Low-Cost Open-Source Real-Time Communication in Industrial IoT: Using the Raspberry Pi 5 with OPC UA over TSN*

Jonathan Lukas Mandl[1], Olaf Saßnick[2], and Thomas Rosenstatter[3]

***Abstract*— Industry 4.0 demands optimized industrial process control and enhanced data permeability, necessitating more capable edge-level hardware. While personal computers with real-time Linux operating systems offer ample computing power at low cost, they suffer from limitations in low-level connectivity, standardized compact form factors, and uncertain long-term supplies. This paper explores the Raspberry Pi 5 as a viable alternative, highlighting its excellent low-level connectivity, compact form factor, and guaranteed long-term availability until 2036.**

This study investigates the Raspberry Pi 5's performance in real-time communication scenarios using Open Platform Communications Unified Architecture (OPC UA) over Time-Sensitive Networking (TSN), a critical requirement for industrial applications. By outlining necessary modifications to the Raspberry Pi 5 and its Linux kernel, we enable real-time communication via TSN. Performance measurements in an OPC UA PubSub scenario are then compared with industrial PCs, highlighting the Raspberry Pi 5's potential as an alternative edge device.

***Index Terms*— Industry 4.0, industrial communication, time-sensitive networking, commodity hardware**

I. Introduction

The rising need for optimized industrial process control and data permeability driven by the Industry 4.0 initiative necessitates more capable hardware at the edge-level. While personal computers with real-time Linux operating systems could provide plenty of computing power at a low cost, they come with downsides, namely: (i) a lack of low-level connectivity, (ii) limited options for standardized compact form factors, and (iii) uncertain long-term supplies.

Connectivity. Basic interfaces are missing to interface sensors, such as RS485, One-Wire, and also General-Purpose Input/Output (GPIO). One can resort to USB adapters, which, however, contribute to latency and introduce new sources of failure.

Form factor. The smallest widespread standardized form factor, Mini-ITX, still occupies a considerable amount of space and is therefore unsuitable for numerous applications.

Long-term supplies. Although personal computers are readily available, new generations are typically introduced every 1-2 years, phasing out the previous models. Additionally, unannounced hardware revisions can significantly impact usability in industrial systems. Once a model has been thoroughly tested, it is undesirable to change and retest it.

Motivation. Considering all three options, the Raspberry Pi 5 is a viable alternative. It offers excellent low-level connectivity in a compact form factor and guarantees long-term supplies, as it will remain in production until 2036. Additionally, the Compute Module 5 [12], which features a high-density perpendicular connector, uses the same hardware. This module simplifies the hardware design process for custom solutions, allowing for a 2-layer printed circuit board. Since the NIC and processor are identical, all findings in this work are applicable to the Compute Module 5.

The Raspberry Pi 5 presents a strong candidate, given the benefits mentioned earlier. Especially in resource-constrained environments like mobile robotics, where the form factor and energy efficiency are crucial, this system could prove highly beneficial. However, it is essential to evaluate its performance in real-time communication scenarios, as this is a critical requirement for industrial applications.

Contribution. In this paper we investigate two research questions (RQs) focusing on the Raspberry Pi 5's performance with OPC UA Time-Sensitive Networking (TSN) and further compare it to industrial PCs.

- **RQ1.** How does the Raspberry Pi 5 perform in real-time communication tasks using OPC UA over TSN?
- **RQ2.** How does the Raspberry Pi 5's performance as a real-time OPC UA node compare to industrial PCs?

By addressing the above-mentioned research questions, we first outline the necessary modifications to the Raspberry Pi 5 and its Linux kernel. These modifications enable real-time communication via TSN. After setting up the Raspberry Pi 5 devices, we measure their performance in a OPC UA PubSub scenario (*RQ1*). We compare this performance with industrial PCs, which serve as a baseline. This comparison highlights the Pi 5's potential as an alternative edge device (*RQ2*).

