Performance Evaluation of an EtherCAT Master using Linux and the RT Patch

Marco Cereia, Ivan Cibrario Bertolotti, and Stefano Scanzio
IEIIT–CNR
C.so Duca degli Abruzzi, 24
I-10129 Torino (Italy)
Email: {marco.cereia, ivan.cibrario, stefano.scanzio}@polito.it

***Abstract*—This paper has the twofold goal of investigating the real-time performance of an EtherCAT master entirely built from open-source components (using Linux and the RT Patch at the operating system level) and assess its ability to support concurrent best-effort tasks without compromising the real-time ones, depending on kernel configuration. This is especially important for the successful adoption of the proposed approach in real-world applications. A hardware-based data acquisition system enables measurements to be taken for long periods of time, and with high resolution and precision. At the same time, this method guarantees that the measurement process does not influence the behavior of the system under test in any way.**

I. Introduction

Recently, industrial control systems derived from personal computers' hardware and software are starting to be used in factory automation systems, even for demanding real-time applications. This trend is further encouraged by the ever increasing diffusion of high-performance, real-time Ethernet networks, which allow a complete industrial control system to be built in a modular way and without installing any custom interface card on the PC side. For example, EtherCAT [1] only needs a standard Ethernet interface on its master nodes.

At the same time, the real-time capabilities of most general-purpose operating system, even open-source ones, are becoming better and better, opening the option of adopting them on the software side. Among other techniques that aim at enhancing Linux from this point of view [2]–[7], the RT Patch [8] is especially promising, because it is easy to deploy and maintain. Moreover, it will likely be fully integrated with the mainline Linux kernel in the future, and will therefore become readily available for use in all Linux distributions.

The primary goal of this paper is to ascertain the timing accuracy to be expected from an EtherCAT master entirely based on open-source components. The importance of this measurement stems from the fact that in many cases, that is, unless time stamping and distributed clocks are systematically used, the acquisition and actuation accuracy of an EtherCAT system is strongly related to the timing accuracy of the master.

The degree of isolation among the real-time application and other concurrent kernel and user-level activities, depending on kernel configuration, is also investigated. This feature is especially useful if one wants to support both real-time and best-effort tasks on the same processor. After giving a detailed account of the testbed adopted for the experiments in Sect. II, the paper proceeds presenting the experimental results in Sect. III. Finally, Sect. IV draws some conclusions and discusses related work.

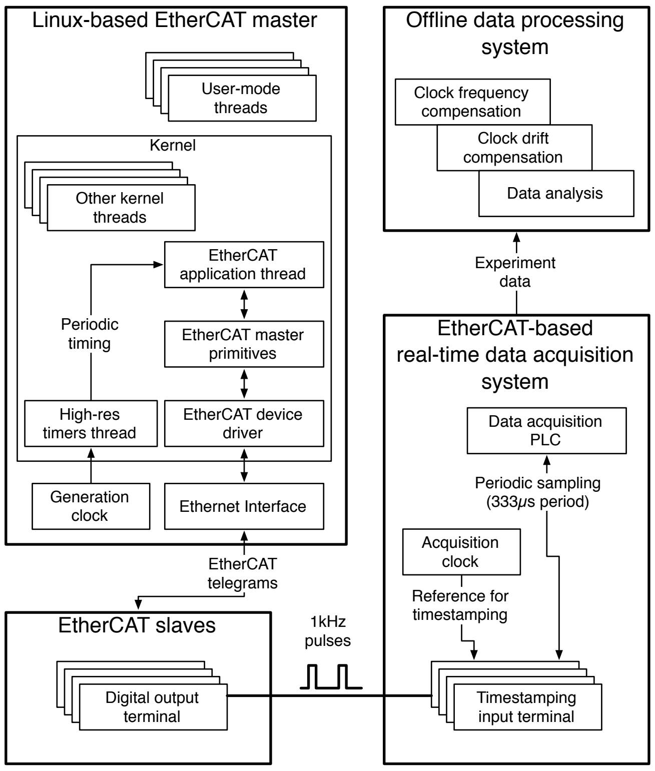

Fig. 1. Simplified block diagram of the testbed used for the experiments.

