Investigating latency effects of the Linux real-time Preemption Patches (PREEMPT_RT) on AMD's GEODE LX Platform

Kushal Koolwal
VersaLogic Corporation
3888 Stewart Road, Eugene, OR 97402 USA
kushalk@versalogic.com

Abstract

When it comes to embedded systems, real-time characteristics like low-latency, deterministic behavior and guaranteed calculations are of utmost importance and there has been an increasing market demand for real-time features on embedded computers.

This paper presents results of benchmarking a standard Linux kernel against a real-time Linux kernel (with PREEMPT_RT patch) using the Debian Linux operating system on AMD Geode LX platform board. As the PREEMPT_RT patch (RT patch) matures further and integrates into the mainline Linux kernel, we try to characterize the latency effects (average and worst-case) of this patch on LX-based platform.

The paper starts with a basic introduction of the RT patch and outlines the methodology and environment used for evaluation. Following that, we present our results with appropriate graphs (bar graphs, histograms and scatter plots) and discuss those results. We also look for any performance degradation due to the real-time patch.

The paper concludes with some future work that can be done to further improve our results and discusses some important issues that need to be considered when using the PREEMPT_RT patch.

1 Introduction

Linux has been regarded as an excellent General Purpose Operating System (GPOS) over the past decade. However, recently many projects have been started to modify the Linux kernel to transform it into a Real-time Operating System (RTOS) as well. One such project is PREEMPT_RT patch[1] (also known as RT patch) led by Ingo Molnar and his team. The goal of this patch is to make the Linux kernel more deterministic and reduce the average latency of the Linux operating system.

This paper builds upon "Myths & Realities of Real-Time Linux Software Systems" paper[2]. For basic real-time concepts (in Linux) we strongly recommend you to first read the FAQ paper before further reading this paper.

In this paper we are going to investigate the latency effects of the PREEMPT_RT patch on an AMD Geode LX800 board. There have been many real-time benchmarking studies (using RT patch) based on Intel, IBM, and ARM platforms such as Pentium, Xeon, Opteron, OMAP, etc. [1,15], but no benchmarking has been done (as of this writing) with PREEMPT_RT patches on AMD's Geode LX platform. Moreover, the Geode platform has certain kind of "virtual" hardware built into it and it would be interesting to find out how does that affect the real-time latencies. The aim of this paper is to assess the RT patch using the 2.6.26 kernel series and discuss its effects on the latency of the Linux OS.

2 Test Environment

2.1 System details

Board Name	EBX-11
Board Revision	5.03
CPU (Processor)	AMD Geode LX800 (500 MHz)
Memory	Swiss-bit PC2700 512 MB
Storage Media	WDC WD200EB-00CPF0 (20.0 GB Hard Drive)
BIOS Version	General Software 5.3.102
Periodic SMI	Disabled
USB2.0/Legacy	Disabled

2.2 Operating System Details

OS Name	Debian 5.0 (Lenny/testing – i386) - Fresh Install
Linux Kernel Version	2.6.26 (without and with RT patch)
RT Patch Version	Patch-2.6.26-rt1
Boot mode	multi-user (Runlevel 2) with "quiet" parameter
Swap space	80MB (/dev/hda5)
ACPI	Off/Minimal

2.3 BIOS/System Settings to reduce large latencies

Following are some System/BIOS settings[1] that are known to induce large latencies (in 100's of msecs) which we need to take care of: - One of the things that make an OS a "good" RTOS is low latency interrupt handling. SMI (System Management Interrupts) interrupts are known to cause large latencies (in several 100's of μs). Therefore we disable the "Periodic SMM IRQ" option[2] in the BIOS[3].

- Enable minimal ACPI functionality. Just enable the ACPI support option in kernel and uncheck/disable all the sub-modules. Features like on-demand CPU scaling can cause high system latencies[4]. Since 2.6.18-rt6 patch we need the ACPI support to activate "pm_timer" since TSC timer is not suitable for high-resolution timer support.

- All tests were run through an SSH connection. It is not recommended to run the tests on a console as the printk (kernel's print command) can induce very high latencies[3].

3 Test Methodology

3.1 Test Selection

Based on our research there is no one "single" test that would test all the improvements/features of the PREEMPT_RT patch. Therefore, we selected various kinds of test for benchmarking, in order to cover all different metrics of real-time measurements such as interrupt latency, scheduling latency, worst-case latency, etc. A table comparing different tests that we have used follows.

3.2 Test Runs

All the tests were executed with (worst-case) and without (normal) "system load". By system load, we mean a program/script which generates sufficient amount of CPU activity and IO operations (example: reading/writing to disks) to keep the system busy 100% of the time. We wrote a simple shell script to generate this kind of load[3]. Please see **Appendix I** for the script code.

Normal (without load) run: Here we simply ran the tests without any explicit program (script) generating system load. This is equivalent of running your real-time application under normal circumstances that is an idle system with no explicit programs running.

Worst-case[4] (with load) run: Here we ran the tests under a heavy system load. This is equivalent of running your real-time application under system load to determine the maximum time your application would take to complete the desired operation in case an unexpected event occurs.

[1]For more details about these issues please see: http://rt.wiki.kernel.org/index.php/HOWTO:_Build_an_RT-application#Latencies

[2]USB Legacy devices are known to cause large latencies so generally it is a good idea to disable the 'USB legacy option' (if it exits) in the BIOS and to also use PS/2 mouse and keyboard instead of USB. However, disabling the 'SMM IRQ option' in the BIOS takes care of this issue.

[3]Ideally we would generate the worst-case system load that our application might encounter. Since we are benchmarking in general, we generate an extremely high load which a real-time application is unlikely to encounter. Therefore in a real scenario we might (or might not) get slightly better real-time performance.

[4]Worst-case means system running under load

Extended Worst-case (with load) run: In addition to running each of the above mentioned tests, under a Normal and Worst-case scenario for approximately 1 minute, we ran the tests for 24-hours also under a system load. Here, we do not show results of extended tests without any system load because the results were not significantly different from what we observed for running the test for 1 min without any system load (Normal Scenario). To get realistic and reliable results, especially for worst-case latencies, we need to run tests for many hours, preferably at least for 24 hours so that we have at least million readings (if possible) [5,6,11]. Running tests over short durations (for example 1 minute) may fail to reveal all the different paths of code that the kernel might take.

A table comparing the different real-time benchmarks, their features, and their basic principles of operation can be found in **Appendix II**.

3.3 Kernels Tested

For benchmarking purposes we tested four different kernels:

a) **2.6.26-1-486**: At the time we conducted the tests, this was the default Debian kernel that came with Debian Lenny (5.0)[7]. This kernel has no major real-time characteristics.

b) **2.6.26-vl-custom-ebx11**: This is a custom configured Debian Linux kernel that is derived from above kernel. We configure/optimize the kernel so that it runs efficiently on EBX-11 board. This kernel is partly real-time – the option "Preemptible Kernel" (CONFIG_PREEMPT) is selected under kernel configuration menu.

c) **2.6.26-1-rt1-ebx11**: We applied the PREEMPT_RT patch to the above kernel in order to make the Linux kernel completely real-time by selecting the option "Complete Preemption" (CONFIG_PREEMPT_RT).

d) **2.6.18-4-486**: Since kernel 2.6.18, some parts of the PREEMPT_RT patch have been incorporated into main-line kernel. We used this default Debian kernel then (April 2007) to see how the latency, in general, of a default Linux kernel has improved in newer releases like 2.6.26. For example, theoretically we should see better results for 2.6.26-1-486 compared to 2.6.18-4-486.

