Abstract

In this paper, we address the latency issue in RT-XEN virtual machines that are available in Xen 4.5. Despite the advantages of applying virtualization to systems, the default credit scheduler is difficult to use with automotive devices and other systems that are developed to assist, complement, and eventually substitute the driver in the complex process of controlling a vehicle due to unpredictable domain scheduling and I/O latency.

This paper analyzes the latency of OS scheduling for symmetric and asymmetric multi-processing support cases – as well as incoming packet handling in Xen – using default credit and real-time schedulers. It also demonstrates how the real-time scheduler affects latency. With RT-Xen support, most of the incoming packets are predictably handled within 1 millisecond with a small overhead at the destined guest OS, which is a feasible time bound for most soft real-time applications.

Introduction

The speed of today's high-performance processors, combined with the real-time support for an embedded OS, appears to have re-opened the question of whether embedded systems support RTOS within a virtualization framework.

To accommodate a real-time guest OS to Xen, the I/O latency issue must be addressed. Predictable I/O latency is particularly critical because most real-time guest OSes demand timely handling for random I/O events. For example, a real-time guest OS should handle network interrupts within a predictable latency bound, regardless of the CPU workload or other general purpose guest OSes. However, the I/O latency over the Xen default scheduler is very unpredictable because the I/O latency is affected by virtual CPU (VCPU) scheduling within Xen.

I/O latency in Xen can be largely presented, although a guest OS will schedule an I/O task as soon as possible. At first, the credit scheduler (i.e., the default inter-VM scheduler for the Xen hypervisor) is insufficient for supporting time-sensitive applications. Specifically, with the split driver model of Xen, packet handling can be considerably delayed because the split driver model requires additional scheduling of the driver domain. Secondly, the latency inside a guest OS can be negatively affected by paravirtualization.

We made an evaluation to support the global scheduler on a SMP system with a global virtual CPU to physical core allocation. The purpose of this evaluation was to measure for different scheduler configurations and to find optimal values on an idle system and under system load.

Testing Parameters

Hardware Platform

We performed a real-time investigation on a DRA74x evaluation board. The hardware platform was equipped with a dual-core ARM Cortex-A15 processor and ran on a 1Ghz and 1Gbps network device.

Native System

The native system ran the SMP Linux kernel on the CPU. Linux has a complete scope of tools available on https://rt.wiki.kernel.org to measure latencies, as well as system calls that can be used for performance evaluations. The most important test is a cyclic test that determines system latencies, and it can be found on https://rt.wiki.kernel.org/index.php/Cyclictest.

We obtained reference values on the Texas Instruments Linux kernel 3.8.13 from the GLSDK. Below is the output from this test:

```
#./cyclictest -t8 -p 20 -n -i 10000
/dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 0.38 0.40 0.32 1/78 1736
T: 0 ( 1726) P: 1 I:10000 C: 144245 Min: 9 Act: 24 Avg: 20 Max: 56
T: 1 ( 1727) P: 1 I:10500 C: 137377 Min: 8 Act: 26 Avg: 20 Max: 57
T: 2 ( 1728) P: 1 I:11000 C: 131132 Min: 8 Act: 26 Avg: 20 Max:
161
T: 3 ( 1729) P: 1 I:11500 C: 125431 Min: 8 Act: 19 Avg: 20 Max: 57
T: 4 ( 1730) P: 1 I:12000 C: 120204 Min: 8 Act: 11 Avg: 20 Max: 66
T: 5 ( 1731) P: 1 I:12500 C: 115396 Min: 9 Act: 12 Avg: 20 Max: 57
T: 6 ( 1732) P: 1 I:13000 C: 110958 Min: 8 Act: 21 Avg: 20 Max: 56
T: 7 ( 1733) P: 1 I:13500 C: 106848 Min: 8 Act: 27 Avg: 20 Max: 63
```

The test showed real-time performance results running on the Linux kernel without a virtualization environment. The average latency was below 50us, and the top latency was within an approximately 200us timeframe. These results should be considered as a top reference value when operating in a virtualized environment with a small overhead.