II. Background

A. Raspberry Pi 5 Hardware

The Raspberry Pi 5, hereafter referred to as the Pi 5, is a single-board computer from the Raspberry Pi Foundation. The term single-board computer describes the hardware, as it is integrated onto a single circuit board. Compared to its predecessor, the Pi 4, the Raspberry Pi Foundation reports a two to threefold increase in performance. Furthermore, the

*The industrial computers used in this work were provided by the Open Source Automation Development Lab (OSADL) eG.

[1]Jonathan Lukas Mandl is a master student in Industrial Informatics & Robotics at the Salzburg University of Applied Sciences, AT, jmandl.iirb-m2024@fh-salzburg.ac.at

[2]Olaf Saßnick is with Josef Ressel Centre for Intelligent and Secure Industrial Automation, Salzburg University of Applied Sciences, AT, olaf.sassnick@fh-salzburg.ac.at

[3]Thomas Rosenstatter is with Josef Ressel Centre for Intelligent and Secure Industrial Automation, Salzburg University of Applied Sciences, AT, thomas.rosenstatter@fh-salzburg.ac.at

https://doi.org/10.34749/3061-0710.2025.10

cache layout is improved on the Pi 5. L3 cache is added and the L2 cache is changed from shared to per-core cache. This helps with the isolation of CPU cores and their caches, improving timing determinism in real-time systems [15].

For this work, the Ethernet peripherals are more critical than the chosen processor. The Pi 5 features, in comparison to its predecessor, utilizes an in-house developed south bridge that connects the Ethernet MAC peripherals via PCI-E. This setup includes the Cadence Gigabit Ethernet MAC design of type GEM_GXL 1p09, which supports IEEE 1588 for precise time synchronization, a standard used in Precision Time Protocols (PTPs) applications. This design allows for time-stamping of packets, which is crucial for applications requiring synchronized timing, such as industrial automation and real-time communication systems [14].

B. Precision Time Protocol

The PTP protocol enables precise synchronization of the clocks of multiple devices in the same network. It uses a master slave topology to determine the exact travel delay between two devices which is then used to synchronize the slave clock to the master clock. Due to the fundamental properties of the protocol, hardware support from the Ethernet MAC is needed for precise synchronization. In this case the Ethernet MAC uses an internal clock inside the Network Interface Card (NIC) to timestamp incoming and outgoing packets. While software timestamping happens inside the Linux kernel and therefore adds operating system latencies. The Linux implementation uses two services: `ptp4l` is used for performing the PTP protocol, `phc2sys` synchronizes the system clock to the PTP hardware clock used for timestamping [5].

C. Time-Sensitive Networking

Time-Sensitive Networking is a series of standards that add to the Ethernet standard to improve the real-time performance of Ethernet. It mainly addresses two important features needed for real-time applications: time synchronization and traffic shaping [2].

Time synchronization and more specifically PTP, synchronizes the clocks of the nodes in a TSN network. Clock synchronization is needed to ensure that time driven communication can be correctly performed. Shaping algorithms also depend on a common and accurate time.

This work focuses solely on the time synchronization part of the TSN standard. Shaping is largely dependent on the implementing software. In the experimental setup there are only two TSN capable devices and no additional non-real-time traffic which would make shaping necessary.

D. PREEMPT_RT-patched Linux kernel

The experiments in this work were done on the version 6.6.23 and 6.6.78 of the Linux kernel and therefore needed the PREEMPT_RT patches for real-time support. The patches 6.6.23-rt28 and 6.6.78-rt51 were used. Even though dynamic preemption in the Linux kernel reduces the latency of tasks a lot, full preemption is needed to minimize latency and improve consistency of process wake up times. The patch achieves this mostly by removing or altering non preemptable kernel code. Another significant change is the adoption of threaded interrupt handling, which allows higher priority interrupts to interrupt lower priority interrupt handlers.

Without threaded interrupt handlers the latency of network interrupts would be less predictable and generally higher. The real-time performance of the patched Linux kernel was tested with the `cyclictest` utility and a synthetic system load. The *cyclictest* utility from the *rt-tests* package uses `clock_nanosleep` to suspend a measuring thread. By calculating the deviation from the expected wake-up time, the utility calculates the systems latencies [19]. Furthermore, the load generating tool *stress* was used for synthetic CPU and memory load. This load ensures that the Linux kernel has to be preempted during the test [18].

