Performance of a Real-Time EtherCAT Master Under Linux

Marco Cereia, Ivan Cibrario Bertolotti, *Member, IEEE*, and Stefano Scanzio

***Abstract*—The adoption of open-source operating systems for the execution of real-time applications is gaining popularity, even in the networked control systems domain, due to cost and flexibility reasons. However, as opposed to their commercial counterparts, the actual performance level to be expected from them is still little known and may often depend on the kind of real-time extension being used, its configuration, and the overall software load of the system, including best-effort components.**

In this paper, an open-source EtherCAT master supported by a popular real-time extension for Linux, the RT Patch, is thoroughly evaluated with long-term measurements, which build confidence on the suitability of the proposed approach for real-world applications. Special attention is devoted to the unexpected, adverse effect that some best-effort components, for instance, graphics applications, may have on the overall real-time characteristics of the system. For reference, the proposed approach is also compared with RTAI, a more traditional and well-known real-time extension for Linux already in use for demanding applications.

***Index Terms*—Industrial control systems, open-source operating systems, real-time and embedded systems.**

I. Motivation and Related Work

In recent years, the real-time capabilities of open-source operating systems, most notably Linux, have been enhanced considerably. In some cases, this trend led to the introduction of commercial products actively used by the real-time systems community [1]–[3]. Open-source initiatives include RT Linux/GPL, RTAI, Xenomai, and AQuoSA [4]–[7]. All of them are hybrid systems, i.e., they support both real-time and best-effort tasks concurrently, and are based either on the *double kernel* method—in which a small, real-time kernel is nested into the standard operating system kernel—or on the enhancement of the Linux scheduler.

Recently, a different approach to real-time execution in Linux has been proposed as well, with the development of the RT Patch [8]. Its most promising advantage is a better integration with standard Linux distributions, which will make software development easier and more flexible. By contrast, its raw performance can be expected, by intuition, to be worse.

On the other hand, the increasing diffusion of industrial networks based on Ethernet signaling such as, for example, EtherCAT [9], enables a complete networked control system to be built without even installing any custom interface card on the controller. Moreover, due to the availability of open-source EtherCAT master components like IgH EtherLab [10], this kind of systems can nowadays be built entirely from open-source components.

Several different real-time extensions of Linux have been chosen for a variety of real-time applications in the recent past, ranging from FireWire resource management to 802.11 TDMA access schemes, for example, [11]–[15]. Their real-time performance has been thoroughly evaluated, also because Linux has sometimes been used as a platform to develop and benchmark novel scheduling paradigms and algorithms [16]–[18]. In some cases, they have been shown to be worth of consideration even with respect to very well-respected, commercial counterparts [19]. Additional work is still ongoing to further analyze and improve their interrupt latency [20].

For what concerns the RT Patch, performance data are instead in scarce supply. Most work [21], [22] focused on synthetic test cases, in which data acquisition is performed by instrumenting the real-time application under test and mutual interference cannot be ruled out.

In addition, even if the test cases have been carefully conceived, they might not be a completely faithful abstraction of a networked control system, because they only scrutinize periodic scheduling and asynchronous event handling latencies separately. They do not consider the effects of device interaction and do not assess the total jitter incurred by a periodic networked control task, from task activation to hardware actuation. A hardware-based acquisition system was used in [23], but the focus of the paper was on evaluating the impact of the Realtime Preemption Patches (RT-Preempt) on the best-effort part of the system, rather than on the real-time side.

These data are especially important for real-world application designers, because it has already been shown [24] that the RT Patch behavior heavily depends on how the RT Patch has been *configured*. Moreover, it can also unexpectedly be influenced by the concurrent execution of some kinds of *best-effort* tasks and other internal kernel activities. A thorough comparison with a double kernel system is also very useful to help the designer achieve a satisfactory tradeoff between performance and flexibility.

The goal of this paper is therefore to measure the real-time performance of a fully open-source EtherCAT master based on IgH EtherLab and the RT Patch, in presence of different classes of interfering loads, which include most of the cases of practical interest. Among all the performance indicators of importance for an industrial network [25], the focus of the paper is on the worst-case jitter of a periodic control task, which was already shown to be a significant issue in software-based communication scheduling [26]. The results are then compared with those

Manuscript received February 16, 2011; revised May 16, 2011; accepted June 20, 2011. Date of publication September 06, 2011; date of current version November 09, 2011. This work was supported in part by Regione Piemonte in the framework of the POR-FESR 2007/2013 Project Flex-Mech "New Advanced Mechanical Systems for Flexible and Customized Production" under Grant 0127000009B). Paper no. TII-11-062.

The authors are with the National Research Council of Italy, Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT), I-10129 Torino, Italy (e-mail: ivan.cibrario@polito.it).

Digital Object Identifier 10.1109/TII.2011.2166777

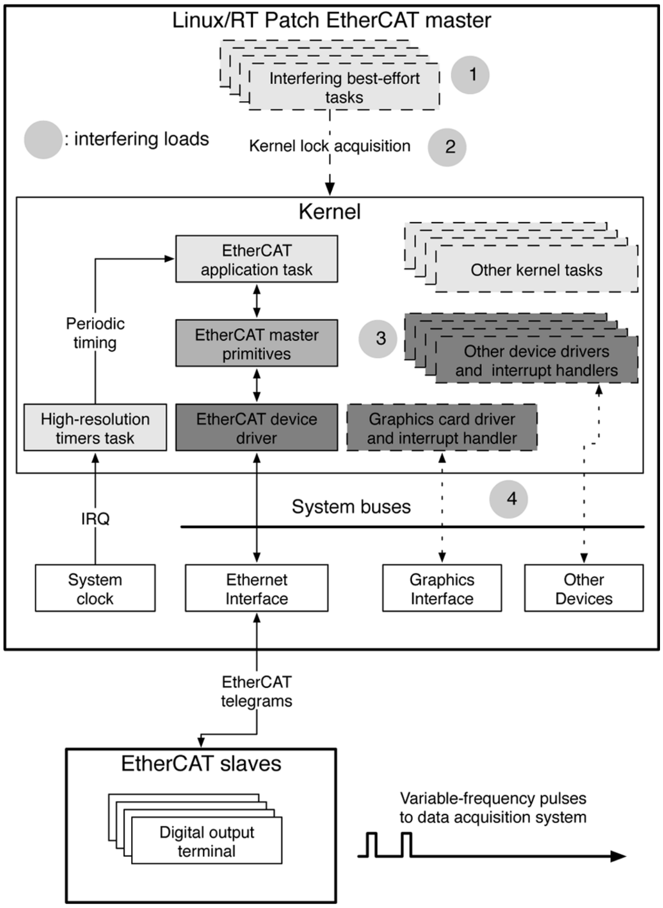
Linux/RT Patch EtherCAT master
Interfering best-effort tasks
1
: interfering loads
Kernel lock acquisition
2
Kernel
EtherCAT application task
Other kernel tasks
Periodic timing
EtherCAT master primitives
3
Other device drivers and interrupt handlers
High-resolution timers task
EtherCAT device driver
Graphics card driver and interrupt handler
System buses
4
IRQ
System clock
Ethernet Interface
Graphics Interface
Other Devices
EtherCAT telegrams
EtherCAT slaves
Digital output terminal
Variable-frequency pulses to data acquisition system