4 Test Results

4.1 GTOD Test Results[5]

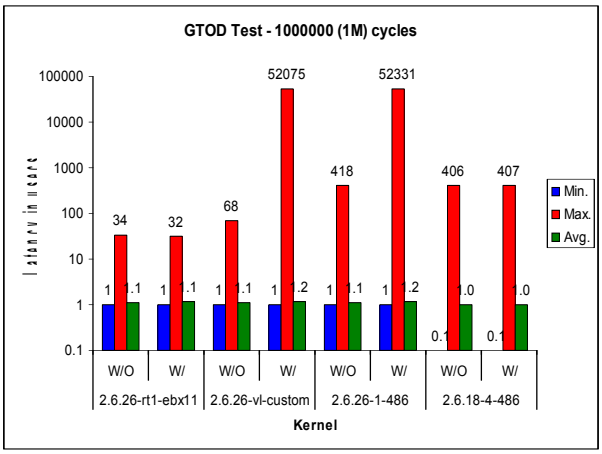


FIGURE 1: *GTOD Results*

For further details on the test results please refer to **Appendix III (a)**.

Scatter plots for GTOD Test under system load[6],[7]

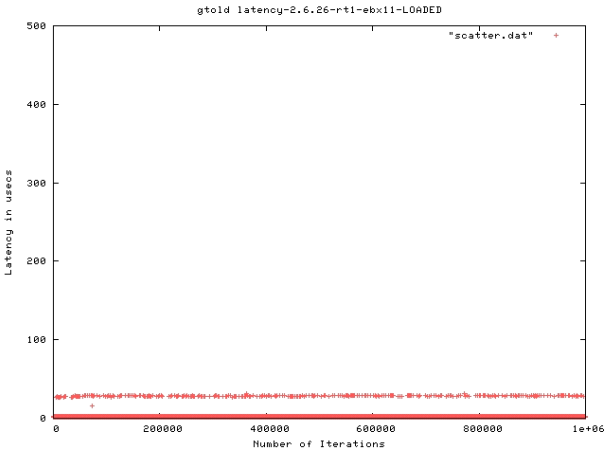


FIGURE 2: *2.6.26-rt1 LOADED*

[5]1 second = 1000 milliseconds = 1000000 μs (secs)
W/O = Without system load
W/ = With system load
1M = 1 Million

[6]Wherever required, we have kept the scale of X and Y axes constant across all the graphs (histogram and scatter plot) of each test by converting them into logarithmic scale.

[7]The suffix "LOADED" at the top section of each graph, as in 2.6.26-rt1-ebx11-LOADED, means system was under load.

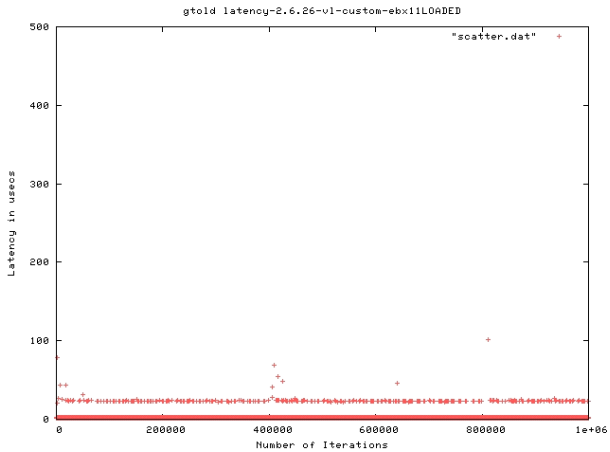


FIGURE 3: *2.6.26-custom LOADED*

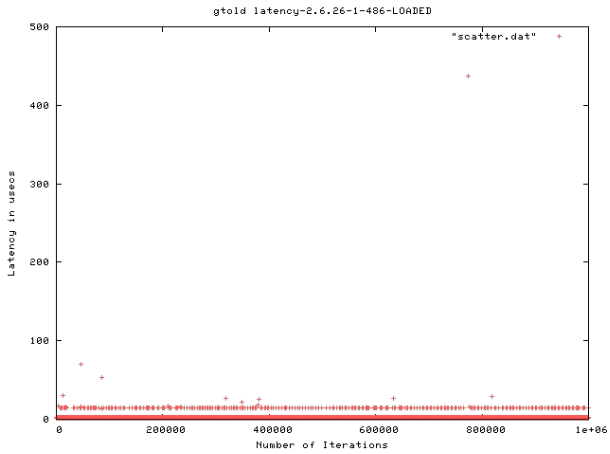


FIGURE 4: *2.6.26-1-486 LOADED*

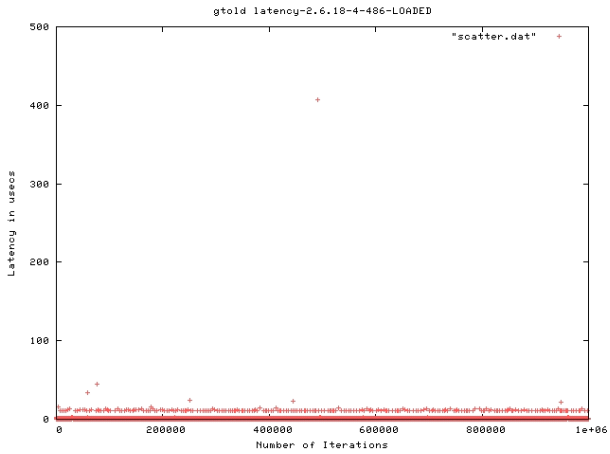


FIGURE 5: *2.6.18-1-486 LOADED*

GTOD TEST: As we can see from figure 1 with the RT kernel version (2.6.26-rt1-ebx11), the maximum (red bar) latency is significantly reduced to 32 μs even under system load. If we were to use a default Debian kernel (2.6.26-1-486), we would see max. latencies on the order of 52331 μs (0.05 secs). Also even though the avg. latencies (green bar) is quite similar across all the kernels (around 1.1 μs), we see significant differences with regards to max. latencies. When we talk about "hard"[8] real-time systems we are more concerned with max. latency rather than avg. latency.

Also the 2.6.18 kernel performed better than non-RT 2.6.26 kernels. Usually we would expect the opposite of this - a recent kernel version should perform better than the older version.

4.2 CYCLICTEST Results

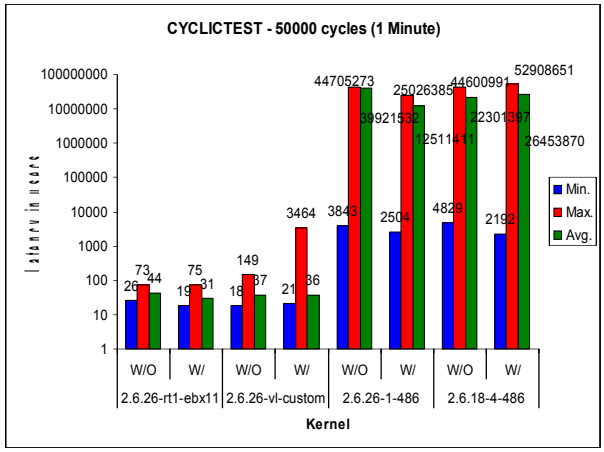


FIGURE 6: *CYCLICTEST Results (1 min)*

For further details on the test results please refer to **Appendix III (b)**.