Virtualization Environment

The Nautilus™ system ran several operating systems in a Xen virtualization environment (i.e., hypervisor). The hypervisor is a version of Xen for ARM-based devices that presents small performance overhead. The Nautilus™ system is represented by the three different domains:

1. Domain-0: thin domain with SATA support
2. Domain-D: driver domain
3. Android_4.3: guest domain

For reliability, Nautilus™ split drivers of Xen for ARM isolate faulty and error-prone device drivers from user domains by locating physical drivers to the driver domain. Since a user domain is completely separated from the driver domain, a user domain can reliably perform its task even if the driver domain is compromised and fails.

Each domain ran Linux kernel 3.8.13 and consumed two virtual processor resources, running a SMP kernel on each virtual core. The hypervisor allocated six virtual processor interfaces total, scheduling each particular processor to one of two physical cores.

By default, the hypervisor scheduled all virtual CPUs on physical cores using a credit scheduler so that the virtual CPU timeslot was granted for the specific time frames (i.e., budget). Xen implemented the run queue of a virtual CPU, and the credit scheduler queued a specific virtual processor from the physical CPU pool based on a round-robin algorithm.

RT-XEN Support

The RT-XEN project extends Xen, the most widely used open source virtual machine manager, to support real-time systems. RT-XEN uses the hierarchical scheduling theory to bridge the gap between virtualization and real-time systems by introducing schedulers in Xen that are compatible with schedulers in guest operating systems.

Xen 4.5 provides the RTDS scheduler required for time-critical tasks. There are several scheduling schemes available, including global scheduling (i.e., all cores share one run queue with a global spinlock) and partition scheduling (i.e., one run queue per core without a spinlock).

Global Scheduling

- Schedules virtual CPU based on global information
- Allows virtual CPU migration across cores
- Flexible use of multiple cores (pros)
- Migration (global spinlock) overhead and L1/L2 cache penalty (cons)

Partition Scheduling

- Assigns and binds virtual CPU to physical core
- Schedules virtual CPU on each core independently
- May underutilize physical core
- No migration overhead (global spinlock) or associated cache penalty

Global Scheduling Evaluation

A system may specify to run global scheduling by passing a *"sched = rtds"* option via boot arguments, which could be achieved either using boot loader settings or a thin domain device tree running under a Xen 4.5 hypervisor. This way, Xen starts to globally schedule all the domains with the RTDS scheduler, and share processors' protected with a global spinlock run-queue among all virtual CPUs,.

We made an evaluation to support the global scheduler on a SMP system with a global virtual CPU to physical core allocation. The purpose of this evaluation was to measure for different scheduler configurations and to find optimal values on an idle system and under system load.

The time measurements made by the independent 32Khz clock source (i.e., high resolution timers) supported the driver domain. The default scheduler configuration supported 4ms budget and 10ms period for all domains. It looked as follows:

```
/ # xl sched-rtds
Cpupool Pool-0: sched=RTDS
Name ID Period Budget
Domain-0 0 10000 4000
Domain-D 1 10000 4000
android_4.3 2 10000 4000
```

We ran the cyclic test on the default configuration in the driver domain. The test does not perform output to the Xen console, thus it does not take into account the inter-domain latency that appears due to Xen-Events Intra-VM communication. This way, Intra-VM latency does not affect our test results.

```
# ./cyclictest -t4 -p20 -n -i 10000 -l 10000 -m > tmp && cat tmp | tail
T: 0 ( 883) P:20 I:10000 C: 10000 Min: 11 Act: 145 Avg: 135 Max:
621
T: 1 ( 884) P:20 I:10500 C: 9520 Min: 9 Act: 105 Avg: 135 Max:
5719
T: 2 ( 885) P:20 I:11000 C: 9083 Min: 11 Act: 80 Avg: 134 Max:
5368
T: 3 ( 886) P:20 I:11500 C: 8684 Min: 13 Act: 196 Avg: 157 Max:
2253
```

The test results demonstrated that all of the threads were scheduled within an approximately 4ms budget and a1ms global scheduling overhead.

The SoC peripheral clock resolution allowed us to make measurements with an accuracy of 325 ns. As a next step, we configured different budgets and periods for the domains. We configured Domain-D to schedule with a 400 us period with the same budget, simulating a real-time domain latency that does not exceed 1 millisecond. The domains that are not required to meet real-time constraints will have more relaxed values of period and budget (e.g. 10 milliseconds for the guest domain and 1 millisecond for the thin domain).

```
# xl sched-rtds -d Domain-D -p 400 -b 400
# xl sched-rtds -d Domain-0 -p 1000 -b 400
# xl sched-rtds -d android_4.3 -p 10000 -b 5000
```

We measured the system latency on Domain-D when the other domains stayed idle after the reconfiguration of the budget and period.