Fig. 1. Block diagram of the real-time system under test, highlighting different classes of interfering loads.

obtained with a double kernel, RTAI-based system. The most important experiments lasted for more than one day, building additional confidence on system stability and on the significance of the experimental data.

To this purpose, Section II briefly describes the experimental setting and the different classes of interfering loads being considered. Then, Section III discusses in detail the effect of these interfering loads and contains the comparison with RTAI. Section IV is devoted to the most troublesome interfering load, involving memory bus contention. Finally, Section V contains some concluding remarks.

II. Experimental Setting and Interfering Loads

The experimental setting is similar to that described in [24] and includes two main components: a PC-based EtherCAT System Under Test (SUT), shown in Fig. 1, and a data acquisition system with a performance comparable to that discussed in [27]. In the figure hardware components, tasks, functions, and device drivers (with their associated interrupt handlers) are shown as boxes, filled with progressively darker shades of gray. Solid arrows represent either a synchronous function call or a causality relationship within the modules being evaluated, for instance, the activation of a task when an event occurs, or a device interaction. Dashed boxes and arrows denote interfering components and the concurrent activities they perform, respectively.

The PC used for the evaluation has an Intel Core2Duo E6750 dual-core CPU running at 2.66 GHz and 4 GB of dual-channel DDR2 RAM. In order to make its performance as similar as possible 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 network interface card carrying EtherCAT traffic is based on the Realtek RTL8139 chip, which is directly supported by the IgH EtherLab software distribution.

Within the SUT an application task generates a test signal, consisting of a train of variable-frequency pulses, using the operating system's timing primitives and the PC clock as time reference. The test signal is put out through a high-speed, asynchronous digital output terminal (Beckhoff EL2202 [28]), with a switching time of less than 1 μs. 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.

The data acquisition system captures the rising edges of the waveform generated by the SUT and stores them for further analysis. All measurements are made by reference to a free-running acquisition clock, with a resolution of 1 ns and a precision of 10 ns. As in [24], the data acquisition system is also based on EtherCAT (it revolves around a Beckhoff EL1252 time stamping input terminal). However, it has no impact on the SUT behavior, because it is completely independent from the SUT itself and does not imply any modification to its source code. For this reason, its internal architecture will not be discussed further.

Fig. 1 also illustrates the different ways a best-effort load executed by the SUT can interfere with the real-time components of the SUT itself when using Linux/RT Patch. In particular, by looking at the internal structure of the system under test, four distinct and independent kinds of interference can be identified.

1) Any best-effort task competes for execution against the real-time components. This is the most basic kind of interference, and must be taken care of at the processor scheduling level.
2) When a best-effort task performs certain kinds of system interaction—for example, accesses to the `/proc` file system [29]—it may need exclusive access to some kernel data structures and hence interfere with the real-time components at the level of kernel locking and internal synchronization.
3) Most operating systems grant a relatively high priority to interrupt handling, regardless of the priority of the task which triggered the interrupt. A best-effort task can therefore interfere with the real-time components by causing a large number of interrupt requests through, for instance, an I/O device.
4) Many fast I/O devices can become bus masters on a number of shared, single-port system buses and are able to impose a significant load on them. As long as the device keeps, for instance, the main memory bus busy, the processor cannot use it and may stall. A best-effort task can hence interfere with real-time components by initiating many operations on those devices.

Since any real-world, best-effort task will likely produce a combination of those different kinds of interference, their impact has been considered and analyzed separately in Section III. The RTAI-based software architecture is slightly different, because interrupt handling is delegated to the Adeos [30] interrupt pipeline and the real-time tasks are scheduled by their own nested kernel, but the categorization of interfering loads just discussed still has relevance.

III. Effect of Interfering Best-Effort Tasks

A. Short-Term Behavior

The first set of experiments has been carried out with a real-time task period of 1000 μs, acquiring a total of 10^6 samples. This corresponds to an experiment length of 1000 s, that is, about 16 min. The three interfering loads being considered correspond to interference classes 1), 2), and 3) described in Section II. Namely, they consist of the following.

1) A CPU-bound Python application executing floating-point arithmetic in a tight, infinite loop. It is able to fully utilize the CPU without posing any other significant kind of load to the system.
2) The `top` system utility, accessing the `/proc` file system at maximum speed through the best-effort part of the Linux kernel, and thus repeatedly acquiring and releasing kernel locks (K. Locks). The average load amounts to 530 system calls per second.
3) A best-effort I/O-bound application, based on the `dd` system utility, generating a sustained stream of interrupts requests (IRQs) from the hard disk controller. With a transfer block size of 512 bytes, the application generates a sustained load of 500 IRQs per second.

The data acquisition system timestamps the test waveform's rising edges and uses this information to determine the distribution of their jitter. The experimental data are corrected before analysis, as described in [24], in order to compensate for the unavoidable frequency error between the generation and the acquisition clocks, due to oscillator tolerances, using a linear regression. Hardware clock drifts due, for example, to temperature variations, are also smoothed out with a moving average comprising a window of 16 000 samples. The window width has been chosen empirically, as a compromise between being able to track the clock drift accurately, and avoid masking the local, short-term jitter caused by software at the same time. A moving average on n samples acts as a rough low-pass filter in the frequency domain, with the first zero at $2\pi/n$ radians per sample period. For $n = 16\,000$, the main passband is between 0 and $3.9 \cdot 10^{-4}$ radians per sample period and the jitter is believed to not have any component in that frequency range.

