sensors

Article

Latency Reduction and Packet Synchronization in Low-Resource Devices Connected by DDS Networks in Autonomous UAVs

Joao Leonardo Silva Cotta [1,*], Daniel Agar [2], Ivan R. Bertaska [3], John P. Inness [3] and Hector Gutierrez [4]

1 Department of Aerospace Engineering, Physics and Space Sciences, Florida Institute of Technology, Melbourne, FL 32901, USA
2 Core Developer, PX4 Autopilot, 8092 Zurich, Switzerland; daniel@agar.ca
3 Control Systems Design and Analysis Branch, NASA Marshall Space Flight Center, Huntsville, AL 35812, USA; ivan.r.bertaska@nasa.gov (I.R.B.); john.p.inness@nasa.gov (J.P.I.)
4 Department of Mechanical and Civil Engineering, Florida Institute of Technology, Melbourne, FL 32901, USA; hgutier@fit.edu
* Correspondence: jcotta2018@my.fit.edu

Abstract: Real-time flight controllers are becoming dependent on general-purpose operating systems, as the modularity and complexity of guidance, navigation, and control systems and algorithms increases. The non-deterministic nature of operating systems creates a critical weakness in the development of motion control systems for robotic platforms due to the random delays introduced by operating systems and communication networks. The high-speed operation and sensitive dynamics of UAVs demand fast and near-deterministic communication between the sensors, companion computer, and flight control unit (FCU) in order to achieve the required performance. In this paper, we present a method to assess communications latency between a companion computer and an RTOS open-source flight controller, which is based on an XRCE-DDS bridge between clients hosted in the low-resource environment and the DDS network used by ROS2. A comparison based on the measured statistics of latency illustrates the advantages of XRCE-DDS compared to the standard communication method based on MAVROS-MAVLink. More importantly, an algorithm to estimate latency offset and clock skew based on an exponential moving average filter is presented, providing a tool for latency estimation and correction that can be used by developers to improve synchronization of processes that rely on timely communication between the FCU and companion computer, such as synchronization of lower-level sensor data at the higher-level layer. This addresses the challenges introduced in GNC applications by the non-deterministic nature of general-purpose operating systems and the inherent limitations of standard flight controller hardware.

Keywords: flight control unit; companion computer; DDS network; non-deterministic delay; communication latency; MAVLink; robot operating system; loosely coupled systems

check for updates

Citation: Silva Cotta, J.L.; Agar, D.; Bertaska, I.R.; Inness, J.P.; Gutierrez, H. Latency Reduction and Packet Synchronization in Low-Resource Devices Connected by DDS Networks in Autonomous UAVs. *Sensors* **2023**, *23*, 9269. https://doi.org/10.3390/s23229269

Academic Editor: Roman Sotner

Received: 13 October 2023
Revised: 9 November 2023
Accepted: 16 November 2023
Published: 18 November 2023

1. Introduction

The growing interest in Autonomous Aerial and Underwater Vehicles (AAUVs) is rapidly reshaping technologies that support new applications and improve performance. The extensive range of applications and increasing complexity of missions are prompting the development of advanced real-time flight controllers capable of integrating guidance, navigation, and control (GNC) algorithms to reliably execute increasingly complex tasks.

State-of-the-art systems use a divided architecture to address the computational limitations of standard flight control units (FCU), separating tasks between low-level and higher-level layers [1]. This allows computationally intensive GNC functions to be handled by the higher-level layers, which rely on companion computers that are typically more powerful than the low-level FCU. While this division is a good alternative to support mission goals, it introduces a critical drawback in that high-level layers often use general-purpose operating systems known for their non-deterministic time management and unpredictable delays [2]. This creates challenges in developing high performance GNC solutions, as random latencies introduced by the operating system and interface networks can affect the overall performance of motion control systems [1–4].

The development of autonomous robotic platforms emphasizes software modularity and integration with widely used frameworks. This study presents an innovative method for reducing latency in packet synchronization between a companion computer and a Real-Time Operating System (RTOS) flight controller (a device notably constrained in ternms of computational resources) based on widely available open-source tools: a DDS network in the Robot Operating System 2 (ROS 2) environment, the PX4 FCU, and the eXtremely Resource Constrained Environment-Data Distribution Service (MicroXRCE-DDS) as an intermediary agent between system level layers. Latency measurements for the proposed method are compared against results from the same flight controller software, PX4, with the well-known open-source Micro Air Vehicle Link (MAVLink) protocol, Robotic Operational System (ROS), and MAVROS, a communication bridge between MAVlink and ROS [1,5–8].

In summary, this paper presents a method to assess latency in communications between the flight controller and companion computer. A comparison based on the measured latency statistics illustrates the advantages of XRCE-DDS compared to standard communication methods based on MAVROS-MAVLink. More importantly, an algorithm to estimate latency offset and clock skew based on an exponential moving average filter is presented, providing a tool for latency estimation and correction that can be used by developers to improve synchronization of processes that rely on timely communication between the FCU and companion computer, such as synchronization of lower-level sensor data at the higher-level layer. This addresses the challenges introduced in GNC applications by the non-deterministic nature of general-purpose operating systems and the inherent limitations of standard flight controller hardware.

2. Materials and Methods

2.1. Flight Controller Unit

The flight control unit used in this study was the Holybro Pixhawk 4, running a slightly modified PX4 firmware version 1.14.0 [7]. The FCU uses two ARM processors: an STM32F765 as the CPU and an STM32F100 as the IO processor. The internal sensors include an inertial measurement unit (IMU), magnetometer, and barometer, which can be fused with other non-deterministic and deterministic sensors for state estimation during operation. Communication between the FCU and high-level layer was implemented using an FCU UART port at 921600 BPS.

2.2. PX4 Firmware

PX4 works through a topic publish/subscribe framework called uORB that uses internal message definitions. This format allows the user to declare custom uORB definitions to be used internally in the FCU, and enables all uORB definitions to be exported for use in an ROS/ROS 2 package. While PX4 can accept MAVlink messages, these have to be converted to uORB in order to properly interface with critical internal functionalities; custom modifications to MAVlink require substantial user effort [7].

2.3. ROS/ROS 2

In robotic systems with divided architectures, the two most popular current frameworks are the Robot Operating System (ROS) and Robot Operating System 2 (ROS 2). Although similar in name, their internal publish/subscribe models have fundamentally different protocols. ROS uses a custom message queuing protocol similar to Advanced Message Queuing Protocol (AMQP), in which two types of nodes are used: ROS nodes and a ROS Master. A roscore is a computational entity that can communicate with other nodes and can publish and subscribe to receive and transmit data. The ROS Master acts as central hub for the ROS ecosystem, keeping a registry of active nodes, services, and topics to ease communication between nodes, i.e., discovery registration. Multiple ROS nodes can be executed at the same time and be registered to a roscore; however, cooperation between multiple roscores is not natively supported by ROS [9]. In contrast, ROS 2 implements a decentralized architecture using a Data Distribution Service (DDS), which eliminates the need for a ROS Master and automates the discovery registration. ROS 2 nodes can transmit and receive data without central coordination, with the direct result that ROS2 incurs less latency than ROS. ROS 2 enables the use of several node types (regular, real-time, intra-process, composable, and lifecycle nodes), enabling cooperation between different layers of a system as long as the ROS 2 nodes of interest are under the same DDS domain [10,11].

