends HEARTBEAT from any ephemeral UDP port to a well-known port on the server, which then replies on the same socket. All under 100 lines of Python
  - ▶ Port 24400 was chosen arbitrarily, no other services using it were found [1, 2, 3, 4]

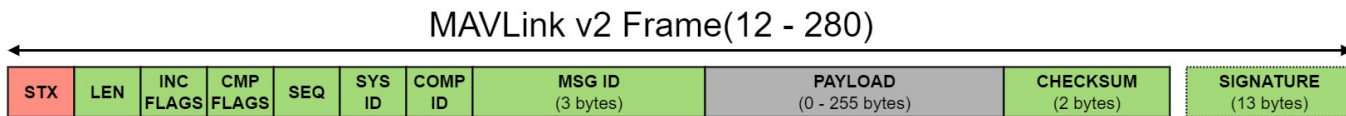
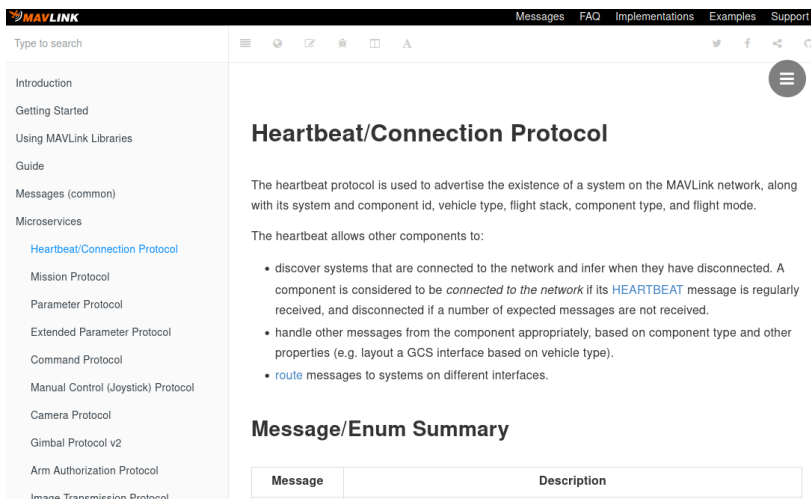


Figure: MAVLink packet format from [Developer Guide](#), CC-BY 4.0

- 1 Check if there is already a relevant convention for a feature in [Microservices](#) section of Dev Guide
  - ▶ If nothing was found, search the [Common Message Set](#)
- 2 Read documentation for the service: [Heartbeat/Connection Protocol](#)
- 3 Read documentation for specific messages: [HEARTBEAT](#)



The screenshot shows the MAVLINK website documentation for the Heartbeat/Connection Protocol. The left sidebar contains a navigation menu with the following items: Introduction, Getting Started, Using MAVLink Libraries, Guide, Messages (common), Microservices (with a sub-link for Heartbeat/Connection Protocol), Mission Protocol, Parameter Protocol, Extended Parameter Protocol, Command Protocol, Manual Control (Joystick) Protocol, Camera Protocol, Gimbal Protocol v2, Arm Authorization Protocol, and Image Transmission Protocol. The main content area is titled "Heartbeat/Connection Protocol" and contains the following text:

The heartbeat protocol is used to advertise the existence of a system on the MAVLink network, along with its system and component id, vehicle type, flight stack, component type, and flight mode.

The heartbeat allows other components to:

- discover systems that are connected to the network and infer when they have disconnected. A component is considered to be *connected to the network* if its [HEARTBEAT](#) message is regularly received, and disconnected if a number of expected messages are not received.
- handle other messages from the component appropriately, based on component type and other properties (e.g. layout a GCS interface based on vehicle type).
- [route](#) messages to systems on different interfaces.

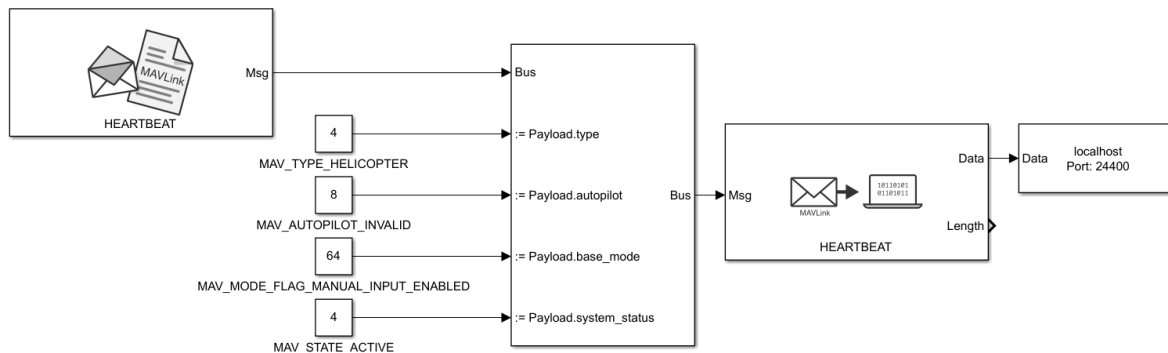
Below the text is a section titled "Message/Enum Summary" which contains a table with two columns: "Message" and "Description".