For the RT Patch, the priorities of the real-time tasks have been assigned as described in [24] for all experiments, to avoid any timing anomaly due to routing cache flushing and reduce the interference due to other kernel tasks. Specifically, the high-resolution timer task priority has been raised one step above the other interrupt sources, and the application task priority has been set to the maximum real-time priority allowed by the FIFO scheduling policy.

Table I shows the summary statistics of the jitter measured during the experiments. For each kernel type/configuration, and for each class of interfering load, the usual mean value (μ) and standard deviation (σ) are given, as well as the minimum (Min) and maximum (Max) value of the jitter J. The last two quantities are useful because they give a more accurate idea of the worst-case behavior of the system.

TABLE I
Summary Statistics of the Jitter J With Different Interfering Loads, 1000 μs Real-Time Period, 10^6 Samples

Kernel type/configuration		**Interfering load** **None**	**CPU**	**K. Lock**	**IRQ**
Standard	μ_J	0.002	0.058	0.001	-0.001
	σ_J	5.724	5.554	4.571	5.752
	Min_J	-0.411	-0.774	-3.771	-4.378
	Max_J	**1434.0**	**1430.2**	**1099.7**	**1151.0**
RT Patch	μ_J	0.001	0.063	0.063	0.000
	σ_J	0.120	0.426	0.900	0.632
	Min_J	-0.424	-0.978	-0.798	-1.171
	Max_J	1.800	5.143	6.349	10.400
RTAI	μ_J	0.063	0.060	0.063	0.064
	σ_J	0.116	0.173	0.384	0.322
	Min_J	-1.185	-0.179	-0.612	-1.240
	Max_J	2.489	2.946	3.789	7.976

All values are expressed in μs

The results, as anticipated, highlight the inability of the standard Linux kernel to cope with the real-time requirements because, in the worst case, the jitter exceeds the real-time task period, even without any interfering load. On the contrary, both the RT Patch and RTAI are only marginally affected by these kinds of load. The greater impact of the IRQ load on the RT Patch than on RTAI can be explained recalling that the RT Patch uses a set of separate tasks, called *kernel threads* in the Linux documentation, for interrupt handling. They have, by construction, a greater scheduling overhead than the Adeos interrupt pipeline within RTAI.

The next set of experiments was focused on the IRQ load, the one with the worst effects in the previous set of experiments. Even if the *K. Lock* load had an even more pronounced influence on the jitter's standard deviation, it was no longer considered because its effect on the worst-case behavior of the system, expressed by the minimum and maximum jitter, was less important.

These experiments were aimed at checking whether the amount of real-time load had any effect on the jitter. To this purpose, the period of the real-time task was shortened until the limit of the data acquisition system was reached, that is, 250 μs. Since the standard Linux kernel was unable to meet less stringent requirements, it was no longer evaluated.

The experimental results, listed in Table II, show that the increased real-time load has negligible effects on both the minimum and maximum observed jitter (labeled Min_J and Max_J in Table II), and on its standard deviation (labeled σ_J). Hence, the system is able to sustain a real-time task period from 1000 μs down to 250 μs with approximately the same amount of *absolute* jitter.

With a decreasing real-time task period, the *relative* jitter—defined in this context as the ratio of the absolute jitter over the task period—will naturally increase. In any case, the worst absolute jitter observed during all the experiments, 13.065 μs, still represents only about 5.2% of the period, a value likely to be acceptable for many control applications.

Fig. 2 shows the time trace of the jitter for the most demanding situation given in Table II, that is, a real-time task period of 250 μs. It further remarks that RTAI sustains the IRQ

TABLE II
SUMMARY STATISTICS OF THE JITTER J, WITH IRQ LOAD AND VARYING REAL-TIME PERIOD, 10^6 SAMPLES

Kernel type/configuration		**Real-time period** 1000 μs	500 μs	250 μs
RT Patch	μ_J	0.000	0.031	0.015
	σ_J	0.632	0.635	0.570
	Min_J	-1.171	-1.429	-1.008
	Max_J	10.400	**13.065**	10.099
RTAI	μ_J	0.064	0.031	0.016
	σ_J	0.322	0.303	0.269
	Min_J	-1.240	-0.516	-0.560
	Max_J	7.976	10.014	5.549

All values are expressed in μs

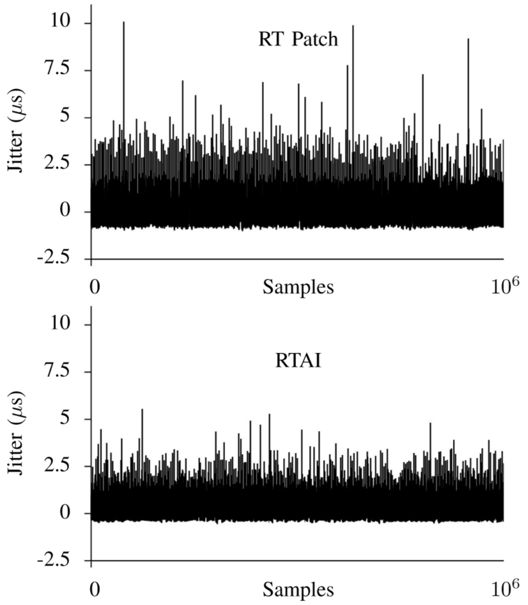

Fig. 2. Time trace of the jitter J with IRQ load, corresponding to the rightmost column of Table II: 10^6 samples with a real-time task period of 250 μs.

load better than the RT Patch, and exhibits a smaller standard deviation, a smaller minimum/maximum jitter excursion, and a smaller number of anomalous peaks.

B. Long-Term Experiments

Albeit the short-term results presented in Section III-A look encouraging for the RT Patch, its behavior has been more thoroughly evaluated with several long-term measurements, in order to build a greater confidence on the suitability of the proposed approach for real-world applications and increase the likelihood of capturing anomalous events, even if they have a very low probability of occurrence. To keep the results commensurable with respect to the previous ones, the behavior of RTAI has again been taken as reference, and the experiments have been conducted using a real-time task period of 250 μs. The total experiment duration has instead been increased to 24 h, corresponding to about $3.45 \cdot 10^8$ samples.