2.4. MAVlink

MAVlink is an open-source messaging protocol widely used by the UAV industry. It uses a header-only message marshaling library optimized for low-resource environments. The implementation of custom messages in MAVlink is a demanding task, in particular when integrating ROS/ROS 2 routines. Custom implementations require changes to the FCU's source code as well as modifications to the communications bridge used to transmit and translate messages between the ROS/ROS 2 nodes and the FCU [8,10–12].

2.5. MAVROS

MAVROS is the most commonly used open-source bridge between ROS and MAVlink, converting ROS messages to their MAVlink equivalent. MAVROS source code has to be modified to implement custom MAVlink messages; for instance, a MAVROS extra plugin, which requires a complex set of tasks, is needed to implement a simple addition. Although the MAVROS team has announced their intention to support ROS 2 in the future, the majority of current UAV applications using MAVlink coupled with ROS/ROS 2 use the ROS-compatible MAVROS version. The use of MAVROS and ROS 2 could be implemented using a ROS–ROS2 bridge, although this would increase computational effort and latency, as ROS and ROS2 use different communication architectures [13].

2.6. MicroXRCE-DDS

MicroXRCE-DDS (Figure 1) is a protocol that allows low-resource devices to be integrated with a DDS network while maintaining real-time and reliable data distribution capabilities (RTPS). In this investigation, it is used as an ROS 2 middleware to enable nodes running on the FCU to communicate with nodes running on the companion computer [2,14,15].

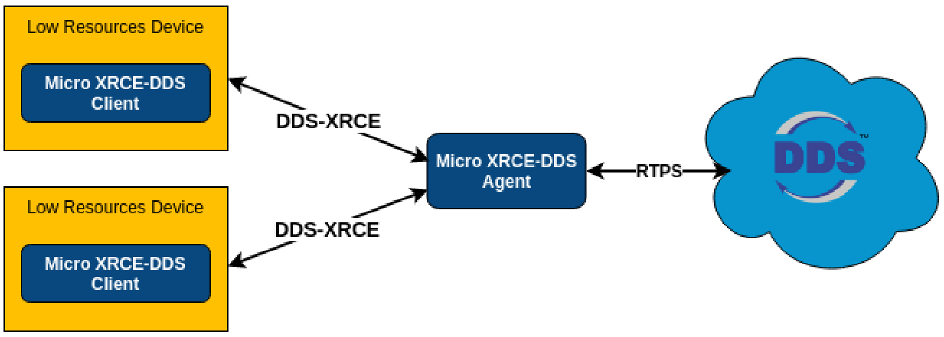

Figure 1. MicroXRCE-DDS architecture [15].

2.7. Companion Computer

The companion is an auxiliary computer running ROS/ROS 2 nodes to support the required mission functionality. It communicates with the FCU using an FTDI cable (USB to UART). For this investigation, the companion computer was an RPi 4 with 8 GB of RAM.

2.8. Host Computer

The host was implemented on an x86 computer with an Intel Xeon(R) Gold 6148 CPU running Ubuntu 22.04 with Gazebo 11 and QGroundControl. The host was used as the terminal for the FCU and as a virtual environment to emulate the PX4 internal sensor data [16,17].

2.9. Latency Assessment

Latency assessment was implemented via hardware-in-the-loop simulation, and consisted of measuring the time offset between transmitted and received messages on both the FCU side and a high-frequency ROS or ROS 2 node. The forward path consisted of messages sent from the companion computer, simulating data from an external vision sensor to be fused at the PX4's Extended Kalman Filter (EKF2); latencies were collected when the message was parsed into the estimator module in the FCU. The reverse path consisted of messages sent from the FCU to the ROS or ROS 2 node containing raw measurements from the FCU's IMU. The choice of messages relates to typical GNC applications such as SLAM or Visual Inertia Odometry, in which rapid IMU feedback and fast state estimate transmission are crucial for mission performance. The messages were custom-modified to carry the same number of 115 bytes.

Figure 2 outlines the PX4's EKF2 algorithm [18]. The top block shows the main estimator, which uses a ring buffer to account for different sensor sampling frequencies and predict the states in a delayed horizon. The second block is the output predictor, which uses corrected high-frequency IMU measurements for quick state prediction and UAV rate control. Pseudocode and diagrams outlining communications between higher-level and lower-level processors are presented for both scenarios in Algorithms 1 and 2, and in Figures 3 and 4.

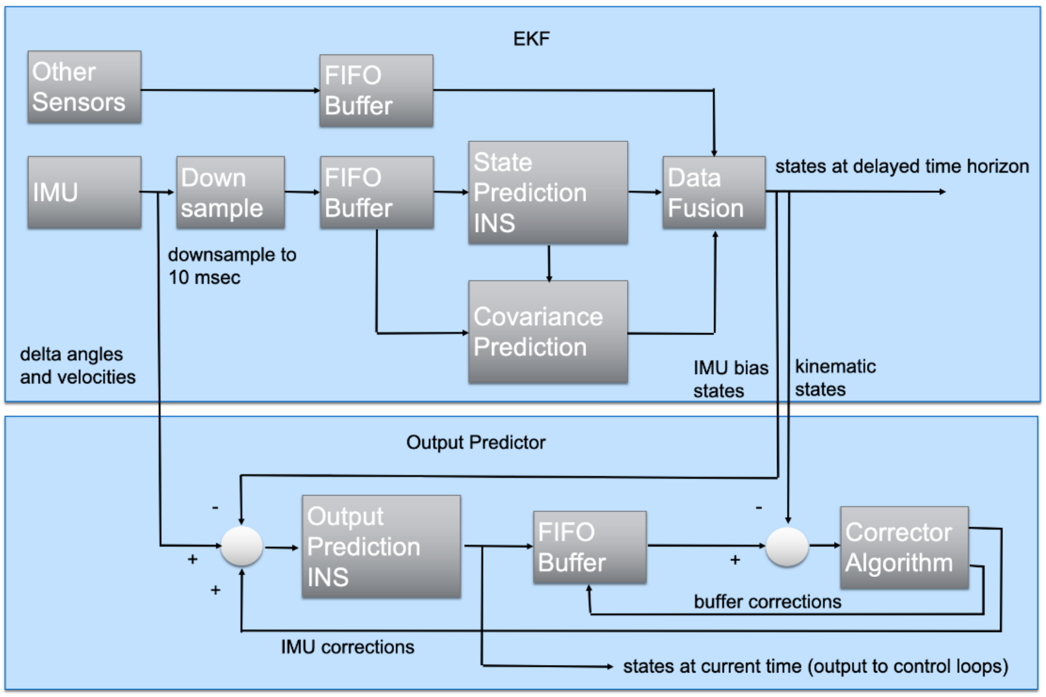

Figure 2. PX4 EKF2 architecture [18].

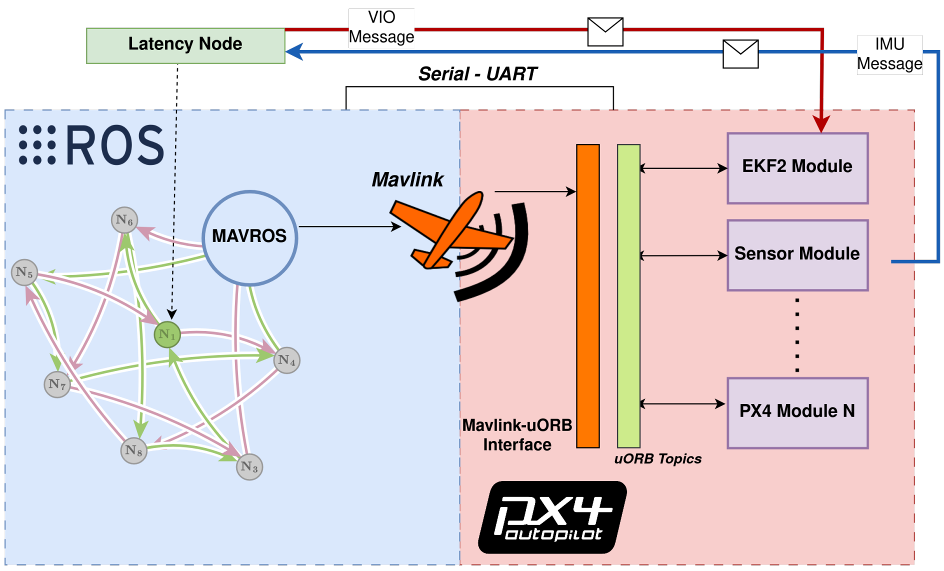

Figure 3. ROS and FCU communications for latency assessment using MAVROS/MAVlink bridge.