E. Open Platform Communications Unified Architecture

OPC UA represents an evolution of the OPC standard which consolidates the previous OPC Classic specifications into a platform-independent framework. OPC UA is a data exchange standard that supports a variety of functions, including data transfer, method calls, and other capabilities. The OPC Foundation describes its primary use case as enabling communication from machine-to-machine and machine-to-enterprise, as well as bridging the two. A key feature is its ability to semantically describe data and organize it within complex, object-oriented structures. While OPC UA offers a wide range of functionalities, this paper focuses on its role as a foundation for data transfer.

Furthermore, the OPC UA application used in this work leverages the PubSub mechanism to transfer data between nodes. The code is provided by Pfrommer et al. [10] and is available in the open62541 library until version 1.4

III. Related Work

The Open Source Automation Development Lab eG (OSADL) is a laboratory dedicated to providing open-source software solutions for industrial systems. They have projects focused on real-time networking with various protocols, including OPC UA PubSub over TSN. In contrast to this paper, their research focuses on the feasibility of open-source software in industrial applications. Measurements from their project [7] can be compared with the results presented in this paper.

The work by Ulbricht et al. [17] describes TSN-FlexTest, a flexible testbed for TSN measurements. This testbed utilizes commodity off-the-shelf hardware and focuses on the TSN communication itself. They also use the same NIC (Intel I210) as the computers for our baseline comparison.

While other studies have utilized the Raspberry Pi in industrial settings using OPC UA, they do not focus on OPC UA over TSN (e.g., [6], [4]). An exception is the work by Reddy et al. [16], who employ a Raspberry Pi 3 only as a subscriber in a TSN network, although they do not provide performance details.

We have not found any comparable research investigating the potential of the Pi 5 computer respectively the Compute

https://doi.org/10.34749/3061-0710.2025.10

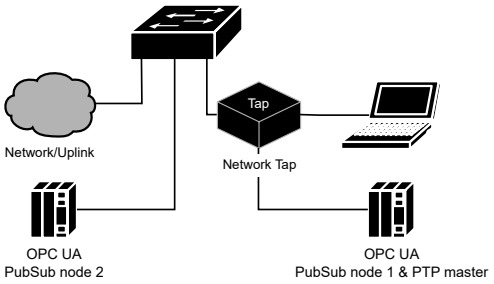

Fig. 1. Network topology of the experiment setup.

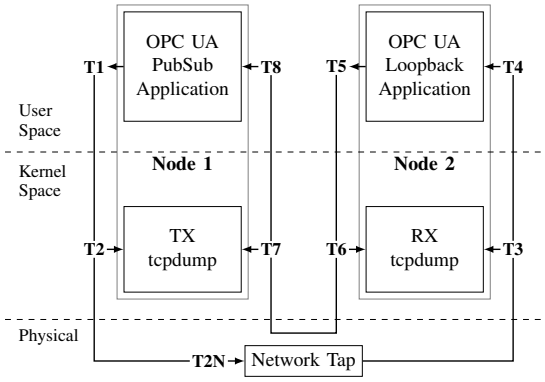

Fig. 2. Overview of the test points used for logging processing times, based on [10], with the addition of **T2N**, provided via a network tap.

Module 5 as a replacement for industrial computers in real-time networking using TSN.

IV. Experiment Setup

The setup comprises two Pi 5 computers, a consumer grade network switch and a network tap. A dedicated network tap is utilized to measure and improve the accuracy of packet timestamps as well as to include jitter of hardware latencies of the computers used. The Pi 5 computers are configured as OPC UA PubSub nodes. Both act as a subscriber and a publisher as presented in the TSN example program[1] in the open62541 library (until Version 1.4) [10]. Moreover, for clock synchronization between the two TSN nodes, one of the OPC UA PubSub nodes acts as a PTP master for the other node.