II. Experimental Testbed

The experimental testbed comprises three main components:

1) The EtherCAT system under test (shown on the left of Fig. 1) is built around a PC running Linux and acting as the master, as well as a number of slaves. The PC has an Intel Core2Duo E6750 dual-core CPU running at 2.66 GHz and 4 GByte of dual-channel DDR2 RAM. In order to make the performance level of this PC more similar to a contemporary industrial PC and to limit the scope of the analysis to uniprocessor systems, one of the cores has been disabled from BIOS.

The test application uses the open-source IgH EtherLab EtherCAT master component [9] as foundation. By means of a high-speed digital output terminal with a switching time of less than $1\,\mu$s (Beckhoff EL2202 [10]) it generates a test signal consisting of a train of pulses with a nominal frequency of $1\,\mathrm{kHz}$, controlled by a generation clock that uses the PC clock as time base. This terminal has been programmed to actuate its outputs asynchronously, as soon as it receives an actuation command from the EtherCAT network, hence the output timings directly reflect the command timings.

2) The measurement system (on the bottom right of Fig. 1) is also EtherCAT-based. A data acquisition PLC [11] uses a time stamping input terminal (Beckhoff EL1252 [12]), with the distributed clock function disabled and thus driven by its own free-running acquisition clock, to capture the rising edges of the waveform generated by the system under test. The PLC sampling period has been set to a third of the nominal edge frequency to accommodate reasonable jitters on both the signal generation and acquisition periods without losing samples.
3) At the end of the experiment, all acquired data are eventually sent to another PC (on the top right of Fig. 1) for offline processing and analysis. This processing will be discussed in greater detail in Sect. III.

In the testbed, the data acquisition and processing systems have been purposefully kept isolated from the system under test to avoid any undue perturbation. For the same reason, all data processing is done offline, since this minimizes the load of the data acquisition system during the experiment. Hence, the resolution and precision of the acquisition system are limited only by the input terminal specifications. According to the manufacturer, the EL1252 provides a resolution of $1\,\mathrm{ns}$ and a precision of $10\,\mathrm{ns}$ with respect to its local acquisition clock.

On the other hand, this choice has the drawback of limiting the maximum number of samples taken in a single experiment, because they must all fit in the limited PLC memory. In this case, the PLC storage can hold the timestamps of up to 10^6 samples without interrupting the experiment to offload data.

From the software point of view it should be noted that—even if the test application running on the EtherCAT master is very simple—all the elements of the chain that goes from the hardware-based generation clock through the operating system kernel and to the Ethernet interface have been kept the same as they would be in other, more complex cases. Hence, the timing accuracy of the test application is representative of the real-world behavior of similar systems.

For the same reason, this paper focuses on the overall jitter of the system actuators and not on the contribution to this jitter made by the individual software components of the actuation chain. The measurement errors ascribed to measurement system inaccuracies, namely clock frequency tolerance and drift, have instead been compensated.

The major software components involved are:

- The *high resolution timers module* is triggered by a hardware time reference and invokes the timer handling functions, registered by other kernel components, when appropriate. With the application of the RT Patch, these handlers are no longer called directly from the soft IRQ context. On the contrary, they are all invoked by a dedicated kernel thread (`sirq_hrtimer`), so that they have no longer a priority higher than any task in the system. In addition, the RT Patch also makes provision for separate threads for each interrupt source. This approach allows the system to give a different scheduling priority to each of them, because interrupt handling functions inherit the priority of the corresponding thread. The default priority assignment is fair, that is, it gives to all interrupt handling threads the same priority `-51`.
- The *EtherCAT application thread* has been derived from the example given in [9]. It is awakened periodically by a high resolution timer handler, with a frequency of $1\,\mathrm{kHz}$, and calls the EtherCAT master primitives to ask for the generation of a pulse on the digital output terminal.
  With respect to the example, the application code is no longer executed directly by the timer handler. This choice keeps the handler short regardless of the application complexity, to bound its impact on the latency of the other timer handlers. The priority of the application thread has been set to `-99`, the maximum real-time priority foreseen by the FIFO scheduling policy.
- The *EtherCAT master code* builds the appropriate command telegrams and sends them to the digital output terminal through the Ethernet interface. The interface is driven by a dedicated device driver included in [9]. In addition, the EtherCAT master code also transmits management telegrams for its own use. However, this has no effect on the accuracy of the application because, in the default operating mode chosen for the tests, the management frames are never sent spontaneously and are always placed *after* the application telegrams within the EtherCAT frames.

