Report Number: WUCSE-2012-81

2012

Limitations and Solutions for Real-Time Local Inter-Domain Communication in Xen

Authors: Sisu Xi, Chong Li, Chenyang Lu, and Christopher Gill

As computer hardware becomes increasingly powerful, there is an ongoing trend towards integrating complex, legacy real-time systems using fewer hosts through virtualization. Especially in embedded systems domains such as avionics and automotive engineering, this kind of system integration can greatly reduce system weight, cost, and power requirements. When systems are integrated in this manner, network communication may become local inter-domain communication (IDC) within the same host. This paper examines the limitations of inter-domain communication in Xen, a widely used open-source virtual machine monitor (VMM) that recently has been extended to support real-time domain scheduling. We find that both the VMM scheduler and the manager domain can significantly impact real-time IDC performance under different conditions, and show that improving the VMM scheduler alone cannot deliver real-time performance for local IDC. To address those limitations, we present the RTCA, a Real-Time Communication Architecture within the manager domain in Xen, along with empirical evaluations whose results demonstrate that the latency of communication tasks can be improved dramatically from ms to μs by a combination of the RTCA... **Read complete abstract on page 2.**

Limitations and Solutions for Real-Time Local Inter-Domain Communication in Xen

Complete Abstract:

As computer hardware becomes increasingly powerful, there is an ongoing trend towards integrating complex, legacy real-time systems using fewer hosts through virtualization. Especially in embedded systems domains such as avionics and automotive engineering, this kind of system integration can greatly reduce system weight, cost, and power requirements. When systems are integrated in this manner, network communication may become local inter-domain communication (IDC) within the same host. This paper examines the limitations of inter-domain communication in Xen, a widely used open-source virtual machine monitor (VMM) that recently has been extended to support real-time domain scheduling. We find that both the VMM scheduler and the manager domain can significantly impact real-time IDC performance under different conditions, and show that improving the VMM scheduler alone cannot deliver real-time performance for local IDC. To address those limitations, we present the RTCA, a Real-Time Communication Architecture within the manager domain in Xen, along with empirical evaluations whose results demonstrate that the latency of communication tasks can be improved dramatically from ms to μs by a combination of the RTCA and a real-time VMM scheduler.

Limitations and Solutions for Real-Time Local Inter-Domain Communication in Xen

Sisu Xi, Chong Li, Chenyang Lu, and Christopher Gill
Department of Computer Science and Engineering
Washington University in Saint Louis
Email: {xis, lu, cdgill}@cse.wustl.edu, chong.li@wustl.edu

***Abstract*—As computer hardware becomes increasingly powerful, there is an ongoing trend towards integrating complex, legacy real-time systems using fewer hosts through virtualization. Especially in embedded systems domains such as avionics and automotive engineering, this kind of system integration can greatly reduce system weight, cost, and power requirements. When systems are integrated in this manner, *network communication* may become local *inter-domain communication* (IDC) within the same host. This paper examines the limitations of inter-domain communication in Xen, a widely used open-source virtual machine monitor (VMM) that recently has been extended to support real-time domain scheduling. We find that both the VMM scheduler and the manager domain can significantly impact real-time IDC performance under different conditions, and show that improving the VMM scheduler alone cannot deliver real-time performance for local IDC. To address those limitations, we present the RTCA, a Real-Time Communication Architecture within the manager domain in Xen, along with empirical evaluations whose results demonstrate that the latency of communication tasks can be improved dramatically from ms to μs by a combination of the RTCA and a real-time VMM scheduler.**

I. Introduction

Modern virtualized systems may seat as many as forty to sixty virtual machines (VM) per physical host [1], and with the increasing popularity of 32-core and 64-core machines [2], the number of VMs per host is likely to keep growing. In the mean time there has been increasing interest in integrating multiple independently developed real-time systems on a common virtualized computing platform. When systems are integrated in this manner, a significant amount of *network communication* may become *local inter-domain communication* (IDC) within the same host.

This paper closely examines the real-time IDC performance of Xen [3], a widely used open-source virtual machine monitor that has been extended to support real-time domain scheduling [4], [5], and points out its key limitations that can cause significant priority inversion in IDC. We show experimentally that improving the VMM scheduler alone cannot achieve real-time performance of IDC, and to address that problem we have designed and implemented a *Real-Time Communication Architecture* (RTCA) within the manager domain that handles communication between guest domains in Xen. Empirical evaluations of our approach demonstrate that the latency of IDC can be improved dramatically from ms to μs by a combination of the RTCA and a real-time VMM scheduler.