Algorithm 1 Latency Test Node in rospy (ROS)

```
Class LatencyTest:
  Initialize variables
  Initialize publisher and subscriber
  Initialize timer

function INITIALIZE VARIABLES
    $time\_packet\_creation \leftarrow$ empty deque of max length 2
    $offset\_estimated, N, high\_dev\_counter, high\_rtt\_counter \leftarrow 0$
    $alpha, beta \leftarrow 0.05$
    $skew \leftarrow 0$
    $convergence\_window \leftarrow 500$

function INITIALIZE PUBLISHER AND SUBSCRIBER
    Subscribe to 'mavros/imu/data_raw'
    Publish to 'mavros/vision_pose/pose'

function INITIALIZE TIMER
    Set timer frequency to 200 Hz

function SENSOR CALLBACK(msg)
    Calculate $imu\_timestamp, current\_time$
    Obtain $imu\_time\_offset\_observed, rtt$
    Update $offset$ and counters based on $rtt$ and $imu\_time\_offset\_observed$
    Write data to log.txt

function UPDATE OFFSET AND COUNTERS(rtt, imu_time_offset_observed)
    if $rtt < 10{,}000$ then
        Update $alpha, beta$, and $offset$ using $imu\_time\_offset\_observed$
    else
        $high\_rtt\_counter \leftarrow high\_rtt\_counter + 1$

function CMDLOOP CALLBACK(event)
    Create and publish $vio\_msg$
    Append $vio\_msg.header.stamp$ to $time\_packet\_creation$

function RESET FILTER
    N $\leftarrow 0$
    Offset$_{Estimated} \leftarrow 0$
    Skew$_{\text{Estimated}} \leftarrow 0$
    $\hat{\alpha} \leftarrow \alpha_{\text{max}}$
    $\hat{\beta} \leftarrow \beta_{\text{max}}$
    high_deviation_count $\leftarrow 0$
    high_rtt_count $\leftarrow 0$
```

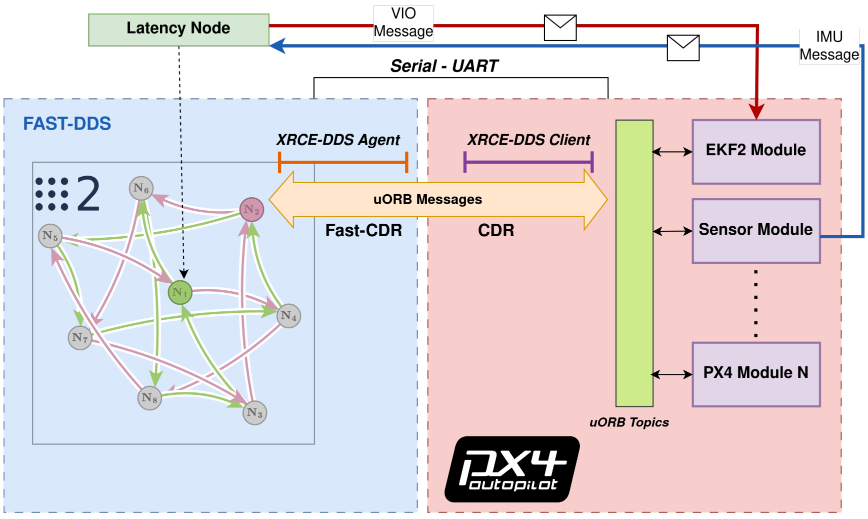

Figure 4. ROS 2 and FCU communications for latency assessment using the MicroXRCE–DDS bridge.

Algorithm 2 Latency Test Node in RCLPY (ROS 2)

```
Class LatencyTest extends Node:
    Initialize variables
    Initialize publisher and subscriber
    Initialize timer

function INITIALIZE VARIABLES
    $time\_packet\_creation \leftarrow$ empty deque of max length 2
    $offset\_estimated, N, high\_dev\_counter, high\_rtt\_counter \leftarrow 0$
    $alpha, beta \leftarrow 0.05$
    $skew \leftarrow 0$
    $convergence\_window \leftarrow 500$

function INITIALIZE PUBLISHER AND SUBSCRIBER
    Create a subscription to '/fmu/out/vehicle_imu'
    Create a publisher to '/fmu/in/vehicle_visual_odometry'

function INITIALIZE TIMER
    Set timer frequency to 200 Hz

function SENSOR CALLBACK(msg)
    Calculate $imu\_timestamp, current\_time$
    Calculate $imu\_time\_offset\_observed, rtt$
    Update $offset$ and counters based on $rtt$ and $imu\_time\_offset\_observed$
    Write data to "time_offsets.txt"

function UPDATE OFFSET AND COUNTERS(rtt, imu_time_offset_observed)
    if $rtt < 10{,}000$ then
        Update $alpha, beta$, and $offset$ using $imu\_time\_offset\_observed$
    else
        $high\_rtt\_counter \leftarrow high\_rtt\_counter + 1$

function CMDLOOP CALLBACK
    Create and publish $vio\_msg$
    Append $vio\_msg.timestamp\_sample$ to $time\_packet\_creation$

function RESET FILTER
    $\mathrm{N} \leftarrow 0$
    $\mathrm{Offset}_{Estimated} \leftarrow 0$
    $\mathrm{Skew}_{\mathrm{Estimated}} \leftarrow 0$
    $\hat{\alpha} \leftarrow \alpha_{\max}$
    $\hat{\beta} \leftarrow \beta_{\max}$
    high_deviation_count $\leftarrow 0$
    high_rtt_count $\leftarrow 0$
```

The ROS messages in Table 1 directly translate into the uORB messages shown in Table 2. The red and blue lines in Figures 3 and 4 show the message flow (IMU and VIO) and modules involved. Interpreting the time offset results requires understanding how the FCU firmware and the ROS/ROS 2 nodes estimate the time offset in each message, including how both lower-level and higher-level system time bases can be aligned and how the communication delay is quantified. The time offset is defined as the difference between two clock readings:

$$\text{Offset}_{\text{Observed}}(i) = \frac{T_{\text{Packet Creation}}(i) + T_{\text{Current}}(i) - 2 \times T_{\text{Remote Stamp}}(i)}{2} \tag{1}$$

where $T_{\text{Packet Creation}}$ is the time when the packet containing the uORB or MAVlink messages was created, that is, the time when information originated from the ROS/ROS 2 node or FCU was serialized and sent, $T_{\text{Remote Stamp}}$ is the time when the message was received and sent back from the remote level (either the higher-level or lower-level system layer), and T_{Current} is the current system time. Clock skew is defined as the difference in the register update rate (loop rate) at both the FCU and companion computer. The offset and clock skew are estimated using an exponential moving average filter, as described in [19,20]:

$$\begin{aligned}\text{Offset}_{\text{Estimated}}(i) &= \alpha \times \text{Offset}_{\text{Observed}}(i) \\ &\quad + (1-\alpha) \times (\text{Offset}_{\text{Estimated}}(i-1) + \text{Skew}_{\text{Estimated}}(i-1))\end{aligned} \tag{2}$$

$$\begin{aligned}\text{Skew}_{\text{Estimated}}(i) &= \beta \times (\text{Offset}_{\text{Estimated}}(i) - \text{Offset}_{\text{Estimated}}(i-1)) \\ &\quad + (1-\beta) \times \text{Skew}_{\text{Estimated}}(i-1)\end{aligned} \tag{3}$$

where α and β are the filter gains for the offset and skew, respectively. To check convergence of the estimated offset, the message round-trip time is obtained and to determine whether it falls within a maximum threshold:

$$T_{\text{RTT}}(i) = T_{\text{Current}}(i) - T_{\text{Packet Creation}}(i) < 10 \text{ ms}. \tag{4}$$