III. Results

A. Dedicated Real-Time System

Among all kinds of measurements that can be conducted using the testbed described in Sect. II, this paper analyzes the jitter of the test waveform's rising edges. This is a basic measurement, but is anyway very useful to evaluate the accuracy of a similarly built control system. The experiments have been conducted for 10^6 samples, in order to capture even the anomalies that occur with low probability. For a test waveform period of $1\,\mathrm{ms}$, this corresponds to an experiment length of $1000\,\mathrm{s}$, that is, about 16 minutes.

Due to the unavoidable oscillator tolerances, the generation and acquisition systems (each driven by its own clock) will not have exactly the same time base frequency. As a consequence, a preliminary post-processing step on the experimental data consists of compensating for this difference. In this case, a linear regression of the measured edge positions with respect to their nominal instant of occurrence has been used to

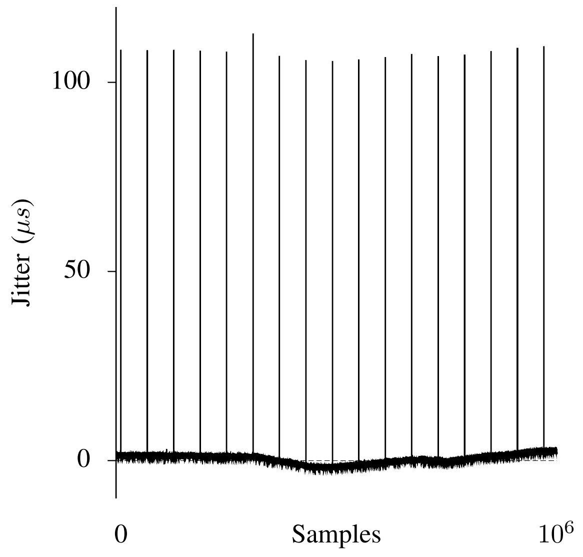

Fig. 2. Example of the experimental data after clock frequency compensation, for a total of 10^6 samples (about 16 minutes of experiment time).

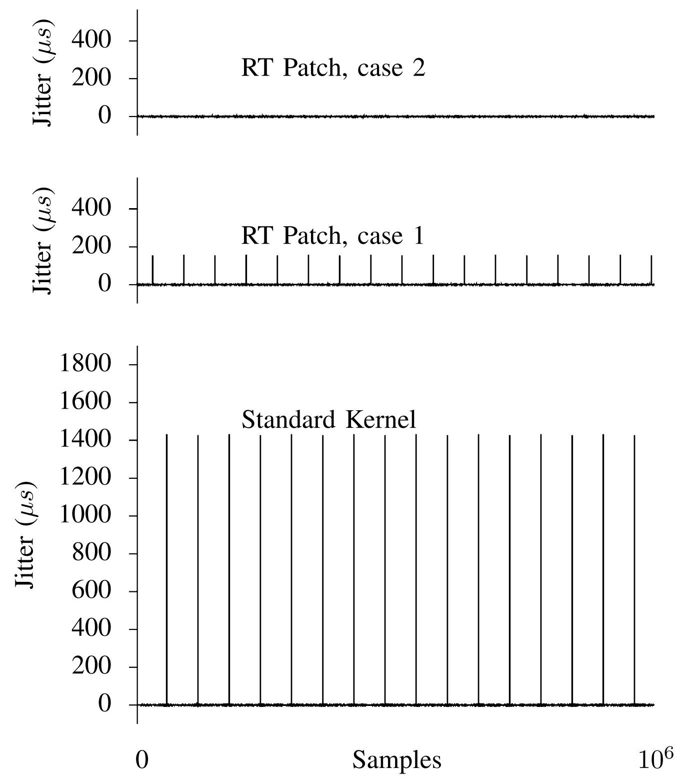

Fig. 3. Rising edge jitter after clock frequency and drift compensation, for different kernel configurations and a total of 10^6 samples (about 16 minutes of experiment time). No best-effort load has been applied to the system.