A. Hardware Details

In addition to this setup, two industrial PCs are used for baseline measurements. These PCs are equipped with Intel i210 NICs, which support optimization for TSN networks. The two primary settings for tuning are: defining a launch time for packets (`SO_TXTIME`) and use of multiple hardware queues for different priority packets. Additionally, in the baseline measurements, one industry PC was also used as the PTP master. This was done in order to remove all Pi 5 out of the tests and should not change the outcome. This difference also highlights that the industrial PCs can operate at very short cycle times with low jitter, as shown in Section V.

It should be noted that the used layer 2 switch does not support all features of the TSN specification. But this reinforces the purpose of this paper to use commodity off-the-shelf hardware for evaluating the performance of real-time networking. Figure 1 illustrates the testing setup. The illustrated OPC-UA nodes are the Pi 5 computers respectively the industrial PCs.

Following is a list of the hardware used in the test setup:

- ProfiShark 1G network tap
- TP-Link TL-SG105E 5 port unmanaged switch
- Raspberry Pi 5 with 4GB memory

For baseline testing with industrial PCs instead of the Pi 5 computers:

- Schubert Prime Box Pico with Intel Atom Processor E3950 and two Intel i210 NICs

B. System Modifications to Pi 5

To further improve the real-time networking performance on the Pi 5 several modifications were applied to the system.

- *Deactivating Energy-Efficient Ethernet:* The Energy-Efficient-Ethernet protocol can be deactivated via an entry in the boot configuration, which means that the network card can no longer switch to an energy-saving mode when there is no network traffic. This energy-saving mode can result in added latencies because of the wake-up time of the NIC [13].
- *Ethernet coalescence:* Even though the Ethernet MAC on the Pi 5 does not support all features which could improve performance for real-time networking, some settings are available. `rx-usecs` and `tx-usecs` are available settings. Both settings should be lowered to reduce the amount of microseconds the Ethernet NIC waits until triggering an interrupt for packet processing in the kernel. A value of `0` leads to instant interrupts, but also generates an interrupt for every incoming packet. In this experiment the value was set to 0 microseconds to improve real-time networking performance. As a trade-off, more interrupts are generated, which lead to more interrupt handlers and more CPU load.
- *Isolating CPU cores:* By default the Linux kernel distributes processes across all CPU cores. This behavior is unwanted in real-time systems, because processes should have dedicated CPU cores that should not be shared with other processes. To isolate the cores, the `isolcpus` kernel command line argument can be used. To further leverage the now single process CPU cores, the scheduler is set to be tick-free on the specified cores with `nohz_full`. This setting in turn also offloads all Read-Copy-Update (RCU) callbacks to other cores [1].

V. Results

For performance analysis, two types of measurements are evaluated: network capture from the network tap and

[1] https://github.com/open62541/open62541/tree/v1.3.10/examples/pubsub_realtime (2025-02-06)

https://doi.org/10.34749/3061-0710.2025.10

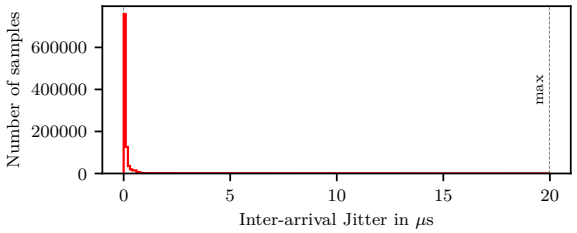

Fig. 3. Jitter on test point T1 with 1 ms cycle time on the Raspberry Pi 5.

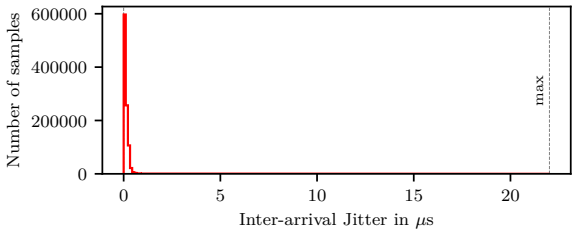

Fig. 4. Jitter on test point T1 with 250 µs cycle time on the Raspberry Pi 5.

