A Preliminary Assessment of the real-time capabilities of Real-Time Linux on Raspberry Pi 5

Wannes Dewit
SOFT Languages Lab
Vrije Universiteit Brussel
Brussels, Belgium
wannes.guido.v.dewit@vub.be

Antonio Paolillo
SOFT Languages Lab
Vrije Universiteit Brussel
Brussels, Belgium
antonio.paolillo@vub.be

Joël Goossens
Département d'Informatique
Université libre de Bruxelles
Brussels, Belgium
joel.goossens@ulb.be

***Abstract*—This preliminary study evaluates the practical real-time capabilities of Real-Time Linux with the `PREEMPT_RT` patch on the Raspberry Pi 5, using Cyclictest and various stressors to simulate extreme operational conditions. By comparing the performance and predictability of Linux kernels with and without the `PREEMPT_RT` patch, we establish quantifiable metrics that demonstrate significant improvements in determinism and reduced latency: notably, we observed on Real-Time Linux a $294\times$ shorter maximum latency than on regular Linux. Our findings contribute to a deeper understanding of Real-Time Linux's potential in industrial applications. Our work aims, in the longer term, to establish a measurement-based assessment methodology of the real-time performance and capabilities of real-time operating systems.**

***Index Terms*—Real-Time Linux, PREEMPT_RT, scheduling latency, benchmarking, determinism**

I. Introduction

The explosion of the number of connected devices, automation, and the increasing need for real-time operations in various sectors have prompted the search for operating systems that can meet strict timing requirements. Along proprietary solutions, Linux, the open-source operating system kernel, has seen adaptations like Real-Time Linux to address these real-time demands [1]. Such adaptations are crucial for sectors such as industrial control [2], automotive [3], and even in video games [4], where milliseconds can make a difference.

Recently, the `PREEMPT_RT` patch series, aimed at making the kernel fully preemptive and real-time but currently not yet fully merged into mainline, has gained traction in the Linux community [5], [6].

Research question. How does the implementation of Real-Time Linux with the `PREEMPT_RT` patch affect the scheduling latency and predictability on the Raspberry Pi 5 compared to previous models and other platforms? The Raspberry Pi 5 introduces significant hardware improvements over its predecessors, potentially offering better performance and more deterministic behavior under real-time conditions. Evaluating Real-Time Linux on this updated platform can provide insights into how these hardware advancements contribute to real-time capabilities and whether they justify (or not) the use of the Raspberry Pi 5 in real-time applications. Our goal is to scrutinize and understand the real-time capabilities of Real-Time Linux and evaluate whether it qualifies as a Real-Time Operating System (RTOS). By doing so, we aim to establish a methodology for RTOS assessment enabling us to compare, for example, the behaviour of Linux when applying (or not) the `PREEMPT_RT` patch. This requires the definition of metrics adapted to real-time workloads (as "fast" does not mean "predictable") and associated benchmarks simulating real-time applications.

Overall project. We plan to use existing benchmarking tools and suites, like `Cyclictest` [7], `benchkit` [8] `RTEval` [9], and `rtbench` [10], to evaluate the impact of the `PREEMPT_RT` patch on both performance and real-time metrics. The eventual goals of this project are (1) to produce a comprehensive analysis detailing the real-time capabilities of Real-Time Linux on the chosen platforms, and (2) to provide a benchmarking-based methodology that can be reused for different hardware platforms, other versions of the kernel or even for different RTOS.

This paper. In this experience report, we present preliminary evaluation results of latencies measured with `Cyclictest` when running on Linux, with and without `PREEMPT_RT`, when the system is under heavy stressing conditions, using the `stress-ng` and `iperf3` stressing tools. Using that setting, applying `PREEMPT_RT` results in reducing the maximum observed latency by a $294\times$ factor. While being currently restricted to a single platform and a single benchmark, these results enable us to show key differences between Linux and its real-time equivalent.