[8]For information about the difference between "hard" and "soft" real-time systems, please refer to the section "Hard vs Soft Real-time" in[2].

Histograms for CYCLICTEST (1 min) under system load[9]

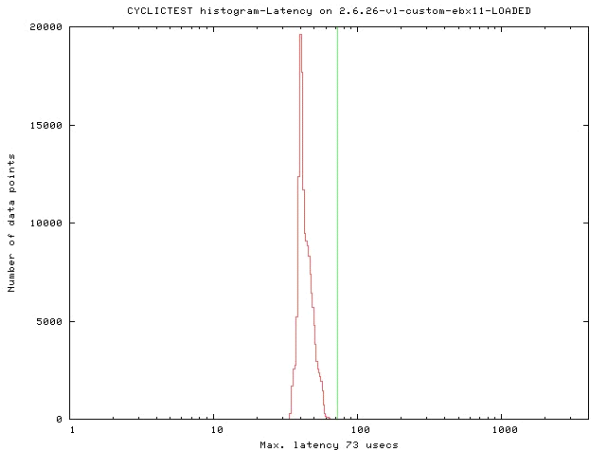


FIGURE 7: *2.6.26-rt1 LOADED*

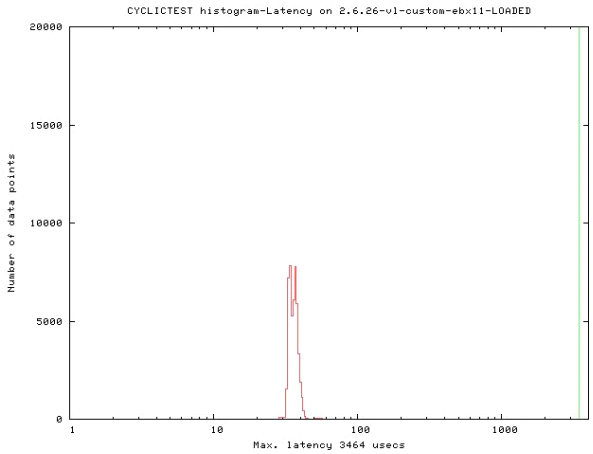
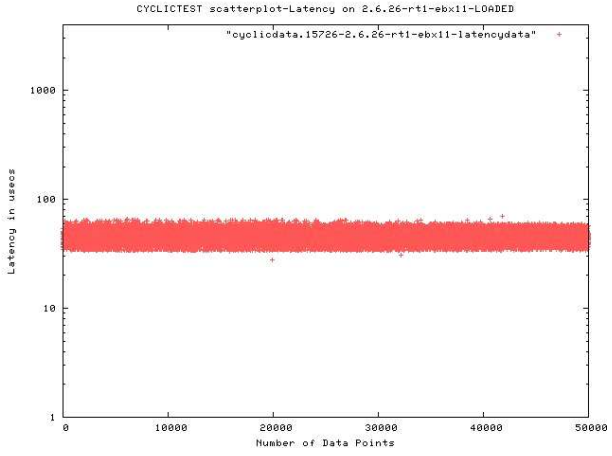


FIGURE 8: *2.6.26-custom LOADED*

Scatter plots for CYCLICTEST (1 min) under system load



[9]The green line in histograms indicates the max. latency point

FIGURE 9: *2.6.26-rt1 LOADED*

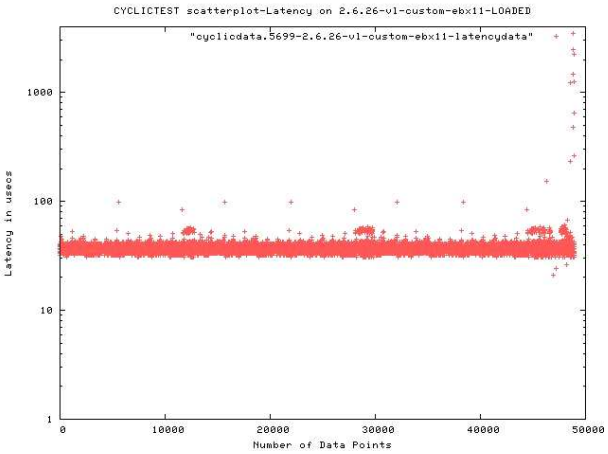


FIGURE 10: *2.6.26-custom LOADED*

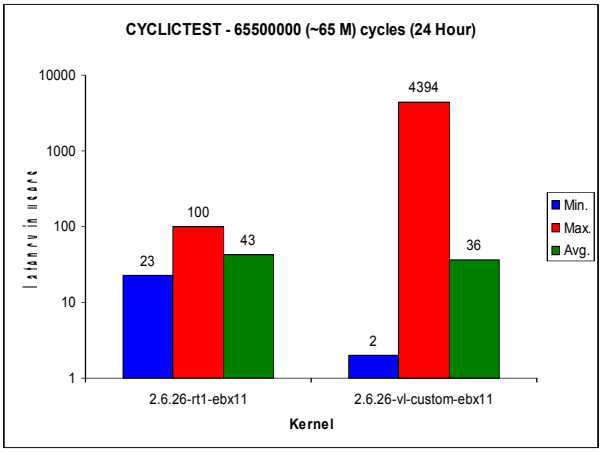


FIGURE 11: *CYCLICTEST Results (24 hr)*

For further details on the test results please refer to **Appendix III (c)**.