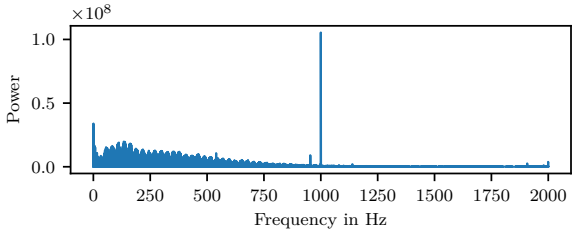

Fig. 5. Frequency spectrum of jitter on test point T1 for 250 µs cycle time on the Raspberry Pi 5.

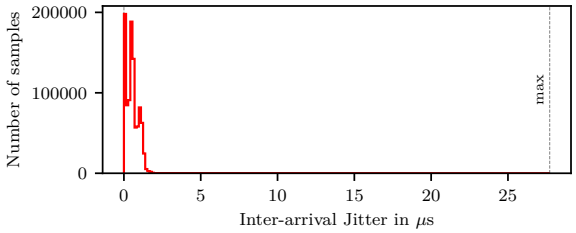

Fig. 6. Jitter of the network packets on test point T2N with 250 µs cycle time on the Raspberry Pi 5.

measurements from test points inside the application (see Figure 2).

The main difference between these two is, that the former includes latencies from the Linux kernel and Ethernet hardware. Therefore, the latter can be used to measure the real-time performance of a specific Linux kernel on the device and to indicate at which cycle times the OPC UA PubSub Protocol over TSN can be used. Even though they are different measurements and should be treated as such, the main performance indicator is shared. Jitter of the cycle time indicates irregular and unpredictable latencies in both measurement methods. Theoretically, latencies in the Linux kernel and Ethernet stack should add onto the processing and process wake-up latencies from within the application. But features in the Linux kernel implemented for better real-time networking capabilities, such as traffic control and the aforementioned `SO_TXTIME` allow for compensation and better handling of latencies. Therefore, results from the industrial PCs can have a different relation between the two measurement methods.

In the TSN example taken from the open62541 library are multiple test points which can be used to log the exact time a packet is processed. Figure 2 provides an overview of the location of these test points.

The differences at the transmission points can be explained by differences in the operating system and processor as well as hardware architecture. Due to the different processor architecture, the operating system could not directly be duplicated from the Pi 5 to the industrial PCs. Adding onto the difference in processor architecture is that the Linux kernel version differs between the two systems, because the Linux kernel specifically adapted for the Pi 5 was used.

The jitter as defined in [11] is the variation in forwarding delay between consecutive packets

$$J = |D_i - D_{i-1}|, \tag{1}$$

where D_i is the forwarding delay of a given packet. Given only the absolute timestamps of packet transmission in our setup, it is not possible to calculate the forwarding delay. As an alternative, the jitter is calculated based on the variation of the difference in timestamps via

$$J = |(T_{i+1} - T_i) - (T_i - T_{i-1})|, \tag{2}$$

where T_i is the absolute timestamp of a given packet. The calculation of jitter outside the application logging is not trivial due to the network tap not being synchronized with PTP. The clock drift of the recorded network tap timestamps is corrected to allow for a meaningful comparison with PTP synchronized timestamps. A linear clock drift can be corrected by computing a scaling factor

$$s = \frac{T_c}{\frac{1}{n}\sum_{i=1}^{n}(T_i - T_{i-1})}, \tag{3}$$

where T_c is the configured cycle time. This scaling factor is then used to adjust each non-synchronized timestamp. The remaining non-linear clock drift cannot be compensated without compromising the integrity of the results. However, due to the short recording duration, the non-linear clock drift per packet is assumed to be negligible.

The results in Figures 3 and 4 show that the jitter is very similar between the cycle time of 1 ms and 250 µs.

To further analyze unwanted cyclic jitter sources in the system the frequency spectrum of the jitter on point T1 was calculated as shown in Figure 5. From the frequency spectrum it is clear that the only cyclic jitter source are the Linux kernel timer interrupts, which were configured to 1kHz

https://doi.org/10.34749/3061-0710.2025.10

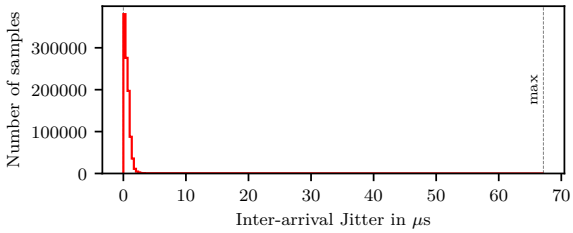

Fig. 7. Jitter on measuring point on test point T1 with 250 µs cycle time on the industrial PC.