II. Background

A. Linux and real-time

The main goal of the `PREEMPT_RT` patch is to make Linux real-time compliant by making the kernel fully preemptible. In the long term, the patch aims to be upstreamed, merging these real-time capabilities as build options available within mainline Linux. As such, a lot of work from the real-time Linux community has already been merged [11] and has even improved Linux performance in non-real-time scenarios [12]. However, notice that it was never the ambition of the real-time Linux community to make the Linux kernel completely hard real-time. This would lead to the loss of a lot of features expected from a modern general-purpose operating system and nowadays considered standard for Linux users.

For example, in the overloaded case – when the task set utilization is greater than the number of CPUs, i.e. there is more work demand than the available processing capacity of the multi-core platform – users expect higher latencies and generally some performance loss. In a real-time environment, such scenario must be avoided to meet the tasks' deadlines. Upstreaming the real-time patch would encourage more users to try it by simply enabling the build option, therefore making the switch as smooth as possible [11], [13].

B. Assessing real-time capabilities

Formally verifying Linux real-time capability is a challenging endeavor [14], [15], and we contend that these initiatives must be complemented with practical testing, i.e., benchmarking, as pursued in prior work [16].

In the first iteration of this project, we propose to use `Cyclictest` to measure the scheduling latency of tasks while the system is under heavy load. `Cyclictest`, developed by Thomas Gleixner, is the de facto benchmarking tool for real-time Linux. It was used in many prior work [17]–[23]. `Cyclictest` measures the so called *scheduling latency* of a real-time system — i.e., the difference between a thread's intended wake-up time and the time at which it actually wakes up. The tool gets periodically invoked in order to calculate the max, min and average scheduling latency [7]. Since the main feature of the `PREEMPT_RT` patch is to make the Linux kernel more deterministic, important metrics to benchmark system performance are its response-time latency and jitter [16], [21], and `Cyclictest` often gets used to provide these metrics. `Cyclictest` *"provides an easy-to-interpret metric that reflects various sources of unpredictability as a single, opaque measure."* [23], making it a very useful tool to quickly compare, for example, different kernel versions. Since it is the most widely accepted benchmarking tool for real-time Linux, it is also very easy to compare new benchmarking results with results of prior work.

`Cyclictest` often gets used together with certain stressing tools [17], [18], [20]. The decision for which parts of the system need to be stressed is usually application-specific, but in general, studies pick stressers which put a load on the CPU (e.g., *stress-ng, phoronix-test-suite's openssl*), I/Os (e.g., *stress-ng, fio, build-linux-kernel*), and networking (e.g., *iperf*) [17], [19], [20].

Since we selected the recently-released Raspberry Pi 5 to conduct our experiments, notice that existing performance-related studies exist [24]. To the best of our knowledge, although many prior work used prior Raspberry Pi models to conduct Real-Time Linux studies [22], there are no other reported studies assessing real-time metrics on this device.

III. Proposed Methodology

A. Selected hardware and OS

We ran the below stressors and benchmark on a Raspberry Pi 5 (Model B Rev 1.0) running Debian 12. The platform has a 2.4 GHz quad-core 64-bit Arm Cortex-A76 CPU and a VideoCore VII GPU.

We ran our experiments in the following scenarios: (1) with a stock kernel, version 6.6.21, and (2) with a `PREEMPT_RT` patched kernel, with the same version and the compatible patch. We analyzed and compared the real-time performance of both kernels in order to determine the benefits of `PREEMPT_RT` in terms of real-time capabilities. Notice that for the stock kernel, we used the vanilla configuration provided by the instructions on Raspberry Pi website [25]. Further configuration tuning could lead to better results – we will explore this as future work. To avoid extra interrupt noise due to graphic processing, we disabled the desktop environment and ran the experiments on the Raspberry Pi through an `ssh` terminal.