However, due to the large amount of experimental data to be collected in this case, and the limitations of the data acquisition system, the compensations discussed in Section III-A have no longer been applied. On the contrary, instead of measuring the *jitter J* directly, the frequency distribution of the real-time task *period S*—as inferred from the time distance between consecutive raising edges of the test waveform—has been computed on the fly, without storing the individual timestamps.

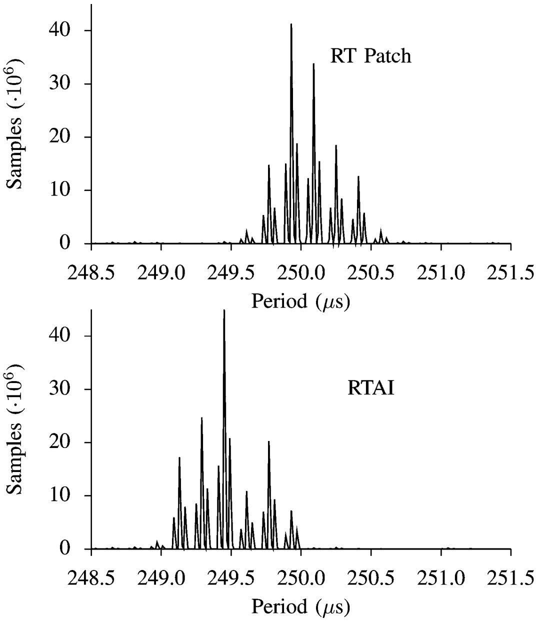

Fig. 3. Frequency distribution of the real-time period S, without interfering load. Nominal task period: 250 μs, $3.45 \cdot 10^8$ samples (experiment duration: 24 hours).

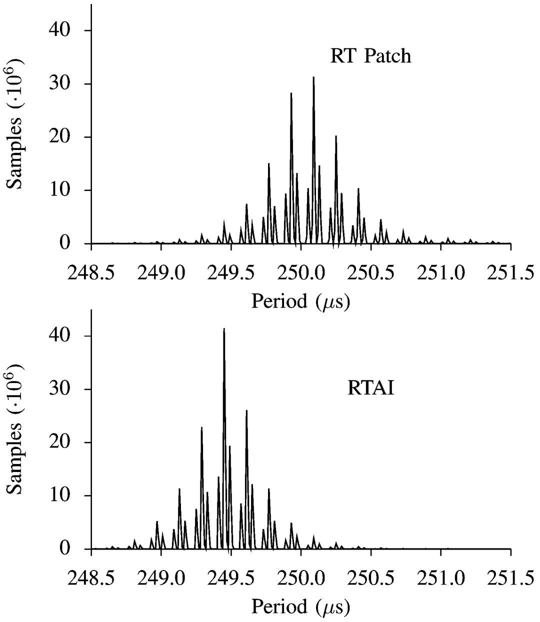

Fig. 4. Frequency distribution of the real-time period S, with IRQ load. Nominal task period: 250 μs, $3.45 \cdot 10^8$ samples (experiment duration: 24 h).

This measurement method still provides very valuable information, because for many real-time control applications the distribution of the real-time period is an important factor to be kept under control, at least as important as the jitter with respect to the nominal release time. For this reason, it has been used in many related works, for instance, [12], [19], [21], [22], [29].

Figs. 3 and 4 show the measured frequency distributions for the best and worst kind of interfering load encountered so far, namely: 1) no interfering load and 2) IRQ interfering load.

TABLE III
SUMMARY STATISTICS OF THE REAL-TIME PERIOD S IN A 24 HOUR EXPERIMENT, 250 μs REAL-TIME PERIOD, $3.45 \cdot 10^8$ SAMPLES

Kernel type/	μ_S	σ_S	Min_S	Max_S
configuration			(μs)	
RT Patch	250.049	0.704	241.720	257.960
(no load)			($\mu - 8.329$)	($\mu + 7.911$)
RT Patch	250.053	0.443	238.600	261.600
(IRQ load)			($\mu - 11.453$)	($\mu + 11.547$)
RTAI	249.462	0.658	243.520	255.320
(no load)			($\mu - 5.942$)	($\mu + 5.858$)
RTAI	249.462	0.655	240.800	258.280
(IRQ load)			($\mu - 8.662$)	($\mu + 8.818$)

Table III lists the corresponding summary statistics. When interpreting these experimental data, it should be remarked that measuring the period instead of the jitter has a few important consequences.

- Any systematic frequency mismatch between the time bases used by the generation and the acquisition system is no longer nullified, but induces a shift of the frequency distribution around the nominal period. The overall shape of the distribution is not affected.
- Uncompensated clock drifts make the measured values wander around their mean, and increase the spread of the frequency distribution. The experimental results are still conservative with respect to an industrial system with a higher quality oscillator.
- Low-frequency jitters are no longer filtered out as a side effect of the drift compensation algorithm. Therefore, their contribution is still present in the experimental data and is part of the data being analyzed.
- The distribution of the real-time task period and its summary statistics are related to, but are not strictly the same as, the distribution and summary statistics of the jitter.

If we denote with $r_i = kT$ the nominal release time of the ith instance of the real-time task with period T and represent the jitter affecting the ith release by the random variable J_i, the actual release times $R_{i-1}, \ldots, R_{i+1}$ will be

$$\begin{cases} R_{i-1} = r_{i-1} + J_{i-1} \\ R_i = r_i + J_i \\ R_{i+1} = r_{i+1} + J_{i+1}. \end{cases} \tag{1}$$

Since it is assumed that the statistical properties of the jitter do not change during the experiment, the random variables $J_i \, \forall i$ will all be drawn from the same distribution.