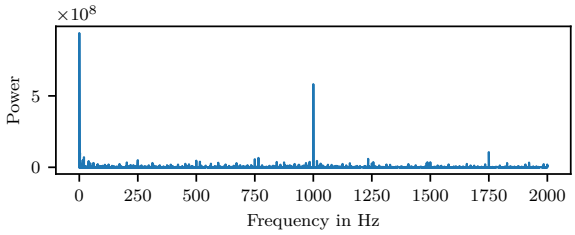

Fig. 8. Frequency spectrum of jitter on test point T1 for 250 µs cycle time on the industrial PC.

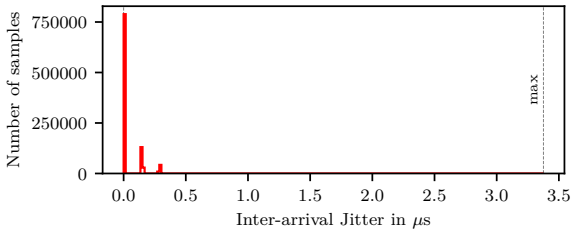

Fig. 9. Jitter of the network packets on test point T2N with 250 µs cycle time on the industrial PCs.

TABLE I
SUMMARY OF THE MEASURED JITTER DISTRIBUTIONS ACROSS DIFFERENT TEST POINTS AND HARDWARE CONFIGURATIONS.

Device	**Test Point**	**Rate** (μs)	**Median** (μs)	**25%** (μs)	**75%** (μs)	**99%** (μs)	**Max** (μs)
RPi5	T1	250	0.075	0.037	0.166	0.610	22.016
RPi5	T1	1000	0.055	0.019	0.093	1.592	19.980
IPC	T1	250	0.466	0.213	0.810	2.158	67.140
RPi5	T2N	250	0.512	0.208	0.792	1.416	27.719
IPC	T2N	250	0.000	0.000	0.008	0.288	3.376

on both systems. This behavior is expected, and the added latencies cannot be completely removed, only mitigated (see Section IV).

After measuring the performance of the real-time Linux kernel on the Pi 5 the behavior is expected and maximum jitter values do not exceed 22.02 µs with 250 µs cycle time. To establish a baseline the same measurements were conducted on the industrial PCs.

In Figure 7 it can be seen that the industrial PCs produce higher maximum jitter values than the Pi 5s. While testing the real-time kernel this behavior has also been observed, which indicates further that the jitter on measuring point T1 closely represents the jitter of thread wake up times. As already mentioned there is a change in processor architecture and hardware topology including but not limited to changes in cache layout between both systems [15]. These changes can have a big impact on real-time performance. Furthermore, Figure 8 shows that there is a low frequency jitter source visible, but the dominant frequency is still caused by Linux kernel timer interrupts. The low frequency jitter sources may be the System management interrupt (SMI) of Intel processors, which cause unavoidable latencies in the system. Using the `hwlatdetect`-utility (from the `rt-tests` package) the SMI interrupts were measured to cause 12 µs of latency with a frequency of 1 Hz [19].

It has to be noted that the minimum cycle time successfully used was 250 µs, lower cycle times lead to crashes in the application on both systems. Therefore, this does not indicate that lower cycle times are not possible on the exact hardware used.

To further compare the performance to the industrial PCs, the network capture is analyzed. As previously mentioned, the network tap adds accurate timestamps to incoming packets to prevent inaccurate timestamps from the computer running the recording software. This does mean that the network tap is not synchronized to the clock of the PTP master. Therefore, the non-linear clock drift between the two clocks cannot be deducted from the packet timestamps. It is also not possible to calculate the absolute jitter of a packet from T1 to the network due to the missing common time. The jitter values in the following plots are calculated from the difference between two timestamps, which mostly removes clock drift from the results.