estimate the ratio between the generation and acquisition time bases; this is given by the slope of the regression line. Then, the compensation has been achieved by time-scaling the experimental data by this ratio. The typical ratio observed during the experiments was close to 1.0002, which corresponds to a frequency difference of 200 parts per million.

Fig. 2 shows an example of the experimental data obtained after clock frequency compensation. Apart from the scheduling anomaly that makes itself evident by the periodic jitter peaks and will be investigated in the following, it can also be seen that the jitter still wanders around zero, in a way consistent with clock drift. Since the drift mostly depends on the quality of the oscillator and not on software, it has been removed by means of a further post-processing step.

In particular, a moving average on 16000 adjacent samples, corresponding to a time window of 16 s, has been used to estimate the drift and to compensate for it. The length of the sliding window has been chosen empirically, as a compromise between being able to track the drift accurately, and avoid masking the local, short-term jitter caused by software.

A first set of experiments has been carried out without imposing any additional load to the system, besides the test application. The results, shown in Fig. 3, represent the behavior of the system when it runs a real-time application exclusively. Three different kernel configurations have been tested:

- A standard kernel, version `2.6.24.7` (labelled “Standard Kernel” in Fig. 3).
- The same kernel with the RT Patch `2.6.24.7-rt27` applied and all interrupt handling threads left to their default priority, `-51` (“RT Patch, case 1”).
- The RT Patched kernel with the priority of the timer interrupt handler raised one step above the other interrupt handlers, that is, from `-51` to `-52` (“RT Patch, case 2”).

It should be noted that, in the last case, the priority change does not involve any modification to the Linux or RT Patch source code, but can be accomplished by means of the command `chrt` at system startup.

The most important observation that can be done on the experimental data is that—even if the jitter is very close to zero on average—there is indeed a major anomaly affecting the behavior of the standard kernel and, to a lesser degree, the kernel with the RT Patch and default interrupt handlers priorities (“RT Patch, case 1”).

Even if this anomaly occurs with a low probability, it has a detrimental effect from the real-time point of view, because its amplitude (about $1400\,\mu$s) exceeds the cycle period of the application under test when using the standard kernel. Even if the amplitude of the jitter is lower when using the RT Patch (about $200\,\mu$s), it still constitutes about one fifth of the cycle period, a value that is likely to be unacceptable for many hard real-time applications.

After some investigations, this anomaly has been attributed to the concurrent and mutually exclusive execution of another periodic, high-priority task within the kernel. In particular, it is due to the flush of the routing table cache, which is performed within a legacy (*jiffies*-based) timer handler. This is a standard maintenance activity carried out periodically inside the Linux kernel to remove stale entries from the routing table cache, a hash table holding the most recently used network routes for quick access.

In order to make the anomaly more evident during the experiments, the flush period has been set to 60 s, whereas the default period is 10 minutes. It should however be noted that keeping the default period only makes the anomaly less evident, not less harmful. On the contrary, a lower probability of occurrence only makes it more difficult to detect in real-world applications.

As discussed in Sect. II for the high resolution timer handlers, also this kind of handler is called from a soft IRQ context in the standard kernel, and by a dedicated thread (`sirq_timer`) when the RT Patch has been applied. As a consequence, with the standard kernel, both high resolution and legacy timer handlers are invoked from a soft IRQ context and therefore interfere with each other, because on a uniprocessor system they are mutually exclusive.