Message	Description
---------	-------------

## HEARTBEAT (#0)

[Message] The heartbeat message shows that a system or component is present and responding. The type and autopilot fields (along with the message component id), allow the receiving system to treat further messages from this system appropriately (e.g. by laying out the user interface based on the autopilot). This microservice is documented at <https://mavlink.io/en/services/heartbeat.html>

Field Name	Type	Values	Description
type	uint8_t	MAV_TYPE	Vehicle or component type. For a flight controller component the vehicle type (quadrotor, helicopter, etc.). For other components the component type (e.g. camera, gimbal, etc.). This should be used in preference to component id for identifying the component type.
autopilot	uint8_t	MAV_AUTOPILOT	Autopilot type / class. Use MAV_AUTOPILOT_INVALID for components that are not flight controllers.
base_mode	uint8_t	MAV_MODE_FLAG	System mode bitmap.
custom_mode	uint32_t		A bitfield for use for autopilot-specific flags
system_status	uint8_t	MAV_STATE	System status flag.
mavlink_version	uint8_t_mavlink_version		MAVLink version, not writable by user, gets added by protocol because of magic data type: uint8_t_mavlink_version

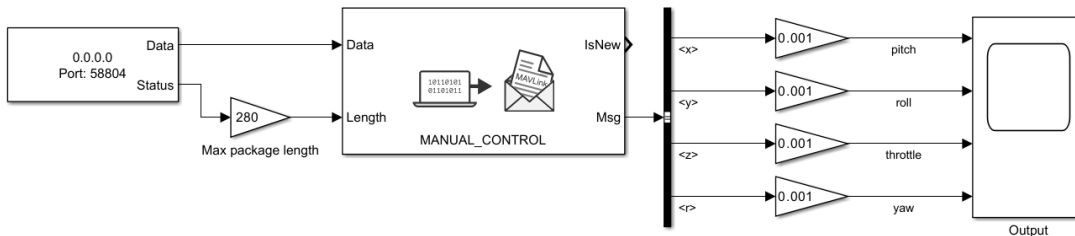


Implementing the HEARTBEAT message. Top: MAVLink Dev Guide, bottom: Simulink blocks

## MANUAL\_CONTROL ( #69 )

[Message] This message provides an API for manually controlling the vehicle using standard joystick axes nomenclature, along with a joystick-like input device. Unused axes can be disabled and buttons states are transmitted as individual on/off bits of a bitmask

Field Name	Type	Description
target	uint8_t	The system to be controlled.
x	int16_t	X-axis, normalized to the range [-1000,1000]. A value of INT16_MAX indicates that this axis is invalid. Generally corresponds to forward(1000)-backward(-1000) movement on a joystick and the pitch of a vehicle.
y	int16_t	Y-axis, normalized to the range [-1000,1000]. A value of INT16_MAX indicates that this axis is invalid. Generally corresponds to left(-1000)-right(1000) movement on a joystick and the roll of a vehicle.
z	int16_t	Z-axis, normalized to the range [-1000,1000]. A value of INT16_MAX indicates that this axis is invalid. Generally corresponds to a separate slider movement with maximum being 1000 and minimum being -1000 on a joystick and the thrust of a vehicle. Positive values are positive thrust, negative values are negative thrust.
r	int16_t	R-axis, normalized to the range [-1000,1000]. A value of INT16_MAX indicates that this axis is invalid. Generally corresponds to a twisting of the joystick, with counter-clockwise being 1000 and clockwise being -1000, and the yaw of a vehicle.
buttons	uint16_t	A bitfield corresponding to the joystick buttons' 0-15 current state, 1 for pressed, 0 for released. The lowest bit corresponds to Button 1.

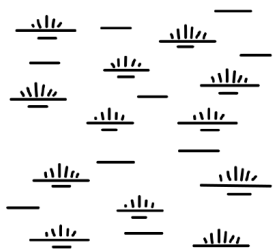


MANUAL\_CONTROL message. Top: MAVLink Dev Guide, bottom: Simulink blocks



Play the screen recording of running the Simulink model here.

*If file missing: [unlisted video on personal YouTube channel](#)*



## MARSH Sim — Modular Architecture for Reconfigurable Simulation of Helicopters

- A real marsh is an **ecosystem** where diverse plants grow together, connected by shallow water
- A pun on ‘marshalling’, quoting [Wikipedia](#):  
*process of transforming the memory representation of an object into a data format suitable for storage or transmission*
- Similar to MAV\_ prefix used in identifiers defined by MAVLink
- Didn’t find any name clashes through online search engines
- Short and simple spelling with no diacritics

- Detailed proposal for connecting simulator modules was prepared using established software design methodology
- In order to minimize personal bias, consulted literature, online documentation and domain professionals
- Multiple options were tested in practice, specifically for the use case and tools used in the research group
- All stages of analysis have identified a dialect of MAVLink as the preferable solution for the simulator facility