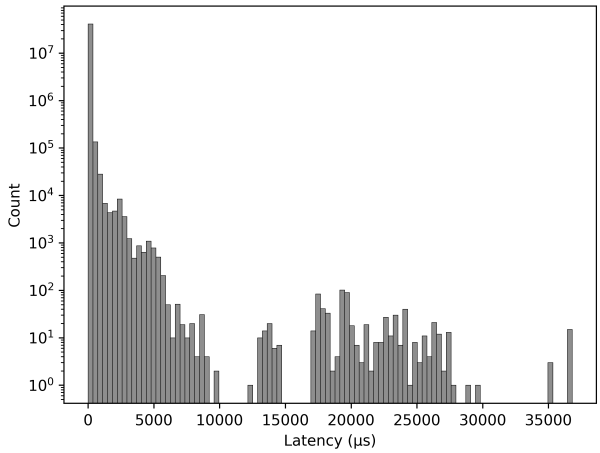
B. Selected software benchmark and stressors

Stressors are used in benchmarking practices in order to generate a computing load to push specific parts of the system to their limit. Since we aim to assess the real-time capabilities of Linux, we use different stressors concurrently with `Cyclictest`. We expect that the combination of these stressors will generate a system wide load to approach the worst-case scenario – this to determine whether the RTOS still performs deterministically under pressured circumstances.

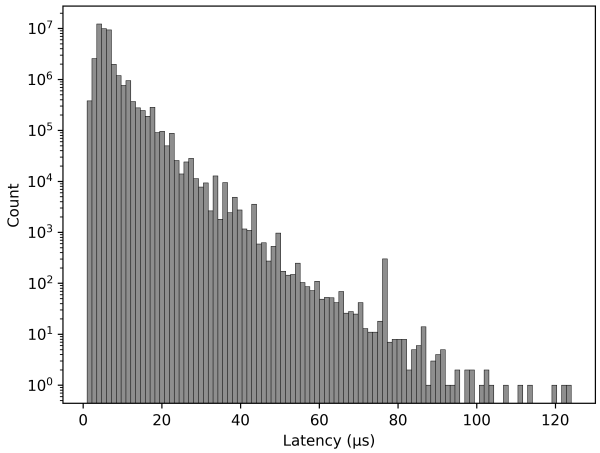
To this end, we used `stress-ng` for generating CPU and I/O load. `stress-ng` is a library of stressors, ranging from tests stressing the CPU, virtual memory, file system or memory/CPU cache [26], allowing us to generate a diverse set of resource-intensive tasks. We also used `iperf3` [27] for generating networking load, which comes with its own set of interrupts and service routines and thus possible sources of latency.

Cyclictest was configured according to best practices in the field [17], [20]–[22], using: `sudo cyclictest -vmn -i100 -p99 -t --duration=1h`. Firstly, memory allocations were disabled (`-m`), `clock_nanosleep` is used instead of POSIX interval timers (`-n`) and the output was set to verbose in order to correctly gather the needed results (`-v`). The real-time tasks that are measured by `Cyclictest` are created every 100 microseconds (`-i100`) with a priority of 99 (`-p99`). The amount of tasks that is created every interval is equal to the amount of processors of the system (`-t`). The tests were run for a duration of 1 hour (`--duration=1h`). As suggested by Adam [22], [28], processor affinity was intentionally not set with `--smp` since this would mitigate task migration. This is an important source of possible latency due to potential acquisition of locks, so it should be included in our tests. Notice that this flag was set in another study by Oliveira [19].

stress-ng ran through docker using the following command: `sudo docker run --rm colinianking/stress-ng --all 1 -t1h 1`. We decided to run all 320+ stressers in parallel (`--all 1`) for a duration of 1 hour (`-t1h`) [17], [20], [29]. Using all stressors in the library in order to generate system wide CPU and I/O load was an idea lent from Delgado, et al [20], with the only difference being that they execute all stressors sequentially, whereas we run all of them in



(a) Latencies for the stock kernel v6.6.21, without the PREEMPT_RT patch.



(b) Latencies for the real-time kernel v6.6.21, with the corresponding PREEMPT_RT patch applied.