If Equation (4) is true, the deviation between the estimated offset and the latest observed offset is compared against a maximum threshold:

$$\text{Offset}_{\text{Estimated}}(i) - \text{Offset}_{\text{Observed}}(i+1) < 100 \text{ ms}. \tag{5}$$

If Equation (5) holds, the statistical quality of α and β is assessed for each estimated offset by counting the number of times the expressions used to determine α and β are called in the code:

$$N \geq 500 = \text{Convergence Window} \Rightarrow \text{Converged}. \tag{6}$$

If Equations (4) and (5) hold while Equation (6) fails (i.e., the number of calls is smaller than the convergence window), then α and β are corrected by interpolation:

$$p = 1.0 - \exp\left(0.5 \times \left(1.0 - \frac{1.0}{1.0 - \frac{N}{500}}\right)\right) \tag{7}$$

$$\hat{\alpha} = p \times \alpha_{\min} + (1.0 - p) \times \alpha_{\max} \tag{8}$$

$$\hat{\beta} = p \times \beta_{\min} + (1.0 - p) \times \beta_{\max} \tag{9}$$

with $\alpha_{\max}$ and $\beta_{\max}$ set as 0.05 and $\alpha_{\min}$ and $\beta_{\min}$ set as 0.003 for convergence of the moving average filter under the tested conditions. The one-way time-synchronized latency is as follows.

$$\text{Latency}_1 = \text{T}_{\text{Higher-level}\Rightarrow\text{Lower-level}} - \text{Offset}_{\text{Higher-level}\Rightarrow\text{Lower-level}} \quad (10)$$

$$\text{Latency}_2 = \text{T}_{\text{Lower-level}\Rightarrow\text{Higher-level}} - \text{Offset}_{\text{Lower-level}\Rightarrow\text{Higher-level}} \quad (11)$$

The respective bounds of 10 ms and 100 ms in Equations (4) and (5) are specific to the implementation in the PX4 platform, and are MAVlink defaults [12,18]. While they can be fine-tuned, this could affect the number of filter resets, in turn impacting estimation of the offset.

Table 1. ROS: FCU communication using MAVROS bridge.

MAVROS Topic	ROS Message	Rate (Hz)
mavros/vision_pose/pose [1]	PoseStamped	200
mavros/imu/data_raw [2]	Imu	200

[1] Published from companion computer, assessed at FCU. [2] Published from FCU, assessed at companion computer. The message and topic rate were custom-modified for comparison; the original rate was 50 Hz.

Table 2. ROS 2: FCU communication using MicroXRCE-DDS bridge.

Topic	uORB	Rate (Hz)
vehicle_visual_odometry [1]	VehicleOdometry	200
vehicle_imu [2]	VehicleImu	200

[1] Published from companion computer, assessed at FCU. [2] Published from FCU, assessed at companion computer. Custom definition.

2.10. Experimental Setup

A hardware-in-the-loop (HIL) setup was used for the latency assessment (Figure 5), based on the following: Pixhawk 4 flight controller FMU-V5 (Lower-level system), companion computer (Higher-level system), and HIL simulation host computer.

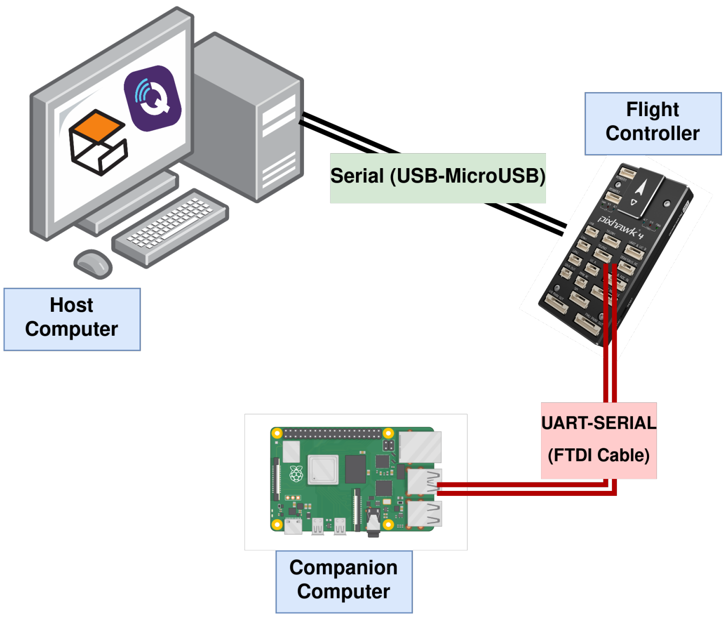

Figure 5. Hardware-in-the-loop setup for end-to-end latency measurements.

3. Results

3.1. Latency Comparison in the Flight Control Unit

The experiment sequence is described in detail in Section 2.9. After collecting time stamps from the Extended Kalman Filter (EKF2) module, the advantages of DDS network communication in UAVs become quite clear. A reduction in average latency is found; more importantly, the reduction of latency peaks with the use of the MicroXRCE–DDS bridge, which enables more accurate delay prediction for external sensor data fusion.

Figures 6 and 7 show results from a stress test at maximum companion computer CPU usage while prioritizing the ROS2 or ROS process in order to check the latency difference when transmitting an external odometry message. In this scenario, the latency is defined as the time elapsed between message creation at the ROS/ROS2 Node and message arrival at the EKF2 module, including the time synchronization process at the FCU, i.e., Algorithm 3 and Equation (10). Figure 8 shows the consequence of transmitting and receiving high-frequency topics in a low-resource device coupled to a slightly non-robust DDS network due to high companion computer CPU usage, namely, latency peaks, which can usually be mitigated using ROS2 Quality of Service settings [21]. For all assessments using the MicroXRCE-DDS–ROS 2 bridge, the publishers and subscribers used the following configuration settings:

- Reliability: *BEST_EFFORT*; the publisher attempts to deliver the maximum number of samples possible.
- History and Queue Size: *KEEP_LAST*; only one message is stored in the processing queue.
- Durability: *TRANSIENT_LOCAL*; the publishers are responsible for sending the last available message to newly discovered subscribers.