Considering the RT Patch the situation is similar because, even if the priority of the EtherCAT application thread is, by itself, higher than the priority of legacy timer handlers, it is nonetheless triggered by the high resolution timer handling thread `sirq_hrtimer`. With the default RT Patch configuration, both `sirq_timer` and `sirq_hrtimer` have the same priority. As a consequence, on a uniprocessor system, they are serialized by the scheduler for execution and interfere. In turn, the additional latency incurred by `sirq_hrtimer` due to this interference is propagated forward to the EtherCAT application thread.

The exact amount of interference is hard to calculate, because it depends not only on the length of the interfering task (that is, on the number of entries in the routing table cache), but also on the ratio of the periods and on the relative phase of the interfering tasks. Hence, it can vary from one experiment to another.

On the other hand, the RT Patch is able to alleviate the problem with respect to the standard kernel, because flushing the routing cache involves a number of computationally expensive operations known as *read-copy-update* (RCU) in the Linux jargon. In the standard kernel, these operations cannot be preempted and are hence executed as a whole within the legacy timer handler itself. Instead, the RT Patch makes these operations preemptable and executes them from their own thread (`sirq_rcu`). Then, each invocation of an RCU from the legacy timer handler becomes a rescheduling (and, possibly, a preemption) point.

In this particular case, due to the relative position of the interfering threads in the scheduling queue, the first invocation of an RCU preempts the legacy timer thread in favor of the high resolution timer thread. After the EtherCAT application thread has been awakened, no further interference is possible, due to its higher priority.

This explanation is supported by the experimental data because, as it may be expected, the anomaly completely disappears when the priority of the high resolution timer thread is set to be higher than the legacy timer thread. This situation corresponds to the experimental data shown on the top of Fig. 3 and labelled "RT Patch, case 2". In this case, the interference can no longer occur because, referring again to Fig. 1, the whole software chain that goes from the generation clock to the Ethernet interface is made up of tasks with a real-time execution priority strictly higher than the interfering one.

B. Best-Effort Application Support

The second set of experiments aims at checking whether, and to which extent, Linux is able to isolate the EtherCAT master from other, concurrent tasks executed in user mode under the control of the standard, best-effort Linux scheduler. This property is especially important if the same system must be able to accommodate both real-time control functions and other application software that, for example, connects the system to the upper layers of the factory automation hierarchy for remote management or data logging.

To this purpose, three different kinds of best-effort load, believed to be representative of different classes of real-world applications, have been run concurrently with the application under test for the whole length of the experiment:

- A CPU-bound Python application. It uses a limited amount of RAM, hence its execution does not involve any paging activity.
- A C application that manipulates a data structure larger than the available RAM and therefore requires heavy paging.
- An I/O-bound application, using the `dd` utility to write large amounts of data to the hard disk through the file system.

Among these loads, the I/O-bound application had the worst effect on the real-time application, whereas the effect of the CPU-bound application was almost negligible. That is, Linux provides an effective isolation between real-time and best-effort tasks for what concerns CPU usage, but is unable to fully remove the interference due, for example, to mutual exclusion regions within the kernel, hard interrupt requests from the I/O devices, or memory bus contention by other bus masters.

For brevity, Fig. 4 shows only the results obtained in the worst case. The kernel configurations being tested are the same as in the previous set of experiments and, like before, the major anomaly affecting the standard kernel and the RT Patch with default thread priorities can still be noticed even if, for the standard kernel, it has a lower peak value (of about $1100\,\mu$s instead of $1400\,\mu$s).

All other experimental conditions being the same, this difference has been attributed to a different relative phase of routing table flush with respect to the test application. This may vary from one experiment to another, because the two tasks are completely asynchronous and their phases are essentially chosen at random during system startup.

Another useful observation, which can be made from these experiments, is that changing the default kernel thread priorities makes the RT Patch significantly more robust with respect to the best-effort I/O load, too. The effect of this action is therefore the same when the disturbance comes from a user mode application or from within the kernel itself.