Histograms for CYCLICTEST (24 hour) under system load

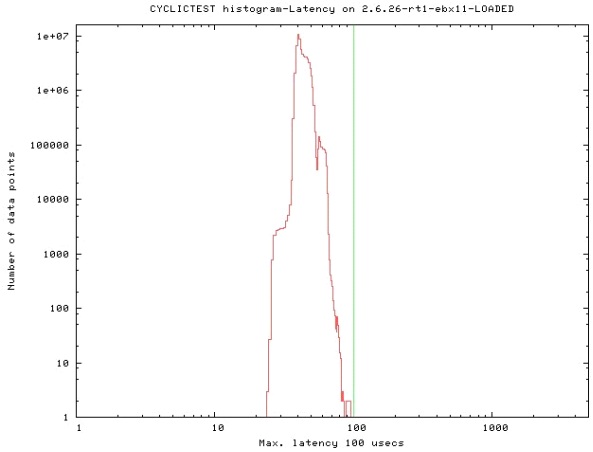


FIGURE 12: *2.6.26-rt1 LOADED*

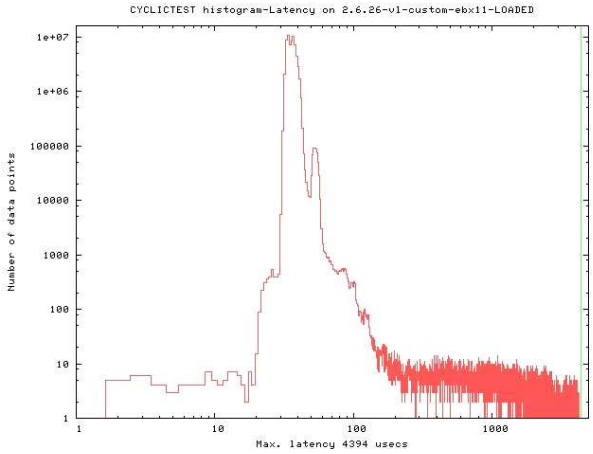


FIGURE 13: *2.6.26-custom LOADED*

Scatter plot for CYCLICTEST (24 hour) under system load

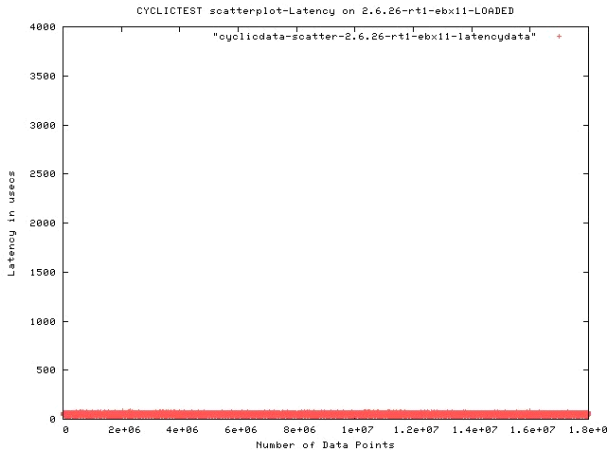


FIGURE 14: *2.6.26-rt1 LOADED*

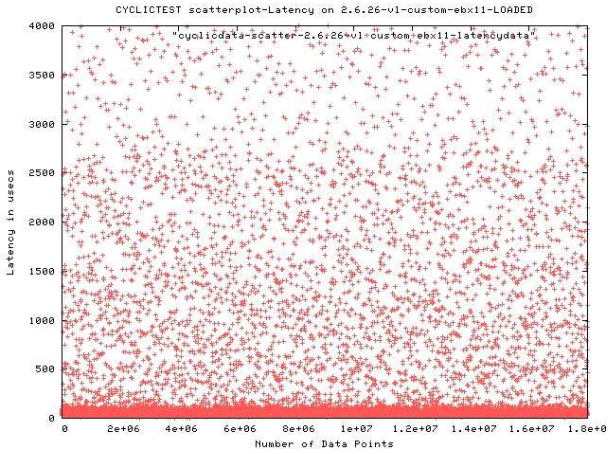


FIGURE 15: *2.6.26-custom LOADED*

CYCLICTEST: From figure 6, we can clearly see that the max. latency for RT kernel has significantly reduced to around 75 μs from 25026385 μs (25 secs). Also overall the avg. latency for RT kernel has also reduced. Also from figure 11, we can see that the max. latency for RT kernel increased to 100 μs (24 hour test) from 75 μs (1 min. test) but it is still less than max. latency of 4394 μs of the corresponding non-RT kernel (2.6.26-vl-custom-ebx11) under system load. This observation is consistent with the extended tests above - running tests for longer duration possibly makes the kernel to go to different code (or error) paths and hence we would expect increase in latencies.

Also from the above scatter plot (24 hour) figures 14,15 for cyclictest, we can see that red dots are distributed all over the graph (bottom right) for non-RT kernel (2.6.26-vl-custom-ebx11) in contrast to RT kernel (bottom left) indicating that there are lots of instances in which latencies have shot well above 100 μs mark which is the max. latency of RT kernel (2.6.26-rt1-ebx11). One can see the nature of distribution of these latencies in the histogram plot (24 hour) also.

Furthermore, from figure 11 we can see that avg. latency of RT kernel (23 μs) is more than that of non-RT kernel (2 μs). This is quite surprising but instances like these are not uncommon[8] for people who have performed similar tests. Morever, from a practical approach, we care more about maximum latencies rather than average latencies.

4.3 LPPTest Results

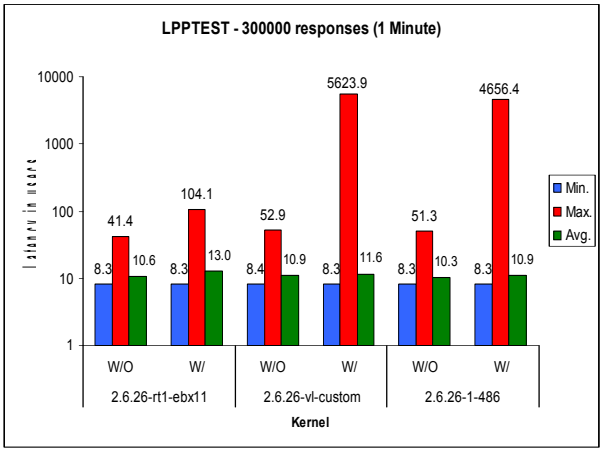


FIGURE 16: *LPPTest Results (1 min)*

For further details on the test results please refer to **Appendix III (d)**[10].