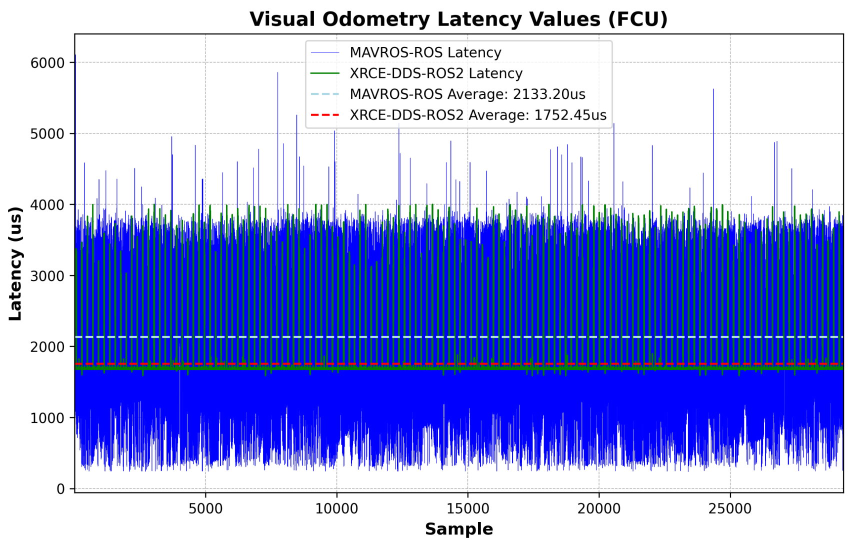

Figure 6. Visual odometry message: latency comparison, FCU.

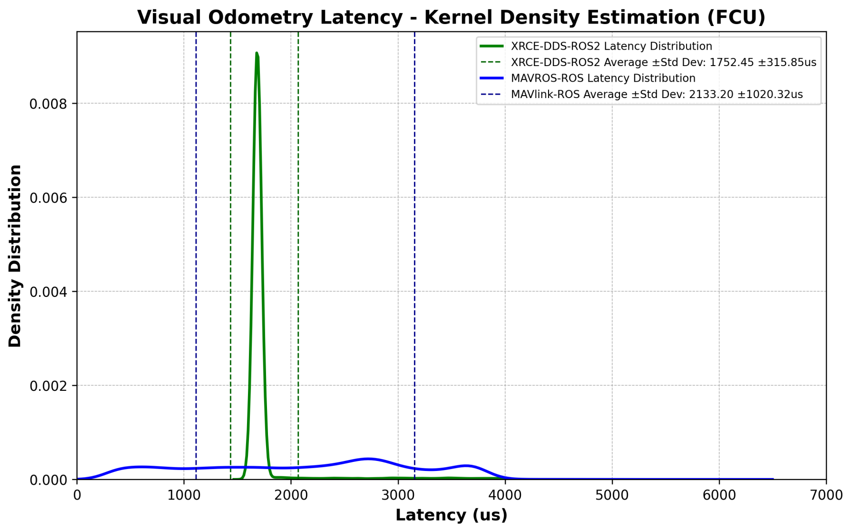

Figure 7. Visual odometry message: comparison of latency probability distribution, FCU.

Algorithm 3 Time Synchronization Algorithm (PX4 uORB/MAVlink [12,18])

```
 1: $N \leftarrow 0$
 2: $\text{Offset}_{Estimated} \leftarrow 0$
 3: $\text{Skew}_{Estimated} \leftarrow 0$
 4: $\hat{\alpha} \leftarrow \alpha_{\max}$                                  ▷ Initialize $\hat{\alpha}$ as $\alpha_{\max}$
 5: $\hat{\beta} \leftarrow \beta_{\max}$                                    ▷ Initialize $\hat{\beta}$ as $\beta_{\max}$
 6: high_deviation_count $\leftarrow 0$
 7: high_rtt_count $\leftarrow 0$
 8:
 9: procedure UPDATE($T_{\text{Current}}$, $T_{\text{Remote Stamp}}$, $T_{\text{Packet Creation}}$)
10:     if $T_{\text{Remote Stamp}} > 0$ then
11:         $\text{Offset}_{Observed} \leftarrow \frac{T_{\text{Packet Creation}} + T_{\text{Current}} - 2 \times (T_{\text{Remote Stamp}})}{2}$
12:         $T_{\text{RTT}} \leftarrow T_{\text{Current}} - T_{\text{Packet Creation}}$
13:         deviation $\leftarrow |\text{Offset}_{Estimated} - \text{Offset}_{Observed}|$
14:         if $T_{\text{RTT}} < 10$ ms then
15:             if est_sync_converged() $\wedge$ (deviation $> 100$ ms) then
16:                 high_deviation_count $\leftarrow$ high_deviation_count $+ 1$
17:                 if high_deviation_count $> 5$ then
18:                     RESET_FILTER
19:             else
20:                 if not EST_SYNC_CONVERGED then
21:                     progress $\leftarrow N/500$
22:                     $p \leftarrow 1 - \exp\left(0.5 \times (1 - \frac{1}{1-\text{progress}})\right)$
23:                     $\hat{\alpha} \leftarrow p \times \alpha_{\min} + (1-p) \times \alpha_{\max}$
24:                     $\hat{\beta} \leftarrow p \times \beta_{\min} + (1-p) \times \beta_{\max}$
25:                 else
26:                     $\hat{\alpha} \leftarrow \alpha_{\min}$
27:                     $\hat{\beta} \leftarrow \beta_{\min}$
28:                 ADD_SAMPLE($\text{Offset}_{Observed}$)
29:                 $N \leftarrow N + 1$
30:                 high_deviation_count $\leftarrow 0$
31:                 high_rtt_count $\leftarrow 0$
32:         else
33:             high_rtt_count $\leftarrow$ high_rtt_count $+ 1$
34:
35: procedure ADD_SAMPLE($\text{Offset}_{Observed}$)
36:     $\text{Offset}_{Estimated-1} \leftarrow \text{Offset}_{Estimated}$
37:     if $N == 0$ then
38:         $\text{Offset}_{Estimated} \leftarrow \text{Offset}_{Observed}$
39:     else
40:         $\text{Offset}_{Estimated} \leftarrow \hat{\alpha} \times \text{Offset}_{Observed} + (1-\hat{\alpha}) \times (\text{Offset}_{Estimated} + \text{Skew}_{\text{Estimated}})$
41:         $\text{Skew}_{\text{Estimated}} \leftarrow \hat{\beta} \times (\text{Offset}_{Estimated} - \text{Offset}_{Estimated-1}) + (1-\hat{\beta}) \times \text{Skew}_{\text{Estimated}}$
42:
43: procedure RESET_FILTER
44:     $N \leftarrow 0$
45:     $\text{Offset}_{Estimated} \leftarrow 0$
46:     $\text{Skew}_{\text{Estimated}} \leftarrow 0$
47:     $\hat{\alpha} \leftarrow \alpha_{\max}$
48:     $\hat{\beta} \leftarrow \beta_{\max}$
49:     high_deviation_count $\leftarrow 0$
50:     high_rtt_count $\leftarrow 0$
```