In order to better appreciate the degree of isolation provided by the RT Patch when configured with appropriate kernel

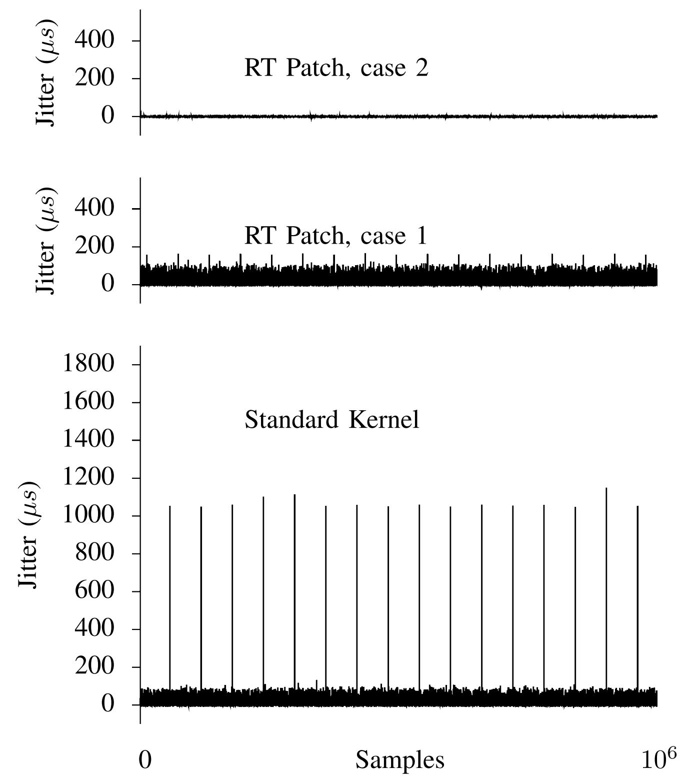

Fig. 4. Rising edge jitter after clock frequency and drift compensation, for different kernel configurations and a total of 10^6 samples (about 16 minutes of experiment time). A heavy best-effort I/O load has been applied to the system.

thread priorities (labelled "RT Patch, case 2" in the previous figures), Fig. 5 contains an expanded view of the RT Patch behavior in this case, with and without a best-effort I/O load.

The worst-case jitter measured during the experiment has an absolute value below $11\,\mu$s. Even with a relatively small cycle period of $1\,\mathrm{ms}$, used in these experiments, the jitter is likely to be negligible for many applications, because it represents only about 1% of the period itself.

Last, Table I gives some summary statistics about all experimental data discussed so far. Besides the usual mean value and standard deviation of the jitter (denoted by μ and σ, respectively) the minimum and maximum value of the jitter are also given, because they give a more accurate idea of the worst-case behavior of the system.

Since the experimental data have been subject to clock frequency and drift compensation before statistical analysis, any deviation of the mean jitter value from zero may reflect an inaccuracy of the post-processing, drift compensation in particular. In turn, this inaccuracy gives rise to an analogous error in jitter calculation, because the jitter is measured with respect to the expected occurrence of the event.

However, as it can be seen from Table I, the worst possible error is very limited, on the order of $2\,\mathrm{ns}$, hence it has no effect on the significance of the results.

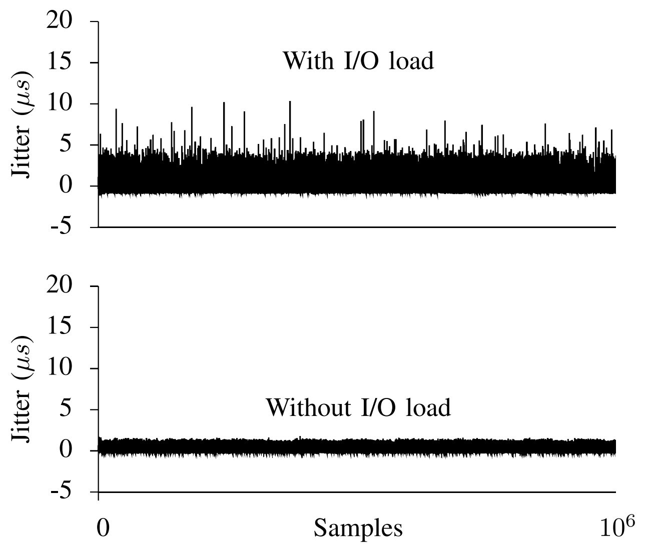

Fig. 5. Rising edge jitter after clock frequency and drift compensation, when using the RT Patch with appropriate kernel thread priorities, with and without I/O load. 10^6 samples (about 16 minutes of experiment time).