<u>Dual-Core SMP Test</u>

```
# ./cyclictest -t4 -p20 -n -i 10000 -l 10000 -m > tmp && cat tmp | tail
T: 0 ( 883) P:20 I:10000 C: 10000 Min: 11 Act: 170 Avg: 137 Max:
523
T: 1 ( 884) P:20 I:10500 C: 9522 Min: 11 Act: 16 Avg: 154 Max:
307
T: 2 ( 885) P:20 I:11000 C: 9087 Min: 11 Act: 208 Avg: 136 Max:
315
T: 3 ( 886) P:20 I:11500 C: 8689 Min: 10 Act: 169 Avg: 148 Max:
704
```

The dual-core SMP test showed that the average maximal latency was reduced to 400 microseconds with a small overhead. Applying affinity to the test allowed per virtual core latency breakthrough. The virtual Core 1 had a bit lower overhead since it wasn’t running the most interrupts. (Interrupts virtualization affects the average maximal system latency with a bit larger maximal overhead observed on Core 0.)

The next test ran the Whetstone benchmark on the non-real-time thin domain to measure system latency on the real-time driver domain. The cyclic test running on the driver domain showed similar results to the idle system when Domain-0 was loaded with the Whetstone benchmark:

```
# ./cyclictest -t4 -p20 -n -i 10000 -l 10000 -m > tmp && cat tmp | tail
T: 0 ( 891) P:20 I:10000 C: 10000 Min: 12 Act: 195 Avg: 157 Max:
314
T: 1 ( 892) P:20 I:10500 C: 9521 Min: 10 Act: 133 Avg: 146 Max:
335
T: 2 ( 893) P:20 I:11000 C: 9087 Min: 10 Act: 145 Avg: 108 Max:
318
T: 3 ( 894) P:20 I:11500 C: 8690 Min: 14 Act: 107 Avg: 141 Max:
367
```

This simulation proved that the RTDS scheduler continued to transfer the control to the driver domain within the predefined period and budget of 400 microseconds, independent of system load. This way, it might be possible to configure different domains to run with a specified budget and period.

Partitioned Scheduling Evaluation

Partition scheduling subscribes to the idea of reallocating all the time-critical code to the separate physical core. This way, the scheduler avoids rescheduling non-time-critical tasks and locking all the cores globally. The following experiments proved that the system is capable of achieving real-time performance when separating time-critical tasks (i.e., OSes) to the separate physical processor by assigning and binding virtual CPUs to the physical core and scheduling virtual CPUs on each core independently.

We bound all non-real-time domains to Core 0 and all real-time domains to Core 1, with a specified budget of 1ms.

```
#xl vcpu-pin Domain-0 all 0
#xl vcpu-pin android_4.3 all 0
#xl vcpu-pin Domain-D all 1
#xl sched-rtds -d Domain-D -p 1000 -b 1000
```

The test running in Domain-D showed an average latency below 100 microseconds and a top latency below 1 millisecond, fitting real-time requirements.

```
# ./cyclictest -t4 -p 20 -n -i 10000 -l 1000 -m
/dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 0.00 0.04 0.05 1/72 892
T: 0 ( 889) P:20 I:10000 C: 459 Min: 17 Act: 56 Avg: 72 Max: 950
T: 1 ( 890) P:20 I:10500 C: 436 Min: 34 Act: 116 Avg: 88 Max: 973
T: 2 ( 891) P:20 I:11000 C: 414 Min: 38 Act: 50 Avg: 95 Max: 880
T: 3 ( 892) P:20 I:11500 C: 395 Min: 19 Act: 54 Avg: 73 Max: 444
```

Mixed Scheduling Evaluation

This is another test scenario to support different schedulers in different physical CPU pools. As the hypervisor schedules all virtual CPUs on a physical core using the default credit scheduler, the real-time domain has to migrate to the physical CPU. This way, the system binds partitioned credit and real-time schedulers to the dedicated PCPUs.

Since the guest domain is the major source of latency, we will first look into the latency with the guest domain. Then we will analyze latency with Dom0 for further investigation.