Fig. 1: Histograms of Cyclictest measured scheduling latencies for both non-real-time/real-time versions of the kernel. The real-time kernel shows better observed latencies and better predictability.

parallel to trigger a highly demanding workload to maximize the stressing. We expect the system to be able to handle a multitude of different stressors at the same time, and sustain its determinism even under heavy load. We acknowledge that running all stressors in parallel on a Raspberry Pi 5, an embedded system, introduces an unrealistic degree of parallelism. However, our goal was to push the system to its absolute limits to understand the worst-case scenario performance and to stress test the capabilities of the PREEMPT_RT patch under extreme conditions. Future work will involve more realistic stress scenarios that reflect typical workloads encountered in embedded systems.

iperf3 was run from a remote computer using: `iperf3 -c <IP> -w 64K -P 100 -t 3800`. In order to generate networking load for our tests, the remote computer sends out 64KB packets (`-w 64K`) from 100 different virtual clients (`-P 100`) at the same time during more than an hour (`-t 3800`). Notice `iperf3` is a client/server application. We configured the Raspberry Pi 5 to act as a server (by running `iperf3` with the `-s` flag on the Raspberry Pi 5) to receive the networking load from the remote computer acting as the client (by running `iperf3` with the `-c` flag on the remote computer) [17], [20].

C. Reproducibility

We documented our system settings and reproducible methodology on a publicly available repository[1]. To reproduce our results, the provided kernel configuration must be used. To ease the process, we streamlined the process of patching and building Linux with/without PREEMPT_RT in a Dockerfile and provided a README with the commands to run the benchmarks on the target system – here the Raspberry Pi 5. In the future, we aim to create a reusable process for running stressors and benchmarks on real-time Linux (or other RTOSes) across different hardware platforms. This will enable us to build a large database of experiments that assess the real-time performance and capabilities of various Platform/OS combinations.

[1]https://github.com/apaolillo/rtlinux

TABLE I: Observed scheduling latencies with Cyclictest

	Average	Max	Std.
Stock kernel	14.69 µs	36802.00 µs	122.08 µs
RT kernel	5.91 µs	124.00 µs	3.25 µs

IV. Results

Cyclictest measures the scheduling latency of four real-time tasks on the system. These tasks are periodically woken up every 100 microseconds for a period of one hour. During our tests, where Cyclictest and the stressors were running concurrently, we observed the system reached 100% CPU load and about 70% RAM load. Figures 1a and 1b show the measured scheduling latencies during our experiment. The benefits of PREEMPT_RT in terms of predictability are clear. The results for the stock kernel are very scattered, with a maximum observed latency of 36802 microseconds. In contrast, the negative slope on the graph for the real-time kernel indicates greater stability, with a maximum observed scheduling latency of 125 microseconds – achieving a $\times 294$ improvement of the maximum observed latency. These observations, together with the general lack of outliers on Figure 1b indicates that the PREEMPT_RT patch succeeds in its goal of making the kernel more deterministic.

The same results are aggregated in Table I for convenience. Notice that the average scheduling latency also improved in the patched system, from 14.69 to 5.91 microseconds, with a less impressive improvement of $\times 2.49$. This points to the fact that

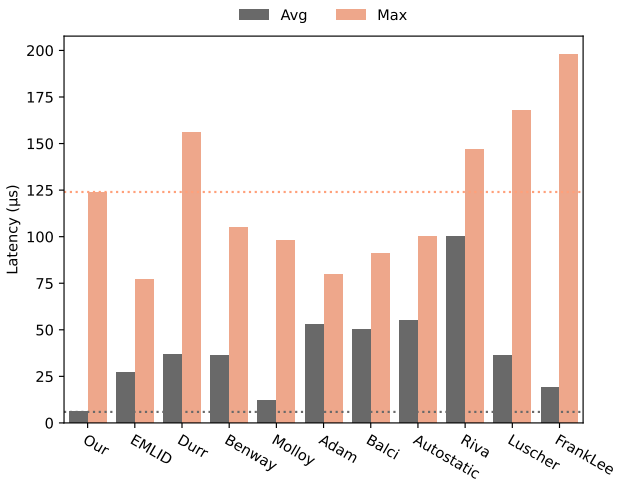


Fig. 2: Average and maximum observed scheduling latencies with `Cyclictest` in our study and various prior studies. Our own measured average and maximum scheduling latencies are drawn as horizontal lines across the graph to ease the comparison.

the goal of the `PREEMPT_RT` patch is not necessarily to make the kernel faster but mainly to make it more deterministic for real-time tasks. The reduction of the standard deviation further supports this claim and indicates how much the determinism of the kernel was improved by the patch.