Histograms for LPPTEST (1 min) under system load

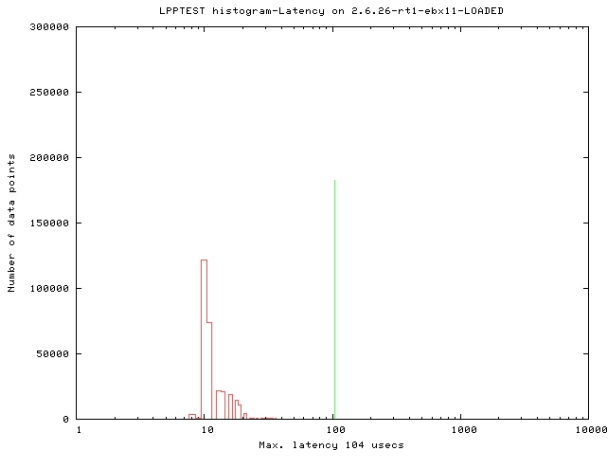


FIGURE 17: *2.6.26-rt1 LOADED*

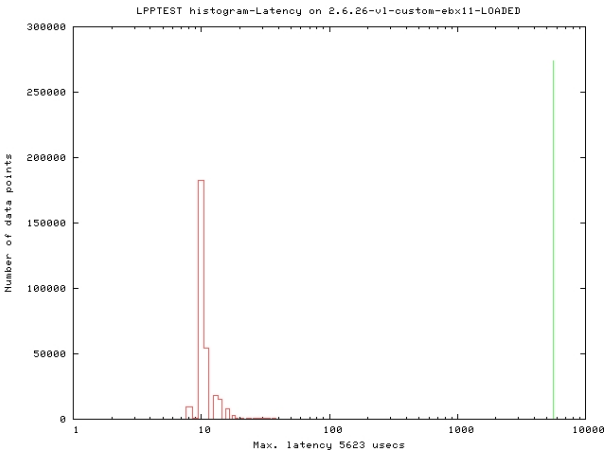


FIGURE 18: *2.6.26-custom LOADED*

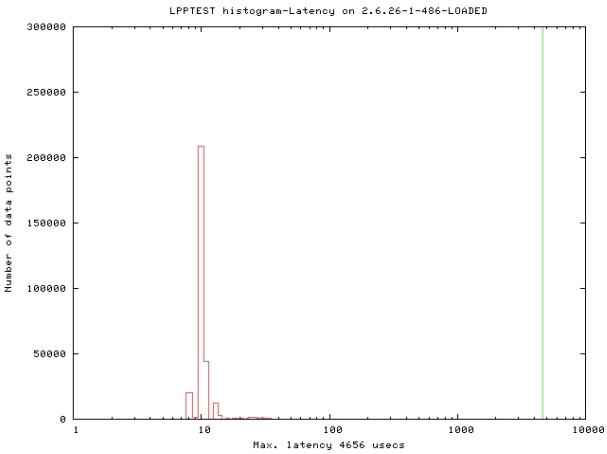
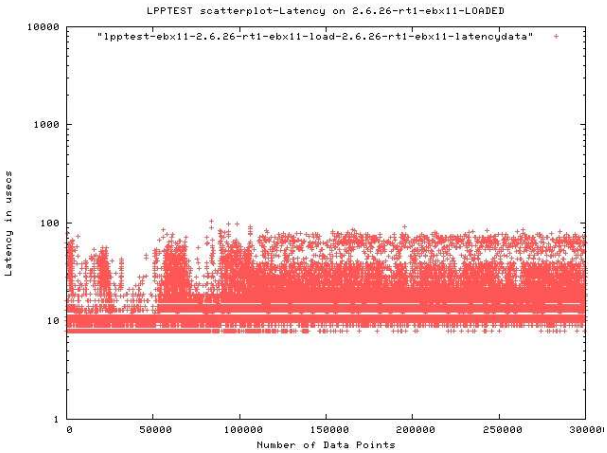


FIGURE 19: *2.6.26-1-486 LOADED*

Scatter Plot for LPPTEST (1 min) under system load



[10]We were unable to test the 2.6.18-4-486 version under lpptest because of the difficulty in porting the lpptest program to 2.6.18 from 2.6.26 due to some major changes to the kernel code structure.

FIGURE 20: *2.6.26-rt1 LOADED*

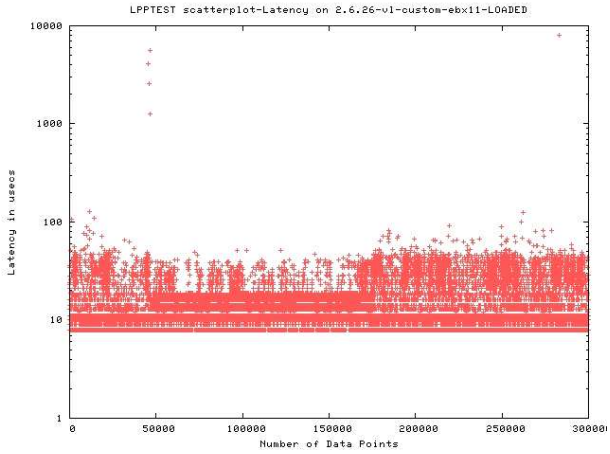


FIGURE 21: *2.6.26-custom LOADED*

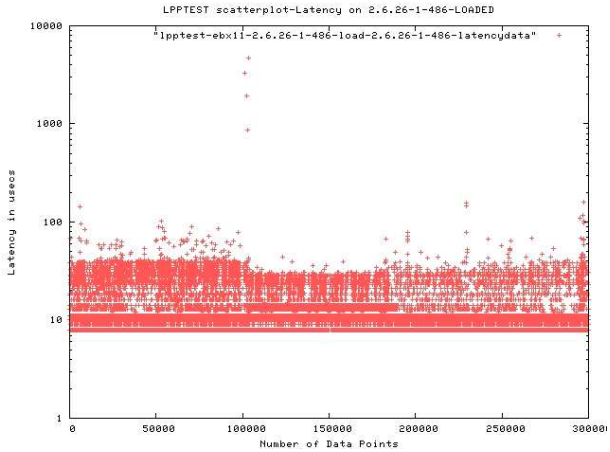


FIGURE 22: *2.6.26-1-486 LOADED*

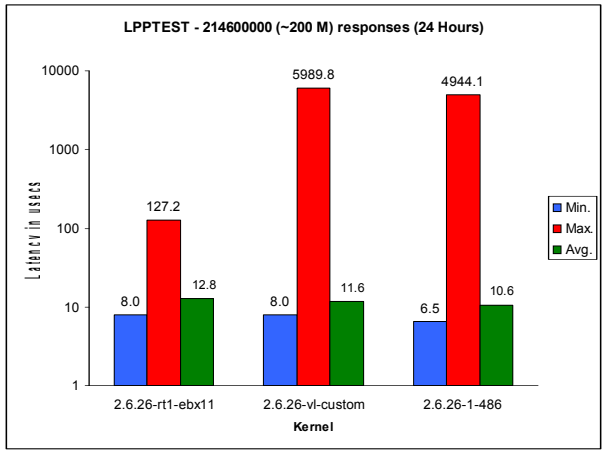


FIGURE 23: *LPPTest Results (24 hr)*

For further details on the test results please refer to **Appendix III (e)**.