To bind the real-time domain, we performed a test with the following command sequence:

```
/ # xl cpupool-cpu-remove Pool-0 1
/ # xl cpupool-create name=\"realtime\" sched=\"rtds\"
/ # xl cpupool-cpu-add realtime 1
/ # xl cpupool-migrate Domain-D realtime
/ # xl sched-rtds -d Domain-D -p 1000 -b 1000
```

The test running in the driver domain showed an average latency below 100 microseconds and a top latency below 1 millisecond, fitting real-time requirements.

```
# ./cyclictest -t4 -p20 -n -i 10000 -l 1000 -m > tmp
# cat tmp
/dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 1.28 0.50 0.18 1/70 894
T: 0 ( 891) P:20 I:10000 C: 1000 Min: 11 Act: 32 Avg: 66 Max: 127
T: 1 ( 892) P:20 I:10500 C: 951 Min: 11 Act: 91 Avg: 66 Max: 132
T: 2 ( 893) P:20 I:11000 C: 906 Min: 11 Act: 34 Avg: 65 Max: 132
T: 3 ( 894) P:20 I:11500 C: 865 Min: 12 Act
```

CyclicTest Evaluation

We investigated the global and partitioned real-time scheduling. We discovered that partitioned and mixed scheduling schemes were suitable for time-critical tasks (e.g. real-time support), while global real-time scheduling required domain parameter configuration and tuning. Global real-time scheduling appears to provide a larger overhead due to virtual CPU migration and global lock based run-queue management and cache penalties.

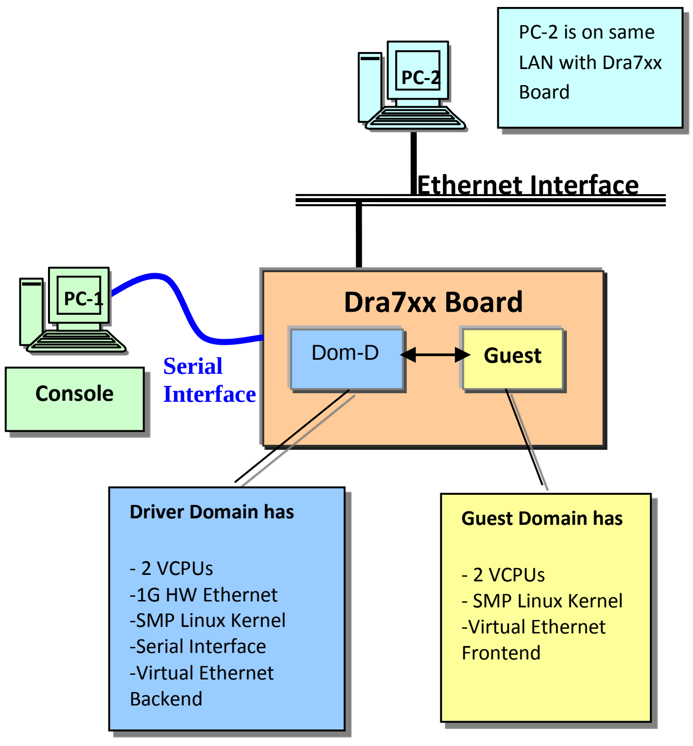
A pro of partitioned real-time scheduling is a real-time domain performance on the dedicated physical processor core, while the con is a performance drop of the domains scheduled on another physical core. Alternatively, global scheduling provides larger overhead with no expected performance drop for non-real-time domains.

Packet Latency

This section presents the packet latency in RT-XEN by measuring the packet latency with ping tests. The measurement environment included three guests OSes run over RT-XEN on the Nautilus ™ system.

The driver domain acted as a server for communication with the host PC over Ethernet and provided a virtual Ethernet interface for communication with the guest domain. The guest domain was a ping recipient domain that had soft real-time requirements in handling network packets. The thin domain had 100% CPU workloads to ensure work-conserving execution. An external server sent ICMP request packets to the guest domain over NAT. RT-XEN used the RTDS scheduler, and a timer tick was set to 10ms. We obtained the distribution graphs from 100 ping samples.

Since the guest domain is the major source of latency, we will first look into the latency within the guest domain. Then we will analyze latency within the driver domain for further investigation.