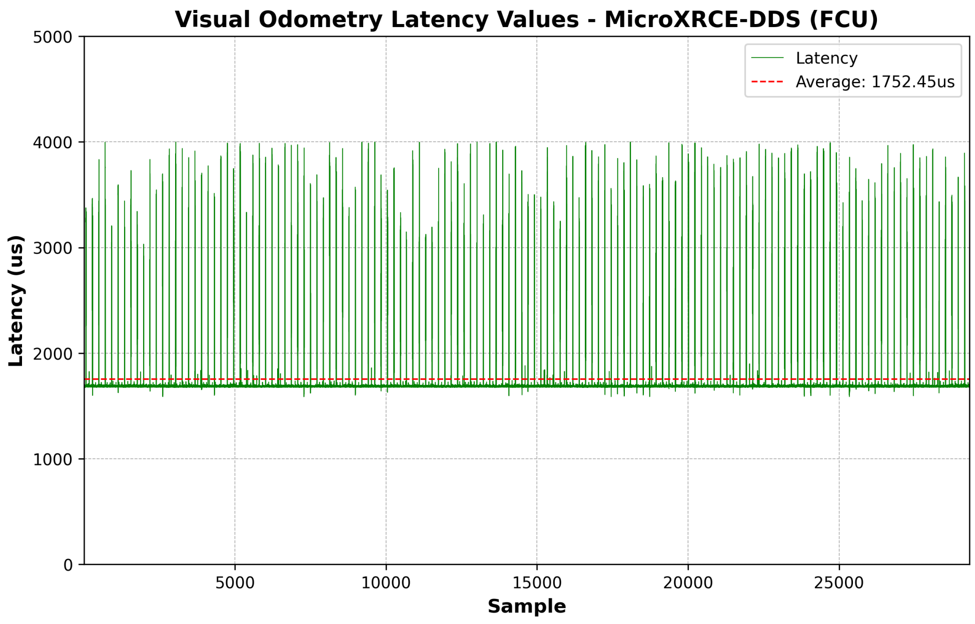

Figure 8. Visual odometry message: latency when using MicroXRCE-DDS–ROS 2.

3.2. Latency Comparison at the Companion Computer

The latency measurements at the companion computer follow the logic shown in Algorithms 1–3. Figure 9 shows the latency values with the elapsed time between an IMU sample and message arrival at the ROS/ROS2 Node, including the time synchronization correction at the companion computer, i.e., the estimated offset from Equation (2) after convergence is reached. Figure 10 shows the corresponding probability distribution. The messages are time-synchronized at the same node where the latency is assessed, as can be seen by comparing Figure 9 to Figure 6. There is a 349.01 us difference in average latency between the MAVROS–ROS and XRCE–DDS architectures.

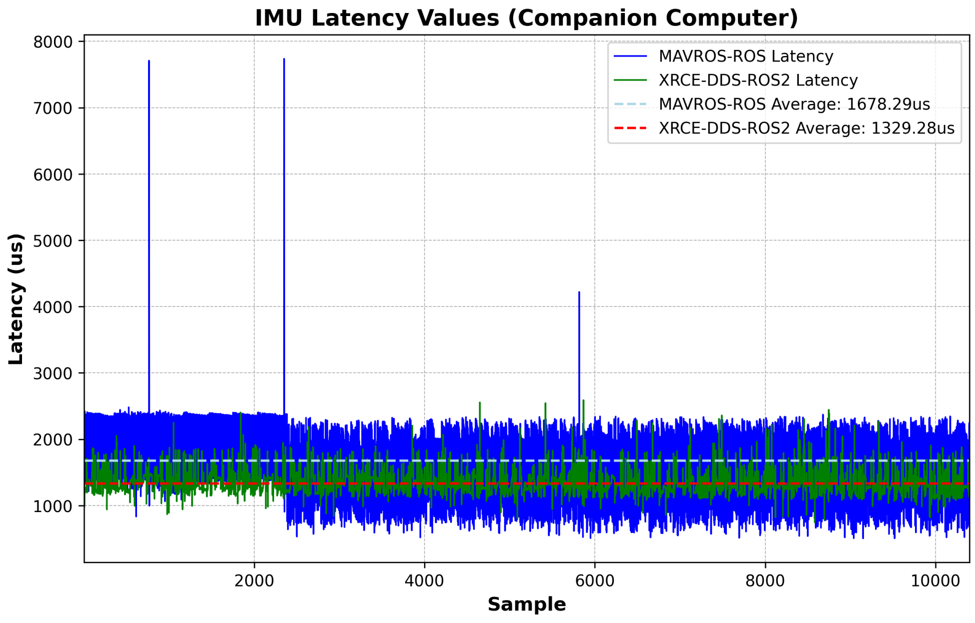

Figure 9. IMU message: latency comparison, companion computer.

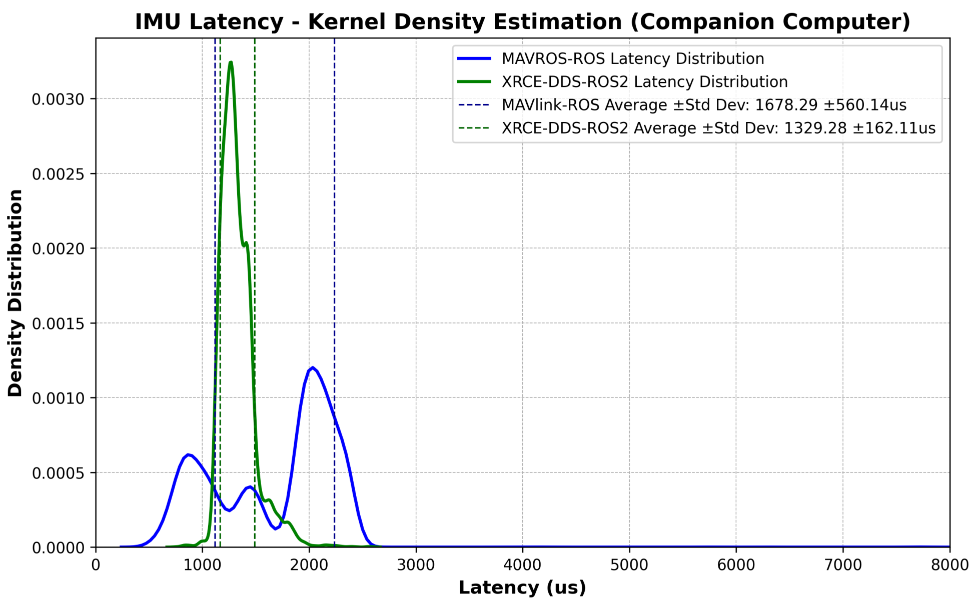

Figure 10. IMU message: comparison of latency probability distribution, companion computer.

3.3. Flight Controller CPU and RAM Utilization

The effect of the MicroXRCE-DDS–ROS2 bridge implementation in terms of FCU CPU and RAM usage is significant, and can provide insights into the minimum hardware requirements of future UAV missions. In this study, both MAVROS–ROS and MicroXRCE–DDS bridges were deployed using only those topics listed in Tables 1 and 2. The CPU demand decreases when using the MicroXRCE-DDS–ROS2 bridge. However, this is related to the capabilities of the chosen FCU hardware (in this case, the PX4 FCU V5, a popular flight control unit) and the number and loop rate of the topics transmitted to the FCU and companion computer. The jump in CPU/RAM usage shown below (Figure 11) corresponds to the beginning of operations, i.e., the start of the Gazebo simulator; a second jump can occur if the MicroXRCE–DDS bridge is not active at the beginning of operations or is not running at its fullest yet, i.e., when the ROS/ROS 2 routines have not yet started.