Histograms for LPPTEST (24 hour) under system load

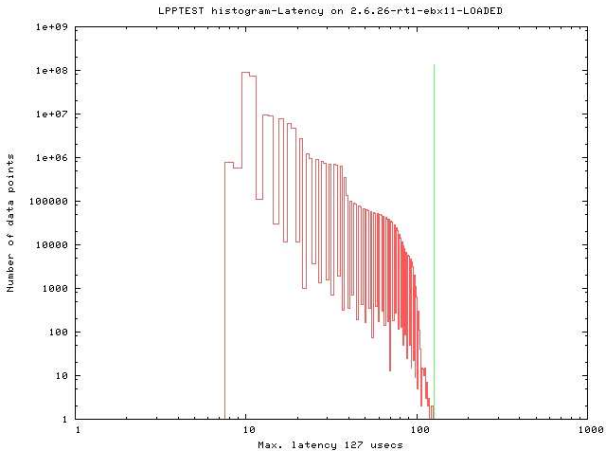


FIGURE 24: *2.6.26-rt1 LOADED*

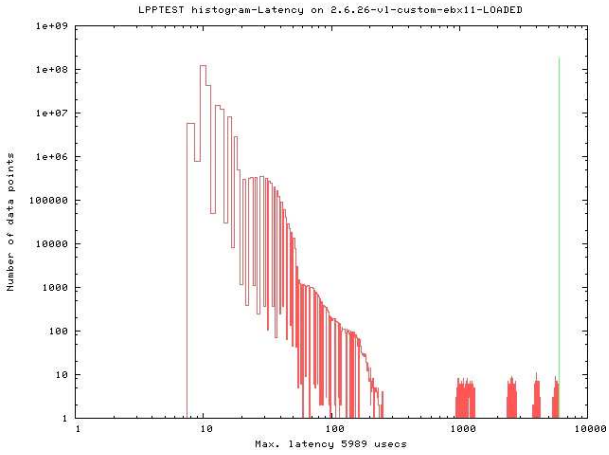


FIGURE 25: *2.6.26-custom LOADED*

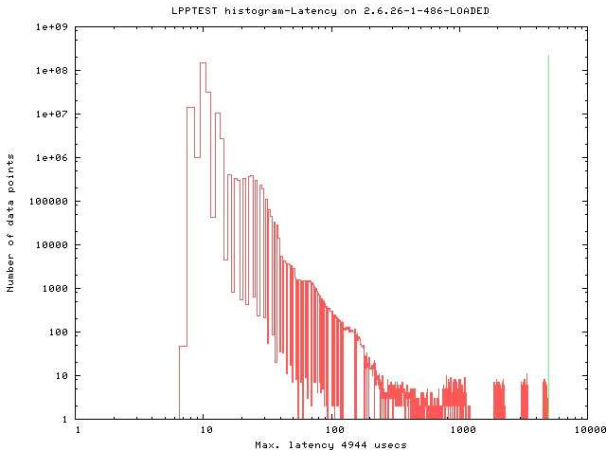


FIGURE 26: *2.6.26-1-486 LOADED*

Scatter Plot for LPPTEST (24 hour) under system load

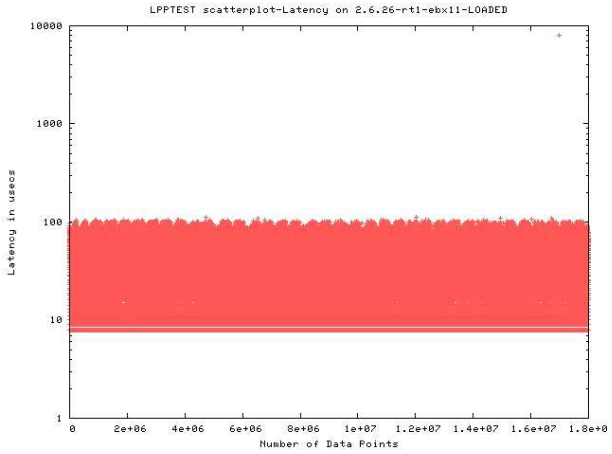


FIGURE 27: *2.6.26-rt1 LOADED*

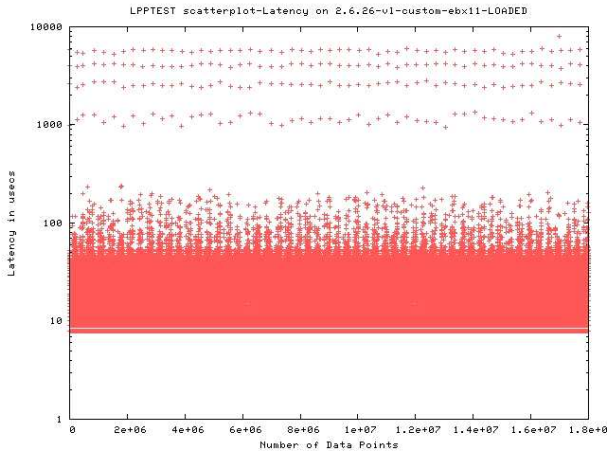


FIGURE 28: *2.6.26-custom LOADED*

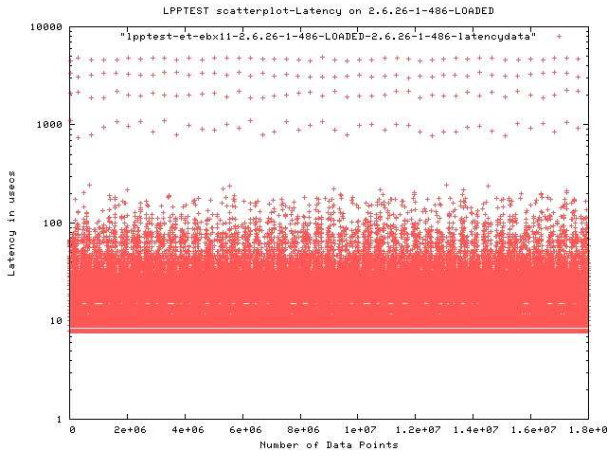


FIGURE 29: *2.6.26-1-486 LOADED*

LPPTEST: The results of lpptest, 1 min. and 24 hour, are consistent with results of the above two tests - max. latencies getting significantly reduced in the RT kernel as compared to non-RT kernels. See figure 4 and 5. Histograms and scatter plots (24 hour) of lpptest test show many latencies above the 127.2 μs mark, the max. latency for the RT kernel.

4.4 Latency comparison of 2.6.26-rt1-ebx11 across all the tests

Here we compare the latencies of the RT kernel (2.6.26-rt1-ebx11) across all the tests.

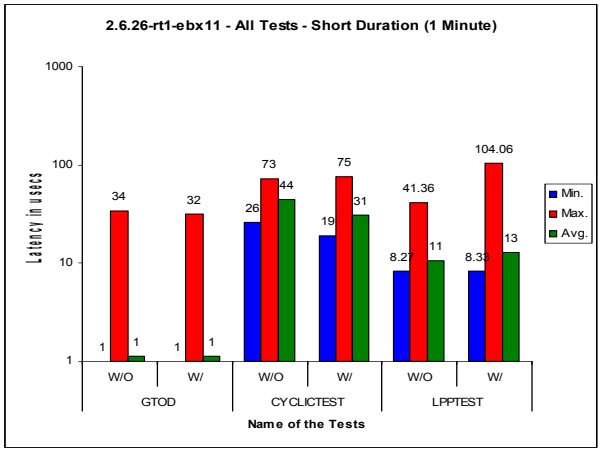


FIGURE 30: *Latency Results (1 min)*

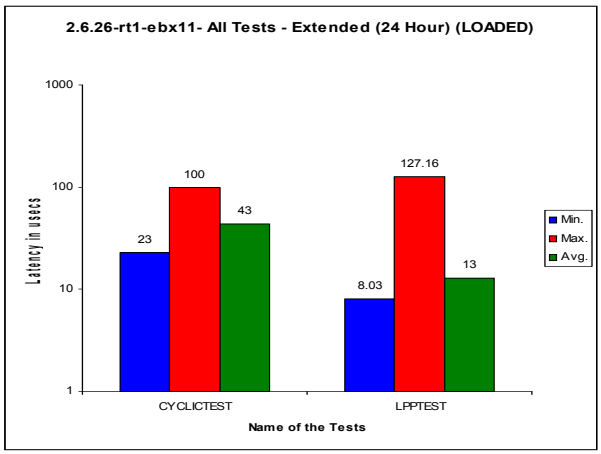


FIGURE 31: *Latency Results (24 hr)*

From the figures 30 and 31, we can see that latencies for each of the extended tests are more than their corresponding short duration tests. Also the latencies for LPPTEST are greater than latencies of other

tests because of the external measurement method[11] involved.

Do these numbers make sense?