If we now consider two adjacent samples of the real-time task period acquired by the measurement, S_i and S_{i+1}, we obtain

$$\begin{cases} S_i = R_i - R_{i-1} = (r_i - r_{i-1}) + J_i - J_{i-1} \\ S_{i+1} = R_{i+1} - R_i = (r_{i+1} - r_i) + J_{i+1} - J_i. \end{cases} \tag{2}$$

But both $r_i - r_{i-1}$ and $r_{i+1} - r_i$ are equal to the nominal real-time task period T, hence

$$\begin{cases} S_i = T + J_i - J_{i-1} \\ S_{i+1} = T + J_{i+1} - J_i. \end{cases} \tag{3}$$

From (3), the expected value of S_i, denoted by $E[S_i]$ is

$$E[S_i] = E[T + J_i - J_{i-1}] = T + E[J_i - J_{i-1}]. \tag{4}$$

However, regardless of the distribution of J_{i-1} and J_i, it is: $E[J_i - J_{i-1}] = E[J_i] - E[J_{i-1}]$. Moreover, $E[J_i] = E[J_{i-1}]$, because J_{i-1} and J_i are drawn from the same distribution, hence

$$E[S_i] = T. \tag{5}$$

The mean of the frequency distribution μ_S is therefore a correct estimator of T and will converge to T for a large number of samples. The variance of S_i, denoted with $\mathrm{var}\{S_i\}$is

$$\mathrm{var}\{S_i\} = \mathrm{var}\{T + J_i - J_{i-1}\}. \tag{6}$$

The nominal period T is a constant, hence $\mathrm{var}\{T\} = 0$. For what concerns $\mathrm{var}\{J_i - J_{i-1}\}$, in general, it is

$$\mathrm{var}\{J_i - J_{i-1}\} = \mathrm{var}\{J_i\} + \mathrm{var}\{J_{i-1}\} - 2\mathrm{cov}(J_i, J_{i-1}) \tag{7}$$

where $\mathrm{cov}(J_i, J_{i-1})$ is the covariance of J_i and J_{i-1}. Both random variables are drawn from the same distribution, therefore $\mathrm{var}\{J_i\} = \mathrm{var}\{J_{i-1}\}$ and we can write

$$\mathrm{var}\{S_i\} = 2\mathrm{var}\{J_i\} - 2\mathrm{cov}(J_i, J_{i-1}). \tag{8}$$

Solving (8) with respect to J_i gives

$$\mathrm{var}\{J_i\} = \frac{1}{2}\mathrm{var}\{S_i\} + \mathrm{cov}(J_i, J_{i-1}). \tag{9}$$

If J_i and J_{i-1} were statistically uncorrelated, it would be $\mathrm{cov}(J_i, J_{i-1}) = 0$ and we could write

$$\mathrm{var}\{J_i\} = \frac{1}{2}\mathrm{var}\{S_i\}. \tag{10}$$

It would then be possible to directly compute the variance of the jitter $\mathrm{var}\{J_i\}$ from the variance of the period $\mathrm{var}\{S_i\}$, the latter being estimated with the σ_S^2 of the long-term experimental results.

Unfortunately, this is not the case, especially for the RT Patch. The covariance of the jitter measured in the short-term experiments can be used to approximate $\mathrm{cov}(J_i, J_{i-1})$ with an IRQ load and a period of 250 μs, and obtain

$$\begin{aligned} &\mathrm{cov}(J_i, J_{i-1}) \simeq 0.292 \text{ for RT Patch} \\ &\mathrm{cov}(J_i, J_{i-1}) \simeq 0.048 \text{ for RTAI.} \end{aligned} \tag{11}$$

Since $\mathrm{cov}(J_i, J_{i-1}) > 0$, the jitters affecting two consecutive release times R_i are directly correlated and part of the jitter will be masked off in the samples S_i. However, it is possible to compensate for this fact and (9) can still be used to calculate a (less accurate) estimation of the variance of the jitter $\mathrm{var}\{J_i\}$. To do this, we use the covariance (11) calculated from the short-term experiments as the best available estimation of the unknown, true jitter's covariance. In specific, we obtain

$$\begin{aligned} &\mathrm{var}\{J_i\} \simeq 0.098 + 0.292 = 0.390 \text{ for RT Patch} \\ &\mathrm{var}\{J_i\} \simeq 0.215 + 0.048 = 0.263 \text{ for RTAI.} \end{aligned} \tag{12}$$

Those values can then be used to calculate σ_J as $\sqrt{\mathrm{var}\{J_i\}}$ and obtain

$$\begin{aligned}\sigma_J &\simeq 0.624 \text{ for RT Patch}\\ \sigma_J &\simeq 0.513 \text{ for RTAI.}\end{aligned} \tag{13}$$

Therefore, the estimate of σ_J from the long-term experiments is either very close to, or greater than, the worst estimates obtained in the set of short-term experiments listed in Table II (0.635 and 0.322, respectively).

This relationship shows that, in general, the long-term results are in good agreement with respect to the short-term measurements discussed in Section III-A. Most importantly, the behavior of both the RT Patch and RTAI, with respect to the IRQ load, does not get any worse in the long term. On the other hand, it must also be remarked that for RTAI the short-term experiments were significant, but yet unable to fully capture the variability of the jitter, as the long-term experiments did.

The minimum and maximum values of S_i are also in good agreement with the minimum and maximum jitter J_i, recalling that if one denotes with $|J|$ the worst-case absolute value of the jitter, the difference of S_i with respect to its distribution mean value will either be zero or $2|J|$. The extreme cases occur when J_{i-1} and J_i are deterministically related either directly or inversely, that is $J_{i-1} = \pm J_i$ meaning that the two consecutive release times R_{i-1} and R_i are both affected by the same amount of jitter, barring the sign.

For the long-term experimental results, it can also be seen that the average real-time task period is not the same for the two operating systems. This small discrepancy (of about 600 ns) can be explained by recalling that, albeit both operating systems derive their timings from the same hardware source (the APIC timer on the machine used for the experiments) they use it in very different ways: the timer operating mode is not the same (periodic versus one-shot mode) and the software modules that program the timer are also different. Although RTAI provides a way to evaluate and correct any systematic timing error by means of an offline calibration utility, this feature has not been used for the experiments, because it makes the mean period more accurate, but shall have no effect on its predictability, which is the aspect being evaluated here.