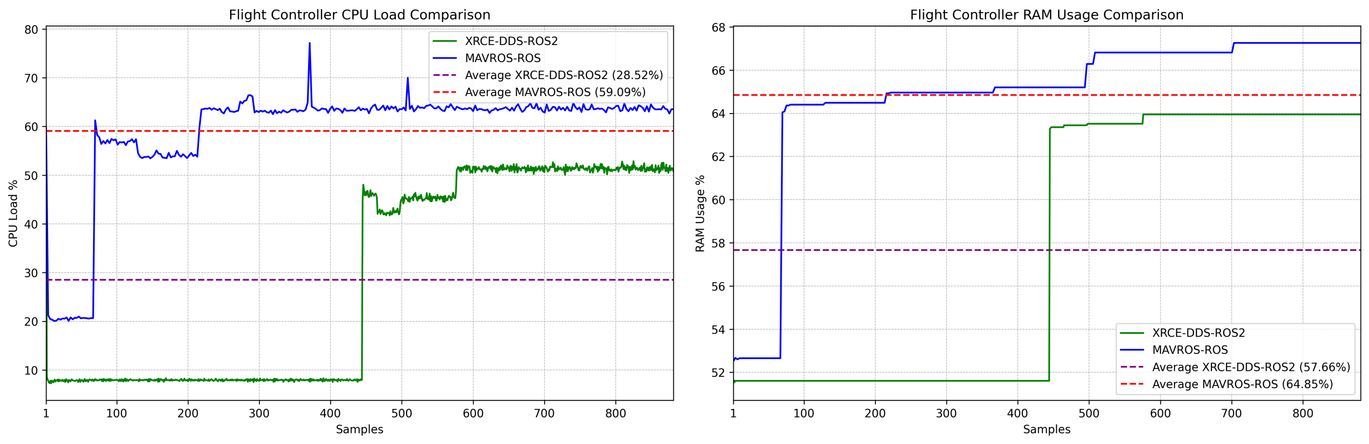

Figure 11. Pixhawk 4 FMU-V5: CPU and RAM load comparison during latency testing.

4. Discussion

4.1. Latency Reduction and Time Synchronization for Enhanced GNC in UAVs

This study highlights a crucial aspect of software development for complex UAV missions: the inherent latency in multi-layer system architectures. The results presented (Figures 6 and 9) illustrate that the XRCE-DDS–ROS 2 bridge is not only an effective method to reduce communication latency; it is a cornerstone for the development of enhanced GNC algorithms, as the performance of GNC algorithms relies on timely and reliable data exchange [22]. The proposed approach paves the way to develop synchronization corrections for internal and external sensor data within UAV systems, enabling higher performance by providing more accurate timing. The latency correction approach boils down to the estimation of the offset and clock skew, as respectively described in Equations (2) and (3). Note that time synchronization is not symmetrical; the latency in each direction needs to be corrected separately.

4.2. Time Synchronization Effects in UAV High-Level Layer (Companion Computer)

Synchronization is a fundamental aspect of real-time systems. Lack of accurate timekeeping introduces jitter in control signals and degrades the performance of sensor fusion estimation. The proposed synchronization approach allows ROS or ROS 2 nodes to combine their time-synchronized sensor data (data from a low-resource device) with sensor data at the higher-level layer (e.g., a camera connected to the companion computer), which enables improved sensor fusion at the companion level for operations such as Visual Inertial Odometry (VIO) and SLAM, thereby enhancing loosely coupled distributed UAV systems [3,4,23]. Implementing Algorithm 1 or Algorithm 2 in the high-level layer accounts for asymmetrical network paths and processing delays until data arrival at the GNC ROS/ROS 2 node by using reliable and updated offset and clock skew estimates as opposed to assuming a symmetrical network path with time synchronization at the communication bridge. Furthermore, the proposed implementation accounts for synchronization of the sample time instead of the message header time, i.e., the timestamp when the measurements (e.g., IMU or VIO) were collected, as opposed to the timestamp when the message was sent from the FCU to the companion computer (or vice-versa). A particular case of interest is where $T_{\text{Remote Timestamp}} = T_{\text{Sensor Sample}}$.

4.3. Event-Driven Communication between Flight Controller and Companion Computer

The modifications implemented in the PX4 firmware to support this study include a complete event-driven MicroXRCE-DDS–ROS 2 bridge, which allows incoming and outgoing messages to be consumed by requesting processes as soon as they are available. A direct consequence of this is a reduction in the latency's standard deviation, increasing offset predictability and lowering end-to-end delays.

4.4. Trade-Offs of Using DDS Networks in UAV Systems

The decentralized nature of a DDS network improves fault tolerance by improving the system's resilience against data outliers. Furthermore, robust security features can be employed in DDS (DDS-security). Security enhancements in ROS2 include authentication of nodes joining the DDS domain and encryption of data transmitted through ROS2 topics. DDS enables deployment of companion computers that use real-time operating systems, with potential to further reduce latency and improve synchronization [9,11,24,25].

On the other hand, even when coupled with MicroXRCE-DDS, DDS networks can cause spikes in the FCU's CPU and RAM utilization if the amount of topics or the message size to be transmitted and received is not properly monitored. This increase is only significant when the FCU publishes and subscribes to a large number of topics (10+), which is not common in UAV missions; nevertheless, it should be carefully monitored, as routine complexity and scalability are related to this issue.

4.5. Current Limitations of MAVlink and MAVROS

The MAVlink protocol, although widely used, is expected to become unsuitable in the future as an internal communication protocol between FCUs and companion computers. Message types are constantly evolving as new algorithms, sensors, and data acquisition technologies are developed. Adding a new MAVlink message and streaming it to and from the FCU using MAVROS is a demanding process, and requires extensive knowledge of the MAVlink and MAVROS libraries. ROS depends on a central node that acts as a look-up table with respect to the network nodes, which creates further increases in latency compared to ROS 2, where nodes are discovered automatically. The PX4 FCU firmware uses one type of message for internal communications, uORB, which can be used directly in ROS 2 nodes. On the other hand, MAVROS-ROS uses MAVlink messages that need to be converted to ROS messages on the companion computer side and to uORB in the FCU side.

The results presented here (Table 3) clearly show the increased latency created by the interface layer that converts MAVlink messages to the internal communication protocol in the PX4 FCU. Even with a MAVROS version capable of working in the ROS 2 framework, the delay created by the interface layer remains an issue for future UAV applications. MAVlink messages will continue to be used in several FCU applications, in particular those with well-establish functionalities such as radio transmission of telemetry packets. That said, the MAVlink community should continue to develop alternative communications solutions that can be used with uORB [12].

Table 3. Results summary.

Method	Path	FCU Resource Utilization (%) [1]	Average Latency and Standard Deviation (Microsec)
MicroXRCE-DDS–ROS 2	Companion Computer $\rightarrow$ Flight Controller Flight Controller $\rightarrow$ Companion Computer	CPU: 28.5 RAM: 57.6	1329 ± 162 1752 ± 315
MAVlink–MAVROS (ROS)	Companion Computer $\rightarrow$ Flight Controller Flight Controller $\rightarrow$ Companion Computer	CPU: 59.1 RAM: 64.9	1678 ± 560 2133 ± 1020

[1] Average values, Holybro Pixhawk 4 FMU-v5.

5. Conclusions

Our analysis and testing results show that MicroXRCE-DDS–ROS 2 is a better option as a communication bridge between a high-level companion computer and a low-resource FCU compared to the MAVROS–ROS bridge, having smaller communication latency and providing operation closer to real-time in GNC applications. The decentralized nature of DDS networks enables enhanced security features and risk reduction in AAUV missions. The proposed approach to time synchronization and latency correction improves performance in multi-layered AAUV systems, as it allows proper time alignment of sensor data from lower-level layers; algorithms in the higher-level layer have have access to data with more accurate timestamps. This can be particularly beneficial in sensor fusion for depth and visual-inertial odometry applications, where IMU measurements from the FCU need to be time-synced with camera frames at the companion computer for improved performance. An algorithm to estimate latency offset and clock skew based on an exponential moving average filter has been presented, providing a tool for latency estimation and correction that can be used by developers to improve synchronization of processes that rely on timely communication between an FCU and companion computer, such as the synchronization of lower-level sensor data at the higher-level layer. This addresses the challenges introduced in GNC applications by the non-deterministic nature of general-purpose operating systems and the inherent limitations of standard FCU hardware.