Now having summarized the results, what can we conclude from these numbers? Are these the right kind of numbers that we should have been expecting? Again, this is an open-ended question and the answer really depends upon your needs, the underlying application and the system[2]. However we can use the following guideline from OSADL[9] to give us some idea whether our numbers are in the ball park range: "As a rule of thumb, the maximum (worst-case) latency amounts to approximately (105 / clock frequency of an idle system)." Upon calculating based on the above guideline for the EBX-11's 500 MHz LX processor, the maximum allowed latency comes to around 200 μs[12]. From figures 30 and 31, we can clearly see that during any of the tests, the max. latency has never exceeded more than 127.2 μs. Therefore, based on the above guideline, the results look well under control, but then again, it really depends upon how much max. latency your application/system can tolerate.

4.5 Impact on performance

As we know[2], real-time improvements come at the cost of possible performance degradation[13]. An RTOS may (or may not) sacrifice performance in order to become more deterministic. Therefore we decided to benchmark performance of 2.6.26 kernels to see how PREEMPT_RT patch affects the performance (or throughput) in general.

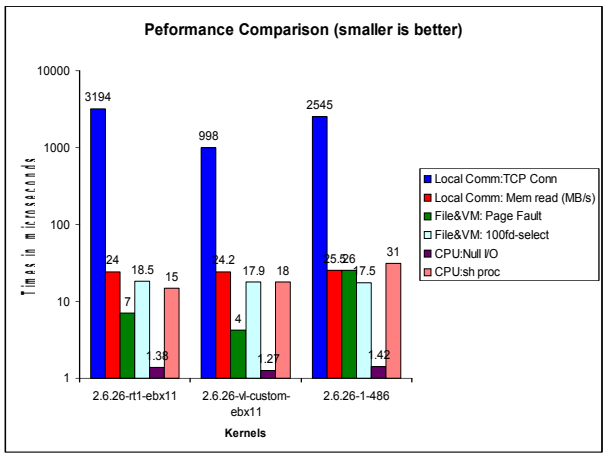


FIGURE 32: *Performance Comparison*

We arbitrarily chose few parameters to compare from various system categories such as Communication, File and Virtual Memory and CPU from the LMbench test suite[10], as shown in figure 32 above. From the above figure[14], we can say the RT patch does affect the system performance negatively across most of the categories as compared to non-RT kernels, especially with TCP Conn., where real-time kernel (2.6.26-rt1-ebx11) takes 3194 μs (tallest blue bar) and non real-time kernel (2.6.26-vl-custom-ebx11) takes only 998 μs. In general, we can say that the negative impact of RT patch is very application dependent, although for majority of the parameters, the RT patch does not affect the system performance negatively by a "significant" margin. Please see **Appendix IV** for other system parameter results.

5 Future work and considerations

Although the above results give us a good idea about the usefulness and effectiveness of PREEMPT_RT patch, there are some additional factors to be considered:

a) BIOS/Firmware impact Real-time metrics can depend upon how well BIOS is configured and how well the BIOS code is written. Settings related to USB device configuration, SMI interrupts, System Management mode, etc. can make a huge difference on the latencies of a given system.

b) Kernel settings The above results could vary depending upon how the Linux kernel is configured. We carefully configured all relevant real-time kernel parameters to take maximum advantage of the PREEMPT_RT patch. However there is a possibility that our current kernel configuration can be tweaked to further improve the determinism and the worst-case latencies.

c) Trace large latencies To further improve deterministic behavior we can trace which kernel functions/threads/programs are responsible for causing large system latencies (in 100s of μs). This can be done by enabling some of the debugging options in kernel[11,12]. We did not enable these debugging

[11] We measured the latencies of the target board (EBX-11) from a host board (another EBX-11 board) using parallel port cable. For all other tests (gtod and cyclictest), the latencies are recorded on the board that is being benchmarked itself.

[12] 100000/500000000=0.0002 secs = 200 μs. The calculation was done assuming 1 GHz = 1 Billion cycles per second.

[13] Note that real-time does not mean better performance. For details refer to[2].

[14] For all parameters in the figure 32, smaller value is better except for Memread (red bar). In order to avoid creating a separate graph just for one parameter (Memread), we decided to add it with other parameters. Note that the unit of Memread is in MB/s and hence bigger value is better for Memread. For rest of the other parameters the unit is secs.

options during our tests because doing so increases the system overhead and causes additional latencies to be introduced by the kernel itself, which would skew our results.

d) Different RT approach Lastly, as mentioned in the beginning of the paper, the PREEMPT_RT is just one of the several approaches to make Linux kernel real-time. It would be interesting to see if we can improve our current results by using some of the other real-time approaches like Xenomai, RTLinux, etc.

6 Conclusion

PREEMPT_RT patch significantly improves the premptiveness of the Linux kernel. However we should keep in mind that these results are statistical in nature rather than being conclusive. Developers should consider the latency requirement of their application rather than relying on these latency numbers. For example, an embedded application developer who defines a worst-case latency requirement to be 200 μs can probably use the above combination of EBX-11 and the RT kernel (2.6.26-rt1-ebx11). In general there is no such "magic" latency figure that one can simply assume.

There is still some debate in the community concerning the readiness of the PREEMPT_RT patch for "hard" real-time systems [14,17]. Therefore, we cannot yet draw broad conclusions about the suitability of the PREEMPT_RT patch for mission critical and life sustaining systems, where missing even a single deadline can lead to catastrophic failures and loss of life.

References

[1] *http://www.kernel.org/pub/linux/kernel/projects/rt/older/patch-2.6.26-rt1.bz2*

[2] *http://versalogic.com/mediacenter/whitepapers/wp_linux_rt.asp*

[3] *http://rt.wiki.kernel.org/index.php/HOWTO:_Build_an_RT-application#Latencies*

[4] *http://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO#Configuration_and_Compilation*

[5] *http://lkml.org/lkml/2005/6/30/9*

[6] *http://lkml.org/lkml/2005/6/22/337*

[7] *http://www.debian.org/CD/http-ftp/*

[8] *http://kerneltrap.org/node/5466*

[9] *http://www.osadl.org/Realtime-Preempt-Kernel.kernel-rt.0.html*

[10] *http://www.bitmover.com/lmbench/get_lmbench.html*

[11] *Read explanation of debugging options under "Kernel Hacking -¿ Kernel Debugging" option in the kernel configuration menu after applying the 2.6.26-rt1 RT patch from [1]*

[12] *http://www.linuxdevices.com/news/NS8896249579.html*

[13] *http://www.linuxdevices.com/articles/AT8906594941.html*

[14] *http://dslab.lzu.edu.cn:8080/docs/publications/Siro.pdf*

[15] *http://www.research.ibm.com/journal/sj/472/hart.html*

[16] *http://www.uwsg.iu.edu/hypermail/linux/kernel/0507.2/0468.html*

[17] *Building Embedded Linux Systems*, 2nd Edition, Karim Yaghmour, Jonathan Masters and Gilad Ben-Yossef, Aug. 2008,O'Reilly Media

APPENDIX I – System Load Script Code

```
#!/bin/bash
defaultseconds=60
seconds=“$1”
if test -z “$seconds” ||test “$seconds” -lt 1
then
seconds=$defaultseconds
fi
cd /usr/src/linux
make clean
make -j8 &
cd;
echo “Started make -j8”
(while true; do dd if=/dev/zero of=/dev/null bs=1024000 count=1000; done) &
echo “Started DD”
ping -l 100 -q -s 10 -f localhost &
echo “Started ping flood”
(while true; do nice ./hackbench 30 >/dev/null; done) &
echo “Started hackbench...”
(while true; do du / 1>/dev/null 2>&1; done) &
echo “Started DU...”
while test $seconds -gt 0
do
echo -n -e \\rStill $seconds seconds to wait ...
sleep 1
seconds=‘expr $seconds - 1‘
done
echo
echo “Script Terminated...”
```