IV. GPU Interference and Memory Contention

The last kind of test considered in this paper evaluates the effect of system bus contention and corresponds to the fourth class of interfering load presented in Section II. In a personal computer, most internal buses (for both I/O and memory) can have multiple, potential bus masters, but they are single-ported and thus support at most one transaction at a time.

In general, bus access is regulated by a fair round-robin arbiter within the north bridge chip, as illustrated in the simplified block diagram shown in Fig. 5. Some devices, such as integrated graphics controllers with a frame buffer residing in main memory, may need a guaranteed amount of main memory bandwidth to work properly. Accordingly, they are always granted a higher priority than any other, most notably higher than the CPU. In any case, the bus arbitration algorithm does not take into account the priority of the task requesting the transfer in any way.

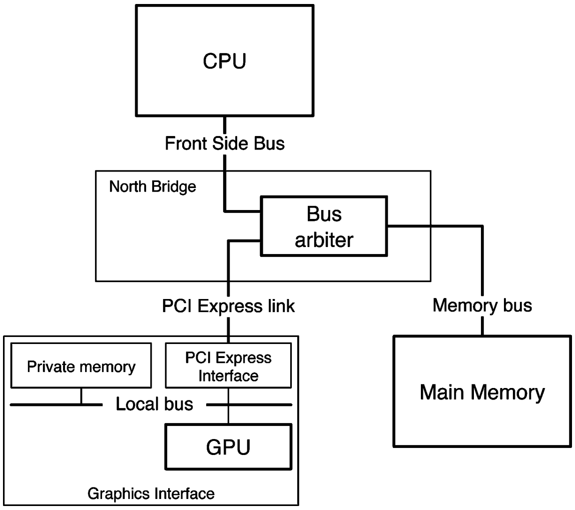

Fig. 5. Simplified block diagram of the various system buses found in a typical personal computer, showing a possible contention between the GPU and the CPU at the main memory access level.

It is therefore possible that a low-priority, best-effort task interferes with a higher priority, real-time task at the bus arbitration level, simply by asking an I/O device to become bus master and perform one or more bus transfers. It can also be expected that this kind of interference will be extremely difficult to avoid by software because, as discussed, it involves a very low-level, hardware-based arbitration mechanism.

For a real-time system, which also supports best-effort tasks, this used not to be a significant problem, because the I/O devices used by best-effort tasks had a relatively low performance. In more recent years, however, even the most inexpensive personal computers have been equipped with very fast I/O devices, most notably accelerated graphics interfaces equipped with a Graphics Processing Unit (GPU). Moreover, they are linked to the main memory with very fast buses such as, for instance, the PCI Express link shown in Fig. 5. As a result, they are able to generate a significant amount of bus traffic.

At the same time, the device drivers associated with this kind of device are very difficult to develop and the available drivers are usually closed-source, hence the application software developer has little or no control on how the device is actually being used at that level. It is therefore very important to assess the impact of a commercial, off-the-shelf GPU driver on the overall performance and predictability of a real-time system.

In order to do this a set of short-term measurements, like those discussed in Section III-A for the other kinds of interfering load, has been carried out. During the 16 min of experiment time, a best-effort task has been used to trigger a sequence of data transfers between the GPU card and the main memory of the PC, with a linear sweep of the data transfer size from 0.5 to 2.6 MB.

The discrete GPU card used for the experiments is a nVidia GeForce 8800 GTS 512, with 512 MB of local memory and a total of 128 processing cores running at 1.62 GHz, divided into 16 MultiProcessors (MPs). A PCI Express 2.0 interface with 16 lanes at 2.5 Gb/s connects the graphics card to the PC. It was

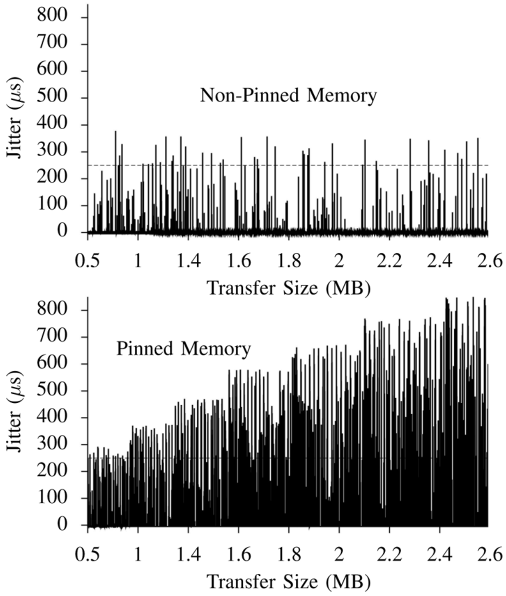

Fig. 6. The trace of the RTAI jitter J, with sustained GPU data transfers to pinned versus nonpinned memory, depending on transfer block size. 10^6 samples with a real-time task period of 250 μs.

driven by a CUDA application, using the nVidia CUDA drivers and runtime, both at version 3.20.

It should be remarked that, by all contemporary standards, this is definitely a low-end graphics card. It is therefore likely that cards with comparable performance are already being used, and will be adopted in many embedded systems in the near future. Similarly, CUDA is gaining popularity as a convenient framework for parallel computing in different application areas [31].

Two different kinds of main memory buffer have been used for the transfer.

1) *Nonpinned* memory. When this kind of memory is in use, the data transfer buffer is divided into a certain number of pages, and each of them can be paged in and out of main memory individually, by the operating system's virtual memory manager. When switching from one page to the next during the data transfer and avoid undue page faults, it is therefore necessary to make sure that the new page is resident in main memory and determine its physical address. This action cannot be carried out autonomously by the GPU hardware. Rather, it requires a software interaction between the GPU device driver and the rest of the operating system.
2) *Pinned* memory. A data transfer buffer composed of this kind of memory is still divided into pages, but they are forced to permanently reside in main memory, and the virtual memory manager is completely disabled for them. Hence, their physical address can be calculated once and for all. In this case, any device capable of scatter-gather data transfers to/from main memory (as virtually all modern GPUs are) will be able to carry out the whole data transfer without any software intervention.

The results obtained by the best-performing operating system considered so far, i.e., RTAI, are shown in Fig. 6. As before, the real-time task period has been set to 250 μs and the figure shows the time trace of the jitter. From the experimental data, two main considerations can be drawn.