A key observation from our analysis and empirical studies is that both the VMM scheduler and the manager domain can affect real-time communication performance. While the former needs to schedule the right domain at the right time to send or receive packets, the latter should provide bounded delays for transferring packets. In this paper, we focus on evaluating each part both separately and in combination. The results of our experiments show that even if the VMM scheduler always makes the right decision, due to limitations of the Xen communication architecture, the IDC latency for even high priority domains can go from μs under no interference to ms with interference from lower priority domains. In contrast, applying the RTCA reduces priority inversion in the manager domain and provides appropriate real-time communication semantics to domains at different priority levels when used in combination with a real-time VMM scheduler.

Specifically, this paper makes the following contributions to the state of the art in real-time virtualization. (1) We identify the key limitations of Xen for real-time IDC in both the manager domain and the VMM scheduler through both quantitative analysis and a systematic experimental study. (2) It introduces the RTCA, a Real-Time Communication Architecture within the manager domain in Xen. By altering the packet processing order and tuning the batch sizes, the response time for higher priority packets can be improved significantly due to reduced priority inversion. (3) It presents comprehensive experimental results with which we evaluate the effect of the VMM scheduler as well as the manager domain. By combining the RTCA and an existing RT-Xen scheduler, the latency is greatly improved for high-priority domains (from ms to μs) in the presence of heavy low-priority traffic.

II. Background

This section provides background information about the virtual machine monitor (VMM) and the key communication architecture components in Xen.

A. Xen Virtual Machine Monitor

A *virtual machine monitor* (VMM) allows a set of guest operating systems (*guest domains*) to run concurrently on the same host. Developed by Harham et al. in 2003, Xen [3] has become the most widely used open-source VMM. It is a stand-alone VMM, where the VMM lies between all domains

and the hardware, providing *virtual memory*, *virtual network* and *virtual CPU* (VCPU) resources to the guest and manager domains running atop it.

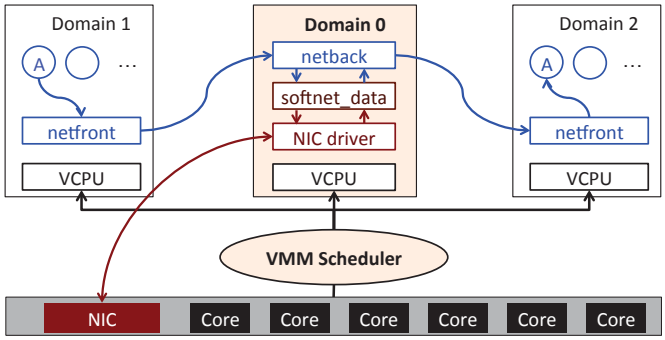


Fig. 1: Xen Communication Architecture Overview

Figure 1 gives an overview of the communication architecture in Xen. A manager domain, referred to as *Domain 0*, is responsible for creating, suspending, resuming, and destroying other (guest) domains. *Domain 0* runs Linux to perform all these functions, while guest domains can use any operating system. Each domain has a set of Virtual CPUs (VCPUs) in the VMM, and the VCPUs are scheduled by a VMM scheduler. For IDC, *Domain 0* contains a *netback* driver which coordinates with a *netfront* driver in each guest domain. For example, the upper connecting lines in Figure 1 show the inter-domain communication for application A from Domain 1 to Domain 2. Application A first sends the packets to the netfront driver in Domain 1 via its socket interface; the netfront driver delivers the packets to *Domain 0*; *Domain 0* examines each packet, finds it is for a local domain, delivers it to Domain 2 and notifies the VMM scheduler; the netfront driver in Domain 2 sends the packets to application A. Note that the applications running atop the guest domains are not aware of this para-virtualization, so no modification to them is needed. Another approach for IDC is to use shared memory to exchange data between domains [6]–[9], thus avoiding the involvement of *Domain 0* to obtain better performance. However, the shared memory approach requires changes to the guest domain, and may even need to change the application as well (a detailed discussion is deferred to Section VII). *Domain 0* also contains a NIC driver and if a packet is for another host, it direct the packets to the NIC driver which in turn sends it out via the network. Improving the real-time performance of inter-host communication is outside the scope of this paper and will be considered as future work.

As Figure 1 illustrates, in IDC two parts play important roles: (1) the VMM scheduler, which needs to schedule the corresponding domain when it has pending/coming packets; and (2) the netback driver in *Domain 0*, which needs to process each packet with reasonable latency. The VMM scheduler has been discussed thoroughly in prior published research [4], [5], [10]–[14], while *Domain 0* is usually treated as a black box with respect to IDC.

B. Default Credit Scheduler

Xen by default provides two schedulers: a Simple Earliest Deadline First (SEDF) scheduler and a proportional share (Credit) scheduler. The SEDF scheduler applies dynamic priorities based on deadlines, and does not treat I/O domains specially. Communication-aware scheduling [10] improves SEDF by raising the priorities of I/O intensive domains, and always scheduling *Domain 0* first when it is competing with other domains. However, SEDF is no longer in active development, and will be phased out in the near future [15].