APPENDIX II – Tests Comparison

Test Name	**GTOD_LATENCY**	**CYCLICTEST**	**LPPTEST**
Interrupt Generation	N/A	Clock clock_nanosleep	Data sent on PlaceNameplaceParallel PlaceTypePort
Measures?	Time between pairs of consecutive calls to determine time	Interrupt and Scheduling Latency	End-to-End Response Latency
CityplaceNormal Iterations (1 Min. approx.)	1000000	50000	300000
WC[1] Iterations (1 Min. approx.)	1000000	50000	300000
Extended Iteration (24 Hr. approx.)	N/A[2]	65500000 (65.5 M[3])	214600000 (214.6 M)
Driver/ Function	Gettimeofday()	N/A	lpptest
IRQ	N/A	N/A	7
Person/ Company	IBM	Thomas Gleixner TimeSys	Ingo Molar Red Hat
Comments	gtod = get_time_of_day. The test calls a function to get system's time of day.	Uses High Resolution timer support. Measures expected time vs. actual time for wakeup.	Built inside kernel. External measurement.

[1] WC = Worst-case scenario
[2] The test fails above 10000000 iterations and hence there are no worst-case results this test.
[3] M = Million

APPENDIX III – Test results

a) GetTimeOfDay - gtod_latency (usecs) – 1000000 Iterations

Kernels	**2.6.26-rt1-ebx11**		**2.6.26-vl-custom**		**2.6.26-1-486**		**2.6.18-4-486**	
Load?	**W/O**	**W/**	**W/O**	**W/**	**W/O**	**W/**	**W/O**	**W/**
Min.	1	1	1	1	1	1	N/A	N/A
Max.	34	32	68	52075	418	52331	406	407
Avg.	1.1268	1.1422	1.1101	1.1621	1.1311	1.1817	0.9869	0.9951
Std Dev	0.5845	0.5720	0.4786	52.0764	0.7188	52.3333	0.4590	0.4509

b) CyclicTest (Interrupt and scheduling latency) (usecs) – 50000 cycles (1 Minute)

Kernels	**2.6.26-rt1-ebx11**		**2.6.26-vl-custom**		**2.6.26-1-486**		**2.6.18-4-486**	
Load?	**W/O**	**W/**	**W/O**	**W/**	**W/O**	**W/**	**W/O**	**W/**
Min.	26	19	18	21	3843	2504	4829	2192
Max.	73	75	149	3464	44705273	25026385	44600991	52908651
Avg.	44	31	37.47	36.33	39921532	12511411	22301397	26453870

c) CyclicTest (Interrupt and scheduling latency) (usecs) – 18000000 cycles (24 hour)

Kernels	**2.6.26-rt1-ebx11**	**2.6.26-vl-custom**
Load?	Yes	Yes
Min.	23	2
Max.	100	4394
Avg.	43.29363198	36.5
Samples	65512035	65543590

d) LPP Test (End to End response latency) (usecs) – 300000 responses (1 Minute)

Kernels	**2.6.26-rt1-ebx11**		**2.6.26-vl-custom**		**2.6.26-1-486**		**2.6.18-4-486**	
Load?	**W/O**	**W/**	**W/O**	**W/**	**W/O**	**W/**	**W/O**	**W/**
Min.	8.27	8.33	8.36	8.33	8.33	8.33	N/A	N/A
Max.	41.36	104.06	52.91	5623.92	51.26	4656.44	N/A	N/A
Avg.	10.59	12.98	10.90	11.58	10.33	10.94	N/A	N/A

e) LPP Test (End to End response latency) (usecs) – 214600000 responses (24 hours)

Kernels	**2.6.26-rt1-ebx11**	**2.6.26-vl-custom**	**2.6.26-1-486**
Min.	8.03	7.95	6.45
Max.	127.16	5989.77	4944.08
Avg.	12.79	11.59	10.63
Responses	214620000	214669636	214619724

APPENDIX IV

L M B E N C H 3 . 0 S U M M A R Y
(Alpha software, do not distribute)

Basic system parameters

Host	OS	Description	Mhz	tlb pages	cache line bytes	mem par	scal load
ebx11	Linux 2.6.26-1-486		495		32		1
ebx11	Linux 2.6.26-v1-custom		494		32		1
ebx11	Linux 2.6.26-rt1-ebx11		493		32		1

Processor, Processes - times in microseconds - smaller is better

Host	OS	Mhz	null call	null I/O	stat	open clos	slct TCP	sig inst	sig hndl	fork proc	exec proc	sh proc
ebx11	Linux 2.6.26-	495	0.55	1.42	7.34	10.7	32.3	2.17	5.63	34.K	49.K	31.K
ebx11	Linux 2.6.26-	494	0.74	1.27	7.67	11.0	32.0	2.13	5.82	383.	4156	18.K
ebx11	Linux 2.6.26-	493	0.89	1.38	9.51	15.8	32.3	2.23	6.63	687.	2816	15.K

Context switching - times in microseconds - smaller is better

Host	OS	2p/0K ctxsw	2p/16K ctxsw	2p/64K ctxsw	8p/16K ctxsw	8p/64K ctxsw	16p/16K ctxsw	16p/64K ctxsw
ebx11	Linux 2.6.26-	65.9	221.5	723.0		6.3800		
ebx11	Linux 2.6.26-	230.6	233.5	797.8				

Local Communication latencies in microseconds - smaller is better

Host	OS	2p/0K ctxsw	Pipe	AF UNIX	UDP	RPC/ UDP	TCP	RPC/ TCP	TCP conn
ebx11	Linux 2.6.26-		36.4	35.1	110.8		378.7		2545
ebx11	Linux 2.6.26-	65.9	49.2	107.	133.3	345.8	367.1		998.
ebx11	Linux 2.6.26-	230.6	117.1	173.	167.4	925.8	347.0		3194

File & VM system latencies in microseconds - smaller is better

Host	OS	0K File Create	0K File Delete	10K File Create	10K File Delete	Mmap Latency	Prot Fault	Page Fault	100fd selct
ebx11	Linux 2.6.26-					71.3K	0.985	25.5	17.5
ebx11	Linux 2.6.26-					20.7K	0.765	4.27620	17.9
ebx11	Linux 2.6.26-					33.2K		7.13720	18.5

Local Communication bandwidths in MB/s - bigger is better

Host	OS	Pipe	AF UNIX	TCP	File reread	Mmap reread	Bcopy (libc)	Bcopy (hand)	Mem read	Mem write
ebx11	Linux 2.6.26-	17.6	19.5	15.3	13.1	24.5	20.7	20.0	25.5	82.8
ebx11	Linux 2.6.26-	15.9	16.8	11.9	12.4	24.4	19.8	19.1	24.2	83.9
ebx11	Linux 2.6.26-	15.1	16.2	1.46	9.2500	23.7	19.5	18.6	24.0	81.7