TABLE I
SUMMARY STATISTICS OF THE JITTER

Kernel	**Best-effort**	μ	σ	Min	Max
configuration	**I/O load**		(μs)		
Standard	No	0.002	5.724	-0.411	1434.0
Rt Patch, case 1	No	0.000	0.659	-0.490	160.0
Rt Patch, case 2	No	0.001	0.120	-0.424	1.8
Standard	Yes	-0.001	5.752	-4.378	1151.0
Rt Patch, case 1	Yes	-0.001	4.599	-1.804	168.0
Rt Patch, case 2	Yes	0.000	0.632	-1.171	10.4

IV. CONCLUSIONS AND RELATED WORK

The topic of supporting real-time applications using the Linux operating system and evaluating their performance has already been addressed in the past.

Among other possible techniques, both RTAI and Xenomai have been shown to be worth of consideration—at least from the performance point of view—to build demanding hard real-time systems, even with respect to very strong competitors such as, for example, VxWorks [13]. The same paper also rules out the adoption of the standard Linux kernel for the same class of applications, but it does not consider the RT Patch at all. Similarly, [14] thoroughly investigated the interrupt response performance of a Linux kernel patched with RTAI, without considering the RT Patch.

Even if the architectural differences between the two test systems (from both the hardware and software perspectives) preclude a faithful comparison, the performance level of the RT Patch analyzed in this paper comes close to the other real-time systems considered in [13]. For example, the worst-case jitter of about $11\,\mu$s attained by an appropriately configured RT Patch, even under heavy I/O load, is of the same order of magnitude as the jitters measured with real-time network communication in [13]. Other data about the general performance of Linux in a real-time, embedded industrial system are shown in [15], although only one of the test cases considered there involves network communication.

Like in this paper, [16] used a hardware-based acquisition system to evaluate the performance of different versions and configurations of the Linux kernel. However, its focus was on evaluating the effects of the Realtime Preemption Patches (RT-Preempt) and, in particular, their impact on the *general purpose* performance of the system, rather than on real-time.

Moreover, in [16] data acquisition is performed by a general purpose digital storage oscilloscope programmed to have a persistence as long as the experiment, and the analysis has been conducted on the base of the resulting oscilloscope images. Whereas these images represent essentially a "time integral" of the system behavior, the acquisition system proposed in this paper provides a complete temporal trace of all relevant events, thus allowing a deeper data analysis.

The latter approach allows, for example, to distinguish between periodic and sporadic anomalies and to assess their relative probability. This has been very useful to analyze them, and locate their origin. In fact, to the best knowledge of the authors, the anomaly due to routing cache flush presented in this paper has not been discussed in literature before, even if some general information about it has recently appeared in a few specialized Internet forums.

Other works [8], [17], [18] perform a performance analysis of several real-time extensions of the Linux kernel, including the RT Patch, but rely entirely on a software-based data acquisition system, implemented by instrumenting the real-time application. This approach has the advantage of being simpler to implement, but it cannot guarantee that the additional code does not perturb the application under test, and it may possibly mask off some anomalies. In addition, the authors of [17] performed their experiment using kernel version `2.5`. This may not accurately reflect the behavior of a contemporary system, because the internal structure of the Linux kernel has evolved considerably in the meantime.

For what concerns the realism of the test applications, it should be remarked that [17], [18] perform their experiment with test cases that are completely synthetic. Even if they have been carefully designed, in order to stress the individual kernel components and mechanisms which are most important from the real-time execution point of view, nonetheless they may still not be fully representative of a real-world industrial control system. For example, [17] does not consider the effect of network interface access on system performance.