The Credit scheduler schedules domains in a round-robin order with a quantum of 30 ms. It schedules domains in three categories: BOOST, UNDER, and OVER. BOOST contains VCPUs that are blocked on incoming I/O, while UNDER contains VCPUs that still have credit to run, and OVER contains VCPUs that have run out of credit. The BOOST category is scheduled in FIFO order, and after execution each domain from it is placed into the UNDER category, while UNDER and OVER are scheduled in a round-robin manner. Ongaro et al. [11] studies I/O performance under 8 different workloads using 11 variants of both Credit and SEDF schedulers. The results show that latency cannot be guaranteed since it depends on both CPU and I/O interference and the boot order of the VMs.

C. RT-Xen Scheduler

In previous work, we have developed RT-Xen [4], [5] which allows users to configure a set of fixed priority schedulers. In this paper, we use a deferrable server scheduler: whenever the domain still has budget but no tasks to run, the remaining budget is preserved for future use.

If a domain has packets to send, it is scheduled according to its priority. Thus, if the domain has the highest priority and also has budget left, it will be scheduled first. When the scheduler is notified that a domain has a packet (via the *wake_up()* function), it compares the domain's priority with the currently running one: if the newly awakened domain has higher priority, it will immediately interrupt the current domain and be scheduled; otherwise it will be inserted into the run queue according to its priority.

D. IDC in Domain 0

To explain how IDC is performed in *Domain 0*, we now describe how Linux processes packets, and how the softirq and kernel thread behavior, and show how Xen hooks its netfront and netback drivers into that execution architecture to process packets.

When a guest domain sends a packet, an *interrupt* is raised to notify the kernel. To reduce context switching and potential cache pollution which can produce *receive livelock* [16], Linux 2.6 and later versions have used the New API packet reception mechanism [17]. The basic idea is that only the first packet raises a NET_RX_SOFTIRQ, and after that the interrupt is disabled and all the following packets are queued without generating interrupts. The softirqs are scheduled by a per-CPU

kernel thread named ksoftirq. Also, a per-CPU data structure called softnet_data is created to hold the incoming packets.

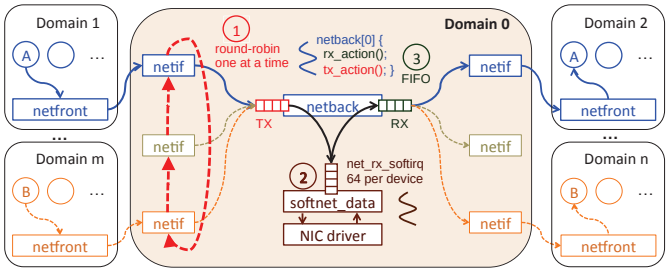


Fig. 2: Xen Communication Architecture in Domain 0

As shown in Figure 1, Xen uses the netfront and netback drivers to transmit packets between guest and manager domains. Figure 2 demonstrates in detail how *Domain 0* works with the source domains on the left sending packets to the destination domains on the right. When *Domain 0* boots up, it creates as many netback devices as it has VCPUs (here we only consider the single core case, with a single netback device in *Domain 0*). The netback device maintains two queues: a TX Queue for receiving packets from all guest domains, and an RX Queue for transmitting packets to all guest domains. They are processed by a single *kernel thread* in Xen 4.1. The kernel thread always performs the *net_rx_action()* first to process the RX Queue, and then performs the *net_tx_action()* to process the TX Queue. When a guest domain boots up, it creates a netif device in *Domain 0* and links it to its netback device.

Within the *Domain 0* kernel, all the netback devices are represented by one backlog device and are treated the same as any other device (e.g., a NIC). As can be seen from Figure 2, when an IDC flow goes through *Domain 0*, there are three queues involved, which we now consider in order by where the packets are processed.

Netback TX Queue: The netback device maintains a *schedule_list* with all netif devices that have pending packets. When the *net_tx_action()* is processed, it picks the first netif device in the list, processes one packet, and if it still has pending packets puts the netif device at the end of the list, which results in a round-robin transmission order with a batch size of 1. In one round, it processes up to some number of packets, which is related to the page size on the machine: on our 64-bit Linux machine, that number is 238. If there are still packets pending after a round, it notifies the scheduler to schedule the kernel thread again later. Xen by default adopts a token-bucket algorithm [18] to achieve rate limiting for each domain within this stage; if a netif device has pending packets but exceeds the rate limit, Xen instead picks the next one. In this paper, we leave the rate control default (unlimited) as it is and instead change the order of pending packets. Our approach can be seamlessly integrated with default or improved rate control mechanisms [14].