Credit Scheduler Packet Latency

First, we measured packet latency within the guest domain and looked at the effect of the credit scheduler on the virtual CPU. To present packet latency within the guest domain, we measured the latency in the interval from I/O task completion at netback in the guest domain to I/O task completion at ICMP reply in the driver domain.

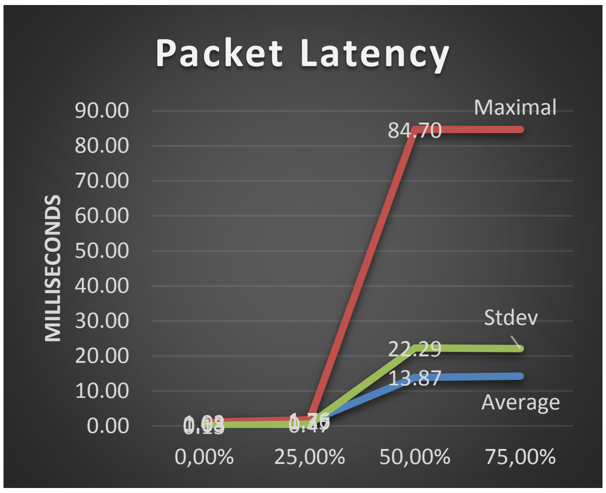
To test non-boosted VCPU, we gave different CPU-intensive workloads (i.e., 0%, 30%, 60%, and 75%) to the guest domain. In addition, we assumed that I/O completion latency within the guest domain was small enough to ignore because it was much smaller than the other latency. Therefore we intentionally omitted it from all our results. This small latency was partly due to ICMP handling in Linux kernel, in which ICMP packets were handled within a softirq context. As such, little overhead was involved.

Load	**Idle**	**30,00%**	**60,00%**	**75,00%**
	0,747	1,27	2,61	2,61
	0,728	1,65	8,93	8,93
	0,569	1,69	1,15	1,15
	0,619	0,609	1,15	1,15
	0,584	1,76	8,93	8,93
	0,817	1,71	8,93	8,93
	0,606	0,672	8,93	8,93
	0,585	1,59	8,93	8,93
	0,732	1,59	8,93	8,93
	0,673	1,59	8,93	8,93
	1,03	0,642	8,93	8,93
	0,556	1,63	8,93	8,93
	0,641	1,62	8,92	8,92
	0,645	1,62	8,93	8,93
	0,673	0,633	8,93	8,93
	0,989	1,66	8,93	8,93
	0,676	1,62	1,18	10,18
	0,634	1,62	79,3	79,3
	0,552	0,548	84,7	84,7
	0,692	1,68	1,18	1,18
	0,525	1,66	8,96	8,96
	0,706	0,531	8,92	8,92
Average	0,680864	1,345227	13,87409	14,28318
Max	1,03	1,76	84,7	84,7
Stdev	0,128651	0,472641	22,29284	22,13079

As shown in the table, when all domains had a 0% workload, 99% of the packets were handled within 1 millisecond. On the other hand, when the thin domain had a 100% CPU load (30% load), only 27% of the packets were handled within 1 millisecond.

The below figure shows a clear trend in which fewer packets were handled within 1 millisecond time-bound as the CPU load increased from 20% to 100%. This is mainly due to a non-boosted virtual CPU, which is affected by the CPU workload of itself. When we added a driver domain 100% virtual CPU load (60% total load), all of the packets missed the 1 millisecond deadline, with peak latency values being more than 100 milliseconds.

The figure also shows a clear trend in which fewer packets were handled within 1 millisecond time-bound as the CPU load increased from 0% to 75%. This is mainly due to a non-boosted virtual CPU, which is affected by the CPU workload of itself under the credit scheduler's control.



Also note that, as shown in the above figure, most packets were handled within 10ms (i.e., the scheduling period of the credit scheduler). When a packet handling was delayed by a non-boosted virtual CPU, it was dispatched at the next scheduling period. Consequently, 70ms + latency was presented.