Future Work

- Assessment of the effect of high-level latency correction in the performance of GNC algorithms, in particular motion control.
- Assessment of scalability effects on latency and latency correction when using denser ROS 2 routines, i.e., when more DDS topics are shared between the FCU and companion computer.
- A latency comparison between the MicroXRCE-DDS–ROS 2 bridge and other emerging technologies, such as Zenoh-Pico–ROS 2 [26].

Author Contributions: Conceptualization, J.L.S.C., D.A., H.G., I.R.B. and J.P.I.; methodology, J.L.S.C., D.A. and H.G.; software, J.L.S.C. and D.A.; validation, J.L.S.C. and D.A.; formal analysis, J.L.S.C., D.A., H.G., I.R.B. and J.P.I.; supervision, H.G.; funding acquisition and project management, H.G., I.R.B. and J.P.I. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by NASA's Marshall Space Flight Center, Cooperative Agreement 80NSSC21M0262, Dual-Use Technology Development, CAN 2021.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data and codes are available upon reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AAUVs	Autonomous Aerial and Underwater Vehicles
DDS	Data Distribution Service
EKF	Extended Kalman Filter
FCU	Flight Controller Unit
IMU	Inertial Measurement Unit
RTPS	Real-Time Publish–Subscribe
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
UAV	Unmanned Aerial Vehicles
HIL	Hardware-in-the-loop

References

1. Bigazzi, L.; Basso, M.; Boni, E.; Innocenti, G.; Pieraccini, M. A Multilevel Architecture for Autonomous UAVs. *Drones* **2021**, *5*, 55. [CrossRef]
2. Dehnavi, S.; Goswami, D.; Koedam, M.; Nelson, A.; Goossens, K. Modeling, implementation, and analysis of XRCE-DDS applications in distributed multi-processor real-time embedded systems. In Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 1–5 February 2021; pp. 1148–1151. [CrossRef]
3. Skog, I.; Handel, P. Time Synchronization Errors in Loosely Coupled GPS-Aided Inertial Navigation Systems. *IEEE Trans. Intell. Transp. Syst.* **2011**, *12*, 1014–1023. [CrossRef]
4. Di Pietra, V.; Dabove, P.; Piras, M. Loosely Coupled GNSS and UWB with INS Integration for Indoor/Outdoor Pedestrian Navigation. *Sensors* **2020**, *20*, 6292. [CrossRef] [PubMed]
5. Bigazzi, L.; Basso, M.; Gherardini, S.; Innocenti, G. Mitigating latency problems in vision-based autonomous UAVs. In Proceedings of the 2021 29th Mediterranean Conference on Control and Automation (MED), Puglia, Italy, 22–25 June 2021; pp. 1203–1208. [CrossRef]
6. Virginio, R.; Adiprawita, W. Utilization of MAVROS for electro optical seeker subsystem in the design of ROS based architecture for autonomous guided platform. *AIP Conf. Proc.* **2021**, *2366*, 060002.
7. Meier, L.; Honegger, D.; Pollefeys, M. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 6235–6240. [CrossRef]
8. Koubâa, A.; Allouch, A.; Alajlan, M.; Javed, Y.; Belghith, A.; Khalgui, M. Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey. *IEEE Access* **2019**, *7*, 87658–87680. [CrossRef]
9. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; Ng, A. ROS: An open-source Robot Operating System. In Proceedings of the ICRA 2009, Kobe, Japan, 2–17 May 2009.
10. Macenski, S.; Foote, T.; Gerkey, B.; Lalancette, C.; Woodall, W. Robot Operating System 2: Design, architecture, and uses in the wild. *Sci. Robot.* **2022**, *7*, eabm6074. [CrossRef] [PubMed]
11. Macenski, S.; Soragna, A.; Carroll, M.; Ge, Z. Impact of ROS 2 Node Composition in Robotic Systems. *IEEE Robot. Auton. Lett. (RA-L)* **2023**, *8*, 3996–4003. [CrossRef]
12. MAVLink: Micro Air Vehicle Communication Protocol. 2023. Available online: https://mavlink.io/en/ (accessed on 23 September 2023).
13. MAVROS: MAVLink Extendable Communication Node for ROS. 2023. Available online: https://github.com/mavlink/mavros (accessed on 23 September 2023).

14. Solpan, S.; Kucuk, K. DDS-XRCE Standard Performance Evaluation of Different Communication Scenarios in IoT Technologies. *EAI Endorsed Trans. Internet Things* **2022**, *8*, e1. [CrossRef]
15. Micro XRCE-DDS: EProsima Micro XRCE-DDS Documentation. 2023. Available online: https://micro-xrce-dds.docs.eprosima.com (accessed on 23 September 2023).
16. Koenig, N.; Howard, A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), Sendai, Japan, 28 September–2 October 2004; Volume 3, pp. 2149–2154. [CrossRef]
17. Ramirez-Atencia, C.; Camacho, D. Extending QGroundControl for Automated Mission Planning of UAVs. *Sensors* **2018**, *18*, 2339. [CrossRef] [PubMed]
18. PX4: Open Source Autopilot. 2023. Available online: https://www.px4.io/ (accessed on 23 September 2023).
19. Chen, H.; Chen, S. A moving average based filtering system with its application to real-time QRS detection. In Proceedings of the Computers in Cardiology, Thessaloniki, Greece, 21–24 September 2003; pp. 585–588. [CrossRef]
20. Silva, J.G.; Aquino Limaverde Filho, J.O.D.; Feitosa Fortaleza, E.L. Adaptive Extended Kalman Filter using Exponencial Moving Average. *IFAC PapersOnLine* **2018**, *51*, 208–211. [CrossRef]
21. Kronauer, T.; Pohlmann, J.; Matthé, M.; Smejkal, T.; Fettweis, G.P. Latency Overhead of ROS2 for Modular Time-Critical Systems. *arXiv* **2021**, arXiv:2101.02074. [CrossRef]
22. Silva Cotta, J.L.; Rakoczy, J.; Gutierrez, H. Precision landing comparison between smartphone video guidance sensor and IRLock by hardware-in-the-loop emulation. *CEAS Space J.* **2023**. [CrossRef]
23. Li, P.; Cao, J.; Liang, D. UAV-BS Formation Control Method Based on Loose Coupling Structure. *IEEE Access* **2022**, *10*, 88330–88339. [CrossRef]
24. Du, J.; Gao, C.; Feng, T. Formal Safety Assessment and Improvement of DDS Protocol for Industrial Data Distribution Service. *Future Internet* **2023**, *15*, 24. [CrossRef]
25. Vilches, V.M.; White, R.; Caiazza, G.; Arguedas, M. SROS2: Usable Cyber Security Tools for ROS 2. *arXiv* **2022**, arXiv:2208.02615. [CrossRef]
26. Corsaro, A.; Cominardi, L.; Hecart, O.; Baldoni, G.; Enoch, J.; Avital, P.; Loudet, J.; Guimarães, C.; Ilyin, M.; Bannov, D. Zenoh: Unifying Communication, Storage and Computation from the Cloud to the Microcontroller. *DSD*. September 2023. Available online: https://www.researchgate.net/publication/373757741_Zenoh_Unifying_Communication_Storage_and_Computation_from_the_Cloud_to_the_Microcontroller (accessed on 12 October 2023).