In conclusion, this paper showed that an appropriately configured RT Patch can support a wide range of hard real-time applications and effectively isolate them from other best-effort loads present in the system. Most importantly, this result can be achieved using only completely general purpose, PC-based hardware and open-source software on the control system side.

At the same time, the results discussed in this paper highlight the importance of an appropriate *configuration* of the RT Patch in order to have a good (or even an acceptable) level of performance and avoid low-probability anomalies that may disrupt the behavior of the system. Due to the inherent complexity of the Linux kernel, the reasons behind these anomalies may be unknown to the real-time programmer and very difficult to detect during bench testing.

As a future work, we plan to re-implement the test application using RTAI, and thus compare the performance of the RT Patch with a more traditional approach to real-time support in Linux. Moreover, further experiments using a multi core processor have been planned because, as suggested in [18], it may have unforeseen effects on the real-time properties of Linux. Last, we also plan to investigate the concurrent use of multiple high resolution timers, to check whether the RT Patch can effectively support *multiple* real-time tasks making use of them.

References

[1] IEC, *Industrial Communication Networks – Fieldbus specifications – Part 3-12: Data-Link Layer Service Definition – Part 4-12: Data-link layer protocol specification – Type 12 elements*, Dec. 2007, ed 1.0, IEC 61158-3/4-12.

[2] FSMLabs Inc. Real-time management systems (RTMS) overview. [Online]. Available: http://www.fsmlabs.com/

[3] LynuxWorks Inc. Embedded Linux: BlueCat Linux operating system. [Online]. Available: http://www.lynuxworks.com/

[4] MontaVista Software, Inc. Real-time Linux development from MontaVista. [Online]. Available: http://www.mvista.com/

[5] Politecnico di Milano, Dip. di Ingegneria Aerospaziale. RTAI 3.4 user manual. [Online]. Available: https://www.rtai.org/

[6] Wind River Inc. Wind River Linux. [Online]. Available: http://www.windriver.com/

[7] Xenomai development team. Xenomai: Real-time framework for Linux. [Online]. Available: http://www.xenomai.org/

[8] S. Rostedt and D. V. Hart, "Internals of the RT Patch," in *Proc. of the Ottawa Linux Symposium*, Ottawa, Canada, Jun. 2007.

[9] IgH. EtherLab – EtherCAT master. [Online]. Available: http://www.etherlab.org/

[10] Beckhoff Automation GmbH. EL2202 2-channel digital output terminal. [Online]. Available: http://www.beckhoff.com/

[11] ——. CX1020 basic CPU module. [Online]. Available: http://www.beckhoff.com/

[12] ——. EL1252 2-channel digital input terminal with time stamp. [Online]. Available: http://www.beckhoff.com/

[13] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance comparison of VxWorks, Linux, RTAI, and Xenomai in a hard real-time application," *IEEE Transactions on Nuclear Science*, vol. 55, no. 1, pp. 435–439, Feb. 2008.

[14] K. Köker, "Embedded RTOS: Performance analysis with high precision counters," in *Autonomous Robots and Agents*. Springer Berlin/Heidelberg, 2007, ch. 20, pp. 171–179.

[15] M. Mossige, P. Sampath, and R. G. Rao, "Evaluation of Linux rt-preempt for embedded industrial devices for automation and power technologies – a case study," in *Proc. 9th Real-Time Linux Workshop*, 2007.

[16] A. Siro, C. Emde, and N. Mc Guire, "Assessment of the realtime preemption patches (RT-Preempt) and their impact on the general purpose performance of the system," in *Proc. 9th Real-Time Linux Workshops*, 2007.

[17] L. Abeni, A. Goel, C. Krasic, J. Snow, and J. Walpole, "A measurement-based analysis of the real-time performance of Linux," in *Proc. 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTTAS)*, 2002, pp. 133–142.

[18] W. Betz, M. Cereia, and I. Cibrario Bertolotti, "Experimental evaluation of the Linux RT Patch for real-time applications," in *Proc. of the 14th IEEE Conference on Emerging Technologies and Factory Automation*. IEEE Press, Sep. 2009.