Partitioned RT Packet Latency

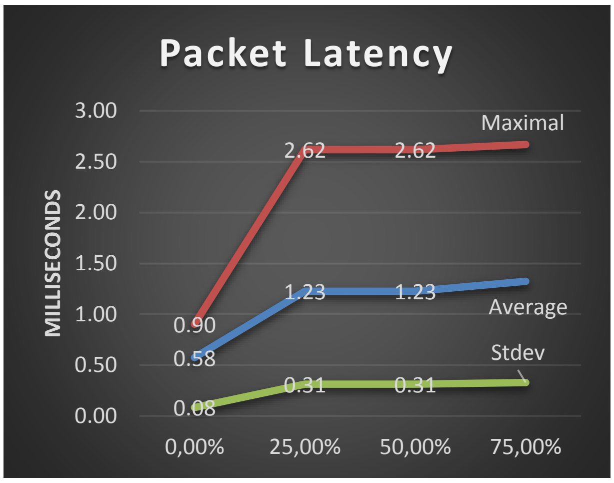
Next, we measured packet latency within the guest domain and looked at the effect of the RTDS scheduler of the virtual CPU. To present packet latency within the guest domain, we measured the latency in the interval from the host to the driver domain to I/O task completion in the guest domain at ICMP reply. We gave the same different CPU-intensive workloads (i.e., 0%, 30%, 60%, and 75%) to the thin and driver domains. Thus, the guest domain was intentionally omitted from all our results.

We configured the driver domain as a high-priority domain with a period and budget of 0.2ms and running net-backend. Since the guest domain could be preempted by the driver domain, we set it to a harmonic period and budget of 400 microseconds, running net-frontend. We kept the thin domain completely preempted by the others and set up 1 millisecond as its scheduling period. The below table represents the round-trip test results:

Load	**Idle**	**25,00%**	**50,00%**	**75,00%**
	0,903	2,62	2,62	2,67
	0,595	1,18	1,18	1,16
	0,554	1,18	1,18	1,25
	0,553	1,15	1,15	1,21
	0,586	1,15	1,15	1,24
	0,576	1,19	1,19	1,23
	0,58	1,16	1,16	1,13
	0,55	1,15	1,15	1,4
	0,55	1,17	1,17	1,37
	0,611	1,15	1,15	1,15
	0,475	1,18	1,18	1,16
	0,584	1,18	1,18	1,15
	0,593	1,15	1,15	1,46
	0,555	1,15	1,15	1,44
	0,635	1,15	1,15	1,43
	0,546	1,16	1,16	1,15
	0,459	1,16	1,16	1,15
	0,586	1,15	1,15	1,12
	0,529	1,15	1,15	1,49
	0,578	1,15	1,15	1,12
	0,5	1,15	1,15	1,15
	0,554	1,15	1,15	1,51
Average	0,575091	1,226364	1,2264	1,324545
Max	0,903	2,62	2,62	2,67
Stdev	0,084125	0,311564	0,3116	0,329888

In the above table, when all domains had a 0% workload, 100% of the packets were handled within 1 millisecond. When we loaded a less responsive thin domain 100% CPU load (30% total load), only 99% of the packets were handled within 1 millisecond with small scheduling overhead (i.e., 150 microseconds). The picture changed a bit when we added a driver domain 100% virtual CPU load (50% - 75% total load). The packet latency deadline adjusted to the small scheduling overhead, although the packets were missing the 1 millisecond deadline. The total average and peak latencies increased a bit to 1.32 and 2.67 milliseconds respectively.

The below figure shows a clear trend in which most packets were handled within 1 millisecond and scheduling overhead (1.5 approximately) was time-bound as the CPU load increased from 0% to 75%. This demonstrates a very smooth packet latency transition within such a configuration and could be applicable to soft-real-time needs.



Global RT Packet Latency

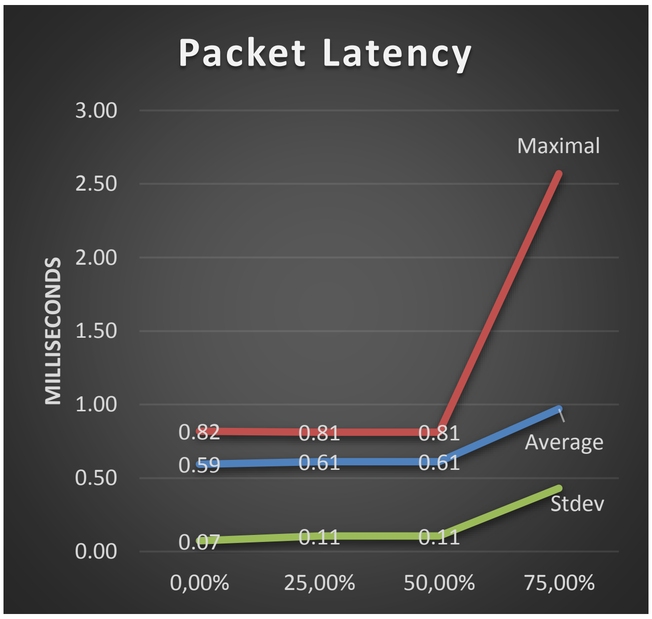
Eventually we would like to estimate the packet latency for the symmetric multi-processing support virtualization support. We are using the same measurement procedure to obtain the latency in the interval from the host to the driver domain to I/O task completion in the guest domain at ICMP reply. We gave the same different CPU-intensive workloads (i.e., 0%, 30%, 60%, and 75%) to the thin and driver domains.