- Albeit RTAI was able to effectively partition the real-time tasks from all the other kinds of interfering load considered so far, it is unable to do so in this case. Even when the GPU data transfer involves nonpinned memory, the most favorable case, the amount of jitter often exceeds the real-time task period, represented by a horizontal, dashed line in the figure.
- The impact of the interfering load heavily depends on low-level details of the data transfer such as, in this case, the kind of memory holding the data buffer and the transfer size.

It can easily be seen that, although data transfers involving nonpinned memory undoubtedly have an important, negative influence on the real-time behavior of the system, the amount of interference is still bounded, regardless of the data transfer size. As a consequence, this scenario might still be acceptable for less demanding tasks. Instead, when using pinned memory, the interference is an almost linear function of the data transfer size and has not got any obvious upper limit, so that is can easily disrupt any real-time requirement and make the system completely unsuitable for use. For this reason, no further long-term experiments have been carried out in this configuration.

Even if it is possible, in principle, to contain the issue by using the "right" kind of memory and/or an appropriate maximum data transfer size, these choices are left to the best-effort application programmer's discipline and may be difficult to enforce. Moreover, those are details the average programmer may even be unaware of.

Last, but not least, the situation is likely to be different for a different graphics card make or model, or for another device driver version. It will therefore be difficult to keep the system up to date with respect to best-effort software and hardware upgrades and, at the same time, make sure that the real-time part of the system still works as required.

The same experiment was also repeated for the RT Patch. The results are not shown for conciseness, because they are very alike to what has been obtained for RTAI. The underlying issues they point out, for this kind of interfering load, are indeed the same just discussed for RTAI itself.

V. Concluding Remarks

The topic of supporting real-time applications using a suitable extension of the Linux operating system is becoming more and more important to reduce the cost and enhance the flexibility of embedded systems, also by supporting both real-time and best-effort tasks on the same hardware.

It is therefore very important to determine the actual performance level to be expected from this kind of systems, depending on the kind of real-time extension being used and its configuration. Assessing the amount of undue interference the best-effort part of the application may have on real-time tasks is of paramount importance, too.

In this paper, two popular, and very different, approaches to real-time support for Linux have been considered, namely, RTAI [5] and the RT Patch [8]. They have been evaluated by means of a synthetic test application, representative of a networked control system, and were subject to several interfering loads, deemed to be representative of a wide class of best-effort applications.

The experimental results show two contrasting facets, both of interest to the embedded system designer. For most classes of interfering loads, both RTAI and the RT Patch were able to fulfill stringent real-time requirements for a periodic task, supporting periods down to 250 μs with a very limited jitter, on the order of 13 μs. Moreover, their favorable characteristics have been confirmed with long-term measurements, about 24 h long, thus building confidence on the suitability of the proposed approach for real-world applications.

These applications will usually consist of a set of periodic tasks, with well-known real-time characteristics. Only one of them will be the EtherCAT master, while the others will implement the control-command algorithms and update a shared process image. A suitable scheduling theory, for instance, [32], can then assess their performance, provided all sources of interference are known. The results presented in this paper can then be used to assess both the jitter endured by the network traffic generated by EtherCAT master task, and the additional interference to which the other tasks will be exposed. Both aspects must necessarily be considered in the scheduling theory to get accurate results, but are not directly taken into account by the theory itself, because they are out of its scope.

On the other hand, it has also been shown that the partitioning mechanism between real-time and best-effort tasks is far from being perfect, and shows its limits for other kinds of interfering loads of practical interest, for example, GPU interaction. Therefore, contrary to what one might expect, the real-time and best-effort parts of the system cannot be considered as two fully independent software loads. Instead, they must still be designed and implemented together, in order to keep their undue interference under control.

An additional, interesting detail emerging from the experiments (Tables II and III in particular) is that the standard deviation of the jitter decreases, instead of increasing, for smaller task periods and/or higher interfering loads. Although it is not severe enough to limit the usability of the system, this seemingly counterintuitive fact is still being investigated. Two contributing factors identified so far are the higher probability/extent of cache pollution due to the execution of best-effort tasks when the real-time load is low, and a more aggressive intervention of CPU and platform power saving features when the overall system load is low.

References

[1] "Embedded Linux: BlueCat linux operating system," LynuxWorks Inc., 2008. [Online]. Available: http://www.lynuxworks.com/

[2] "Real-time Linux development from MontaVista," MontaVista Software, Inc., 2005. [Online]. Available: http://www.mvista.com/

[3] "Wind River Linux, Wind River Inc., 2011. [Online]. Available: http://www.windriver.com/

[4] M. Barabanov and V. Yodaiken, "Introducing real-time Linux," *Linux J.*, no. 34, pp. 19–23, Feb. 1997.

[5] P. Mantegazza, E. Bianchi, L. Dozio, S. Papacharalambous, S. Hughes, and D. Beal, "RTAI: Real-time application interface," *Linux J.*, no. 72, pp. 142–148, Apr. 2000.

[6] P. Gerum, "Xenomai—Implementing a RTOS emulation framework on GNU/Linux," 2004. [Online]. Available: http://www.xenomai.org/

[7] L. Palopoli, T. Cucinotta, L. Marzario, and G. Lipari, "AQuoSA—Adaptive quality of service architecture," *Software: Practice and Experience*, vol. 39, no. 1, pp. 1–31, 2009.

[8] A. Garg, "Real-time Linux kernel scheduler," *Linux J.*, no. 184, pp. 54–60, Aug. 2009.

[9] *Industrial Communication Networks—Fieldbus Specifications—Part 3–12: Data-Link Layer Service Definition—Part 4–12: Data-Link Layer Protocol Specification—Type 12 Elements*, IEC 61158-3/4-12, 1.0 ed., Dec. 2007.

[10] IgH, "EtherLab—EtherCAT master," 2010. [Online]. Available: http://www.etherlab.org/

[11] G. Boggia, P. Camarda, L. Grieco, and G. Zacheo, "An experimental evaluation on using TDMA over 802.11 MAC for wireless networked control systems," in *Proc. 13th IEEE Int. Conf. Emerging Technol. Factory Autom.*, Sep. 2008, pp. 1157–1160.