In Figure 2, we compare our results obtained with the patched kernel to a series of benchmarks gathered by Adam, et al. [22], supplemented with some more recent studies [30], [31]. These benchmarks were gathered from other studies which benchmarked other Raspberry Pi models with `Cyclictest` [22], [30]–[38]. Notice that the objective of this comparison is primarily to corroborate the validity of our findings, noting that precise quantitative comparisons are not expected. Instead, an affirmation of the order of magnitude is sought, as the referenced benchmarks employed no stressors or utilized different ones, yielding varying testing conditions for the experiments. It is clear that the average scheduling latency that we obtained is substantially better than the other results. This was to be expected however, since these benchmarks were run on previous iterations of the Raspberry Pi, namely the Raspberry Pi 1, 2, 3 and 4. The Raspberry Pi 5 that we used for our benchmarks has a CPU with a clock frequency of 2.4 GHz, which is double the clock frequency of the CPU on for example the Raspberry Pi 3. The leap in average scheduling latency could thus be ascribed to the improvement in hardware, but also to improvements in the Linux kernel or the `PREEMPT_RT` patch (since other studies used different versions of those). Results of the maximum observed scheduling latency is quite comparable to the other results, which is a bit disappointing. We ascribe these subpar results to the fact that we decided to run the benchmarks on an untuned kernel for this first iteration of our project.

V. Future work

The next step of our project will be to further experiment with configuring and tuning the kernel in order to get better real-time performance results. We are confident that the Raspberry Pi 5 can still be pushed further in order to obtain better results. For example, kernel configuration options that could be explored are: disabling RT-throttling, disabling CPU frequency scaling, and raising software interrupt priority [29]. We are also interested in experimenting with the best practices in configuring the Linux kernel for real-time as described in the Red Hat manual [39].

As the goal of the research project is to establish a methodology for assessing the real-time capabilities of RTOSes, a natural next step for comparison would also be to assess the capabilities of other, non-Linux RTOSes, such as Zephyr, FreeRTOS, LITMUS^{RT}, or proprietary products. We also plan to explore other benchmarking tools such as `rt-bench` [10] or `RTEval` [9]. RTEval also uses `Cyclictest` as a measuring tool but has another approach to stressing the system. In future work, besides scheduling latency, other metrics will be considered to complete our assessment methodology, such as end-to-end response latency and RTOS jitter.

We will also consider running our experiments on other hardware platforms such as other embedded systems (e.g., Orange Pi, Raspberry Pi 4, Rock Pi 4) or many-core processors (e.g., AMD EPYC, Huawei Kunpeng, Ampere Altra).

Finally, we will streamline the process of getting real-time KPIs through the benchkit [8], aiming for a fully open-source and reproducible benchmarking pipeline.

References

[1] Canonical, “Real-time Ubuntu is now generally available,” February 2023, [Online] Accessed: 2024-05-06. [Online]. Available: https://canonical.com/blog/real-time-ubuntu-is-now-generally-available

[2] C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. M. Vilches, “Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications,” *CoRR*, vol. abs/1809.02595, 2018. [Online]. Available: http://arxiv.org/abs/1809.02595

[3] I. Wind River Systems, “Wind River Acquires Hard Real-Time Linux Technology from FSMLabs,” February 2007, [Online] Accessed: 2024-05-06. [Online]. Available: https://www.windriver.com/news/press/news-4261

[4] M. Larabel, “SteamOS Compositor Details, Kernel Patches, Screenshots,” December 2013, [Online] Accessed: 2024-05-06. [Online]. Available: https://www.phoronix.com/news/MTU0MzY

[5] ——, “PREEMPT_RT Might Be Ready To Finally Land In Linux 5.20,” July 2022, [Online] Accessed: 2024-05-06. [Online]. Available: https://www.phoronix.com/news/520-Maybe-Real-Time-PREEMPT_RT