As Figures 6 and 9 illustrate, there is a significant difference in jitter between the two systems when comparing jitter of the network packets. The Pi 5 procures similar jitter values on the network as on measuring point T1. In contrast, the industrial PC is able to reduce the maxmium jitter of network packets down to 3.38 µs. This reduction in jitter can solely be contributed to the `SO_TXTIME` feature of the Intel i210 NIC.

VI. Discussion and Future Work

The results clearly show that industrial PCs have better real-time networking performance than the Pi 5, which was expected. The main reason is the Intel I210 NIC, which can be fine-tuned for a specific network topology and traffic to achieve better timing. Note that this finding only applies to the industrial PCs used in this experiment as there are various configurations with different NICs. The identified required features are available in Intel I210 and similar featured NICs [3].

The Pi 5, on the other hand, does not use an equally featured NIC and therefore offers reduced performance in this application. It has to be decided on a case-by-case basis whether this system is sufficient. For software based real-time networking the jitter can be compared to different testing setups including Ethercat and Powerlink from OSADL. These tests are available at [8] and [9]. At the time of writing

https://doi.org/10.34749/3061-0710.2025.10

the maximum jitter of 5 minute intervals over 24 hours on the Powerlink setup fluctuated from 31 µs down to 12 µs on a 500 µs cycle time. This setup does not utilize an Intel i210 NIC [9]. If the maximum jitter of network packets on the Pi 5 does not increase on long-term measurements, the real-time networking performance of the Pi 5 may suffice for the task of a software based Powerlink master.

To strengthen our findings, a long-term experiment using the testing setup described in this paper should be done. Furthermore, an OT device leveraging the real-time networking capabilities of the Pi 5 needs to be implemented to ensure the findings also apply outside the testing environment. Such an environment introduces non-real-time packets and therefore tests the resiliency possible on the NIC of the Pi 5.

VII. Conclusion

In this paper, we investigated the feasibility of using a Raspberry Pi 5 computer as a replacement of an industrial computer for real time communication. The Pi 5 computer was selected for this purpose due to its excellent low-level connectivity, compact form factor and the guaranteed long-term availability until 2036.

The findings in regard to the performance of the Pi 5 to perform real-time communication tasks using OPC UA over Time-Sensitive Networking (TSN) (*RQ1*) indicate that the Pi 5 indeed can be used as an OPC UA Node for real-time communication using the PubSub mechanism. However, we have to note that the results of this work are only applicable to small TSN networks with no additional network traffic and cross load. Furthermore, the lack of the NIC features may degrade the real-time networking performance.

The performance analysis of the Pi 5 to an industrial P (*RQ2*) showed that the latter using an Intel i210 NIC improves the jitter behavior of network packets utilizing the `SO_TXTIME` option. In detail, the maximum jitter of 3.38 µs is significantly reduced when compared to the maximum jitter on the Pi 5, which is 27.72 µs. One should also take into account that, industrial PCs typically offer other advanced features that enable hardware traffic control for packets with different priorities, which is important in mixed or large TSN networks.

Although the Pi 5 does not achieve a comparable result in packet jitter in the network, it offers other advantages over industrial PCs. The powerful and efficient single-board computer provides a vast range of interfaces including GPIO, Camera Serial Interface and Display Serial Interface. Furthermore, alternative variants like the Compute Module 5 make custom real-time networking hardware more accessible [14]. Moreover, we showed that the Pi 5 presents a compelling alternative, facilitating rapid prototyping of applications and significantly reducing associated costs for laboratory setups only requiring small-scale networks.

Acknowledgment

The financial support by the Christian Doppler Research Association, the Austrian Federal Ministry for Digital and Economic Affairs and the Federal State of Salzburg is gratefully acknowledged.