The schedule configuration remained the same as that for the partitioned scheduled: 200 microseconds for netback at the driver domain, 400 microseconds for the front-end in the guest domain, and 1 millisecond for the thin domain. The below table represents the round-trip test results:

Load	**Idle**	**25,00%**	**50,00%**	**75,00%**
	0,631	0,764	0,764	0,771
	0,819	0,716	0,716	0,534
	0,549	0,663	0,663	0,612
	0,551	0,729	0,729	1,2
	0,515	0,492	0,492	0,855
	0,539	0,624	0,624	1,14
	0,581	0,652	0,652	2,57
	0,556	0,517	0,517	1,01
	0,589	0,659	0,659	1,33
	0,626	0,584	0,584	0,681
	0,612	0,501	0,501	1,13
	0,578	0,52	0,52	0,713
	0,626	0,61	0,61	1,31
	0,605	0,427	0,427	0,883
	0,524	0,644	0,644	1,08
	0,523	0,594	0,594	0,687
	0,677	0,757	0,757	0,714
	0,707	0,629	0,629	0,588
	0,508	0,812	0,812	1
	0,601	0,439	0,439	0,585
	0,535	0,588	0,588	1,05
	0,613	0,538	0,538	0,897
Average	0,5938636	0,611773	0,6117727	0,97
Max	0,819	0,812	0,812	2,57
Stdev	0,0725083	0,105114	0,105114	0,4313284

100% of packets were handled within 1 millisecond on the idle system. The thin domain load of 100% (30% total load) barely increased peak and average latencies. Again, the picture changes as we added the driver domain workload (75% total load). The packet latency deadline adjusted to small scheduling overhead, although the packets were missing the 1 millisecond deadlines. The total average and peak latencies increased similarly to the partitioned scheduling case.

The below figure shows a clear trend in which most packets were handled within 1 millisecond for CPU load increases of 0% to 30% (i.e., thin domain workload). The peak latency adjusted just when we added the driver domain (net-backend) to the game, while the average latency remained below the 1 millisecond bound.



RT-XEN vs Credit I/O Latency

We performed PV-NET latency measurements for three different scheduling configurations:

- Credit scheduler
- RT-Xen global scheduler
- RT-Xen partitioned scheduler

The results are available for different schedulers, and we performed every test for different workloads (i.e., idle, 25% load, 50% load, 75% load). The workload was generated by running several whetstone instances in the thin and driver domains.

The credit scheduler showed a very large maximal latency that is above 80 milliseconds under heavy load conditions, with 10 milliseconds being the average.

The partitioned RT-XEN scheduler showed optimal latency values. The worse-case packet latency deadline was 2.67 ms under heavy load conditions.

The global scheduler test results were pretty close to those of the partitioned scheduler, although the average and peak latency increased more slowly than the partitioned as workload progressed.

Conclusion

In this paper, we discussed scheduling the packet latency in RT-Xen. Due to the limitations of the current default credit scheduler and interrupt paravirtualization of guest OSes, a guest domain cannot handle network packets within a predictable time bound with Xen. To provide a predictable latency bound, this paper proposes using RT-Xen scheduler.

Our conclusion is that every domain configuration has an impact on the Intra-VM latency during the tuning of the global scheduler settings. In fact, the domain with a smaller period and budget is scheduled more often (thus being able to operate within time-critical constraints), while non-real-time domains should be configured with a larger period and the budget. This approach allows Xen to preempt non-real-time domains by real-time, thereby keeping the latency low even for an Intra-VM switch.