[6] ——, “Real-time ”rt” patches updated against linux 6.6 git,” September 2023, [Online] Accessed: 2024-05-06. [Online]. Available: https://www.phoronix.com/news/Linux-RT-Patches-Linux-6.6

[7] T. L. Foundation, “Linux Foundation Realtime Wiki - HowTo - Cyclictest,” [Accessed 2024-05-06]. [Online]. Available: https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start

[8] open s4c, “benchkit: Benchmarking Toolkit for Performance Analysis,” 2024, [Accessed 2024-05-07]. [Online]. Available: https://github.com/open-s4c/benchkit

[9] “Rteval,” [Online] Accessed: 2024-05-06. [Online]. Available: https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/rteval

[10] M. Nicolella, S. Roozkhosh, D. Hoornaert, A. Bastoni, and R. Mancuso, "Rt-bench: an extensible benchmark framework for the analysis and management of real-time applications," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, ser. RTNS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 184–195. [Online]. Available: https://doi.org/10.1145/3534879.3534888

[11] J. Perlow. (2021) In the trenches with thomas gleixner: Real-time linux kernel patch set. [Accessed 2024-05-06]. [Online]. Available: https://www.linux.com/news/in-the-trenches-with-thomas-gleixner-real-time-linux-kernel-patch-set/

[12] T. Gleixner. (2022) A guided tour through the preempt rt castle. [Accessed 2024-05-09]. [Online]. Available: https://www.youtube.com/watch?v=o58ff38eD64&t=440s

[13] I. Stoica and H. Abdel-Waheb, "Earliest eligible virtual deadline first: A flexible and accurate mechanism for proportional share resource allocation," Old Dominion University, USA, Tech. Rep., 1996. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=805acf7726282721504c8f00575d91ebfd750564

[14] S. Rostedt, "Kernel Recipes 2016 - Who needs a Real-Time Operating System (Not You!)," https://youtu.be/4UY7hQjEW34?si=EL8w75sHzS9WjqwK, 2016, [Accessed 2024-05-06].

[15] Bielmeier, Benno and Mauerer, Wolfgang, "Formal Verification of Embedded Linux Systems Using Trace-Based Models," https://www.youtube.com/watch?v=w42ab8-CH1o, 2022, [Accessed 2024-05-09].

[16] F. Reghenzani, G. Massari, and W. Fornaciari, "The Real-Time Linux Kernel: A Survey on PREEMPT_RT," *ACM Comput. Surv.*, vol. 52, no. 1, 02 2019. [Online]. Available: https://doi.org/10.1145/3297714

[17] Y. H. Jo and B. W. Choi, "Performance Evaluation of Real-time Linux for an Industrial Real-time Platform," *International Journal of Advanced Smart Convergence*, vol. 11, no. 1, pp. 28–35, 2022. [Online]. Available: https://doi.org/10.7236/IJASC.2022.11.1.28

[18] K. Koolwal, "Investigating latency effects of the Linux real-time Preemption Patches (PREEMPT RT) on AMD ' s GEODE LX Platform Kushal Koolwal," in *Proceedings of the 11th OSADL Real-Time Linux Workshop*, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:11519524

[19] D. B. de Oliveira, D. Casini, R. S. de Oliveira, and T. Cucinotta, "Demystifying the Real-Time Linux Scheduling Latency," in *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), M. Völp, Ed., vol. 165. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, pp. 9:1–9:23. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECRTS.2020.9

[20] R. Delgado and B. W. Choi, "New Insights Into the Real-Time Performance of a Multicore Processor," *IEEE Access*, vol. 8, pp. 186 199–186 211, 2020.

[21] N. Litayem and S. Ben Saoud, "Impact of the Linux real-time enhancements on the system performances for multi-core Intel architectures," *International Journal of Computer Applications*, vol. 17, no. 3, pp. 17–23, Mar. 2011.