References

[1] "The Linux kernel documentation," accessed: 2025-03-04. [Online]. Available: docs.kernel.org

[2] IEEE 802.1 Time-Sensitive Networking Task Group, "Time-sensitive networking (TSN) task group," 2024, accessed: 2025-03-04. [Online]. Available: https://1.ieee802.org/tsn/

[3] Intel Corporation, "Intel® Ethernet controller I210 datasheet," jan 2021, revision Number: 3.7. [Online]. Available: https://www.intel.de/content/www/de/de/products/sku/64402/intel-ethernet-controller-i210it/specifications.html

[4] M. Ladegourdie and J. Kua, "Performance analysis of OPC UA for industrial interoperability towards industry 4.0," *IoT*, vol. 3, no. 4, pp. 507–525, 2022. [Online]. Available: https://www.mdpi.com/2624-831X/3/4/27

[5] Linux PTP Project, "The Linux PTP project," 2019, accessed: 2025-03-04. [Online]. Available: https://linuxptp.sourceforge.net/

[6] A. Morato, S. Vitturi, F. Tramarin, and A. Cenedese, "Assessment of different OPC UA implementations for industrial IoT-based measurement applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–11, 2021.

[7] Open Source Automation Development Lab *OSADL* eG, "OSADL QA farm on real-time of mainline Linux," accessed: 2025-03-04. [Online]. Available: https://www.osadl.org/OSADL-QA-Farm-Real-time.linux-real-time.0.html

[8] ——, "OSADL QA farm on real-time of mainline Linux: Real-time Ethernet *ethercat* worst-case round-trip time monitoring," accessed: 2025-03-04. [Online]. Available: https://www.osadl.org/Real-time-Ethernet-Ethercat-worst-case.qa-farm-rt-ethernet-recording.0.html

[9] ——, "OSADL QA farm on real-time of mainline Linux: Real-time Ethernet *powerlink* packet interval and jitter analysis," accessed: 2025-05-04. [Online]. Available: https://www.osadl.org/Real-time-Ethernet-Powerlink-jitter-an.qa-farm-rt-powerlink-jitter.0.html

[10] J. Pfrommer, A. Ebner, S. Ravikumar, and B. Karunakaran, "Open source OPC UA PubSub over TSN for realtime industrial communication," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, Sept. 2018, pp. 1087–1090. [Online]. Available: https://ieeexplore.ieee.org/document/8502479/

[11] S. Poretsky, S. Erramilli, J. Perser, and S. Khurana, "Terminology for Benchmarking Network-layer Traffic Control Mechanisms," RFC 4689, Oct. 2006. [Online]. Available: https://www.rfc-editor.org/info/rfc4689

[12] Raspberry Pi Ltd, "Raspberry Pi compute module 5 datasheet: A Raspberry Pi for deeply embedded applications." 2024, accessed: 2025-03-04. [Online]. Available: https://datasheets.raspberrypi.com/cm5/cm5-datasheet.pdf

[13] ——, "Raspberry Pi documentation: config.txt," 2024, accessed: 2025-03-04. [Online]. Available: https://www.raspberrypi.com/documentation/computers/config_txt.html

[14] ——, "Raspberry Pi RP1 peripherals datasheet," 2024, accessed: 2025-03-04. [Online]. Available: https://datasheets.raspberrypi.com/rp1/rp1-peripherals.pdf

[15] ——, "Raspberry Pi 5 product brief," 2025, accessed: 2025-03-04. [Online]. Available: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf

[16] G. P. Reddy, Y. V. P. Kumar, Y. J. Reddy, S. R. Maddireddy, S. Prabhudesai, and C. P. Reddy, "OPC UA implementation for industrial automation - part 2: Integrating PubSub model with TSN," in *2023 1st International Conference on Circuits, Power and Intelligent Systems (CCPIS)*, 2023, pp. 1–6.

[17] M. Ulbricht, S. Senk, H. K. Nazari, H.-H. Liu, M. Reisslein, G. T. Nguyen, and F. H. P. Fitzek, "TSN-FlexTest: Flexible TSN measurement testbed," *IEEE Transactions on Network and Service Management*, vol. 21, no. 2, pp. 1387–1402, 2024.

[18] A. Waterland, "stress(1) - linux man page." [Online]. Available: https://linux.die.net/man/1/stress

[19] K. Williams, "rt-tests - suite of real-time tests." [Online]. Available: https://web.git.kernel.org/pub/scm/utils/rt-tests