[12] C. Centioli, F. Iannone, G. Mazza, M. Panella, L. Pangione, V. Vitale, and L. Zaccarian, "Open source real-time operating systems for plasma control at FTU," *IEEE Trans. Nuclear Sci.*, vol. 51, no. 3, pp. 476–481, Jun. 2004.

[13] T. Cucinotta, A. Mancina, G. F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusinà, "A real-time service-oriented architecture for industrial automation," *IEEE Trans. Ind. Inform.*, vol. 5, no. 3, pp. 267–277, Aug. 2009.

[14] T. Cucinotta, L. Palopoli, L. Abeni, D. Faggioli, and G. Lipari, "On the integration of application level and resource level QoS control for real-time applications," *IEEE Trans. Ind. Inform.*, vol. 6, no. 4, pp. 479–491, Nov. 2010.

[15] G.-H. Jung and S.-J. Kang, "Active isochronous resource manager for intense dynamic resource allocation service with IEEE 1394," *IEEE Trans. Consumer Electron.*, vol. 51, no. 2, pp. 501–506, May 2005.

[16] L. Abeni, L. Palopoli, C. Scordino, and G. Lipari, "Resource reservations for general purpose applications," *IEEE Trans. Ind. Inform.*, vol. 5, no. 1, pp. 12–21, Feb. 2009.

[17] L. Palopoli and L. Abeni, "Legacy real-time applications in a reservation-based system," *IEEE Trans. Ind. Inform.*, vol. 5, no. 3, pp. 220–228, Aug. 2009.

[18] A. Rowe, K. Lakshmanan, H. Zhu, and R. Rajkumar, "Rate-harmonized scheduling and its applicability to energy management," *IEEE Trans. Ind. Inform.*, vol. 6, no. 3, pp. 265–275, Aug. 2010.

[19] 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 Trans. Nuclear Sci.*, vol. 55, no. 1, pp. 435–439, Feb. 2008.

[20] M. Liu, D. Liu, Y. Wang, M. Wang, and Z. Shao, "On improving real-time interrupt latencies of hybrid operating systems with two-level hardware interrupts," *IEEE Trans. Comput.*, vol. 60, no. 7, pp. 978–991, Jun. 2010.

[21] S. Rostedt and D. V. Hart, "Internals of the RT Patch," in *Proc. Ottawa Linux Symp.*, Ottawa, Canada, Jun. 2007, vol. 2, pp. 161–172.

[22] W. Betz, M. Cereia, and I. Cibrario Bertolotti, "Experimental evaluation of the Linux RT Patch for real-time applications," in *Proc. 14th IEEE Conf. Emerging Technol. Factory Autom.*, Sep. 2009, pp. 1–8.

[23] A. Siro, C. Emde, and N. McGuire, "Assessment of the real time preemption patches (RT-Preempt) and their impact on the general purpose performance of the system," in *Proc. 9th Real-Time Linux Workshops*, 2007, pp. 1–9.

[24] M. Cereia, I. Cibrario Bertolotti, and S. Scanzio, "Performance evaluation of an EtherCAT master using Linux and the RT Patch," in *Proc. IEEE Int. Symp. Ind. Electron.*, Jul. 2010, pp. 1748–1753.

[25] M. Felser, "Real time Ethernet: Standardization and implementations," in *Proc. IEEE Int. Symp. Ind. Electron.*, 2010, pp. 3766–3771.

[26] S. Fischmeister, R. Trausmuth, and I. Lee, "Hardware acceleration for conditional state-based communication scheduling on real-time Ethernet," *IEEE Trans. Ind. Inform.*, vol. 5, no. 3, pp. 325–337, Aug. 2009.

[27] P. Ferrari, A. Flammini, D. Marioli, and A. Taroni, "A distributed instrument for performance analysis of real-time ethernet networks," *IEEE Trans. Ind. Inform.*, vol. 4, no. 1, pp. 16–25, Feb. 2008.

[28] "EL2202 2-Channel Digital Output Terminal," Beckhoff Automation GmbH, 2011. [Online]. Available: http://www.beckhoff.com/

[29] 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 Technol. Appl. Symp. (RTTAS)*, 2002, pp. 133–142.

[30] K. Yaghmour, "Adaptive domain environment for operating systems," 2001. [Online]. Available: http://www.opersys.com/ftp/pub/Adeos/adeos.pdf

[31] H. Kasim, V. March, R. Zhang, and S. See, "Survey on parallel programming model," in *Proc. IFIP Int. Conf. Network and Parallel Computing*, 2008, vol. 5245, pp. 266–275.

[32] L. Sha, R. Rajkumar, and S. S. Sathaye, "Generalized rate-monotonic scheduling theory: A framework for developing real-time systems," *Proc. IEEE*, vol. 82, no. 1, pp. 68–82, Jan. 1994.

Marco Cereia graduated with a Degree in electronic engineering from the Politecnico di Torino, Torino, Italy, in 2001.

In 2001, he was with the Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni of the National Research Council of Italy (IEIIT-CNR), Torino, Italy, and since 2003, he has been a Research Fellow with the same institute. His research interests are in the area of real-time operating systems, embedded devices and industrial communication systems and protocols.

Ivan Cibrario Bertolotti (M'06) received the Laurea degree (*summa cum laude*) in computer science from the University of Torino, Torino, Italy, in 1996.

Since then, he has been a Researcher with the National Research Council of Italy (CNR). Currently, he is with the Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT), Torino. He teaches several courses on real-time operating systems at Politecnico di Torino, Torino, and has served as a technical referee for several international conferences and journals. His current research interests include real-time operating system design and implementation, industrial communication systems and protocols, and formal methods for vulnerability and dependability analysis of distributed systems.

Stefano Scanzio was born in Italy in 1979. He received the Laurea and Ph.D.degrees in computer science engineering from the Politecnico di Torino, Turin, Italy, in 2004 and 2008, respectively.

From 2004 to 2009, he was with the Department of Computer Engineering, Politecnico di Torino, where he was engaged in research on speech recognition and, in particular, he has been active in classification methods and algorithms. Since 2009, he worked with the National Research Council of Italy (CNR) and, currently, he is with the Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT), Turin. His current research interests include communication protocols, industrial communication systems, real-time networks, and real-time operating systems. He coauthored a number of technical papers in the area of computer science.