[22] G. K. Adam, N. Petrellis, and L. T. Doulos, "Performance assessment of linux kernels with preempt_rt on arm-based embedded devices," *Electronics*, vol. 10, no. 11, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/11/1331

[23] F. Cerqueira and B. B. Brandenburg, "A comparison of scheduling latency in linux, preempt-rt, and litmus rt," in *Proceedings of the 9th Annual Workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT 2013)*, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:14096981

[24] M. Larabel, "Raspberry Pi 5 Benchmarks: Significantly Better Performance, Improved I/O," September 2023, Accessed: 2024-05-08. [Online]. Available: https://www.phoronix.com/review/raspberry-pi-5-benchmarks/4

[25] Raspberry Pi Foundation. Building the Linux Kernel. [Accessed 2024-05-09]. [Online]. Available: https://www.raspberrypi.com/documentation/computers/linux_kernel.html#building

[26] C. I. King, "stress-ng (stress next generation)," [Accessed 2024-05-06]. [Online]. Available: https://github.com/ColinIanKing/stress-ng

[27] ESnet, "iPerf - The ultimate speed test tool for TCP, UDP and SCTP," [Accessed 2024-05-06]. [Online]. Available: https://iperf.fr/

[28] G. K. Adam, "Real-time performance and response latency measurements of linux kernels on single-board computers," *Computers*, vol. 10, no. 5, 2021. [Online]. Available: https://www.mdpi.com/2073-431X/10/5/64

[29] P. Karachatzis, J. Ruh, and S. S. Craciunas, "An evaluation of time-triggered scheduling in the linux kernel," in *Proceedings of the 31st International Conference on Real-Time Networks and Systems*, ser. RTNS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 119–131. [Online]. Available: https://doi.org/10.1145/3575757.3593660

[30] M. Lüscher, "Real-Time Linux on the Raspberry Pi," 2018, [Accessed 2024-05-09]. [Online]. Available: https://www.get-edi.io/Real-Time-Linux-on-the-Raspberry-Pi/

[31] F. Lee, "How to optimize real-time performance," 2023, [Accessed 2024-05-09]. [Online]. Available: https://forums.raspberrypi.com/viewtopic.php?t=363635

[32] D. Molloy, *Exploring Raspberry Pi Interfacing to the Real World with Embedded Linux*. Indianapolis, IN, USA: John Wiley & Sons, Inc., 2016.

[33] EMLID, "EMLID Raspberry Pi Real-Time Kernel," [Accessed 03-05-2024]. [Online]. Available: https://emlid.com/raspberry-pi-real-time-kernel/

[34] Durr, Frank, "Raspberry Pi Going Realtime with RT Preempt," [Accessed 03-05-2024]. [Online]. Available: http://www.frank-durr.de/?p=203

[35] Benway, Joel, "Real-Time on Raspberry Pi," [Accessed 03-05-2024]. [Online]. Available: https://www.distek.com/blog/part-2-real-time-on-raspberry-pi/

[36] Metehan Balci, "Latency of Raspberry Pi3 on Standard and Real-Time Linux 4.9 Kernel," [Accessed 03-05-2024]. [Online]. Available: https://metebalci.com/blog/latency-of-raspberry-pi-3-on-standard-and-real-time-linux-4.9-kernel/

[37] AUTOSTATIC, "RPi3 and the Real Time Kernel," [Accessed 03-05-2024]. [Online]. Available: https://autostatic.com/2017/06/27/rpi-3-and-the-real-time-kernel/

[38] Lema Riva, "Raspberry Pi: Preempt-RT vs. Standard Kernel 4.14.y," [Accessed 03-05-2024]. [Online]. Available: https://lemariva.com/blog/2018/02/raspberry-pi-rt-preempt-vs-standard-kernel-4-14-y

[39] Red Hat, "Real-Time Kernel Tuning in RHEL 8," [Accessed 03-05-2024]. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/8/html/optimizing_rhel_8_for_real_time_for_low_latency_operation/real-time-kernel-tuning-in-rhel-8_optimizing-rhel8-for-real-time-for-low-latency-operation#doc-wrapper