Softnet_Data Queue: All the packets dequeued from the TX Queue are enqueued into a single softnet_data queue. *Domain 0* processes this queue when responding to the NET_RX_SOFTIRQ. A list of all active devices (usually NIC and backlog) is maintained, and *Domain 0* processes up to 64 packets for the first device, puts it at the end of the list, and then processes the next one, also resulting in a round-robin order with a batch size of 64. In one round, the function quits after either a total of 300 packets are processed or 2 jiffies have passed. If there are still pending packets at the end of a round, another NET_RX_SOFTIRQ is raised. When processing the packets, if *Domain 0* finds that its destination is a local domain, it bridges it to the RX Queue in the corresponding netback device; if it is the first packet, it also notifies the scheduler to schedule the kernel thread. Note that there is also a 1000 packet limit for the backlog device [19]. We only consider IDC in this paper and defer integration with the NIC as future work.

Netback RX Queue: Similar to the TX Queue, the netback driver also has an RX Queue (associated with net_rx_action()) that contains packets whose destination domain's netif is associated with that netback device. All the packets in this case are processed in FIFO order and are delivered to the corresponding netif device. Note that this queue also has a limit (238) for one round, and after that if there are still packets pending, it tells the scheduler to schedule them later.

III. Limitations of the Communication Architecture in Xen

As Figure 1 shows, both the VMM scheduler and *Domain 0* play important roles. However, neither of them alone can guarantee real-time I/O performance.

The default Credit scheduler has two major problems: (1) it schedules outgoing packets in a round-robin fashion with a quantum of 30 ms, which is too coarse; (2) for incoming packets, it applies a general boost to a blocked VCPU. Several papers improve the Credit scheduler. Specifically, for problem (1), Cheng et al. [14] provide a dual run-queue scheduler: for VCPUs with periodic outgoing packets, they are scheduled in a Earliest Deadline First queue, while for other VCPUs are still scheduled in the default Credit queue. For problem (2), Lee et al. [12] patched the Credit scheduler so it will boost not only blocked CPUs, but also active CPUs (so that if a VCPU also runs a background CPU intensive task, it can benefit from the boost as well). However, note that none of those approaches strictly prioritize VCPUs. When there are multiple domains doing I/O together, they are all scheduled in a round-robin fashion.

The RT-Xen scheduler [4], [5] applies a strict priority policy for VCPUs for both outgoing and incoming packets, and thus can easily prevent interference from lower priority domains within the same core. However, it uses 1 ms as the scheduling quantum, and when a domain executes for less than 0.5 ms, its budget is not consumed. On a modern machine, however, the typical time for a domain to send a packet is less than 10 μs. Consider a case where one packet is bouncing between two domains on the same core: if these two domains runs no other tasks, the RT-Xen scheduler would switch rapidly between these two domains, with each executing for only

about 10 μs. As a result, neither domain's budget will be reduced, resulting a 50% share for each regardless of their budget and period configuration. This clearly violates the resource isolation property of the VMM scheduler. In this paper, we address this limitation of our previous work by providing a dual resolution: μs for CPU time accounting, and ms for VCPU scheduling. The dual resolution provides better resource isolation, while maintaining appropriate scheduling 1 ms scheduling quantum for real-time applications. For all the evaluations in this paper we use this improved RT-Xen scheduler.

Domain 0 also has the following major limitations in terms of real-time performance:

No Prioritization between Domains: As was described above, all three queues (TX, softnet_data, and RX) are shared by all guest domains together with a round-robin policy for processing the TX and softnet_data queues, which can lead to priority inversion. We show in Section VI that even under light interference from other cores (which cannot be prevented by any VMM scheduler), the I/O performance for high priority domains is severely affected.

Mismatched Sizes: the TX and RX Queues have total processing sizes of 238 with batch size 1 for each domain, while the softnet_data queue has a total processing size of 300 with batch size 64 for each device. These large and mismatched sizes make timing analysis difficult and may degrade performance. For example, under a heavy IDC workload where a NIC also is doing heavy communication, the softnet_data queue (total size of 300) is equally shared by backlog and NIC devices. Every time the TX Queue delivers 238 packets to the softnet_data queue, the softnet_data queue is only able to process 150 of them, causing the backlog queue to become full and to start dropping packets when its limit of 1000 packets is reached.

No Strict Prioritization between Queues: Ideally the three queues would support multiple priorities, and the higher priority packets could pre-empt lower priority ones. Before Linux 3.0, TX and RX processing was executed by two TASKLETs in arbitrary order. As a result, the "TX - softnet_data - RX" stage could be interrupted by the RX processing for previous packets and by the TX processing for future packets. Linux (as of version 3.0 and later) fixed this by using one kernel thread to process both TX and RX Queues, with the RX Queue always being processed first. However, this introduces another problem that the higher priority packets may need to wait until a previous lower priority one has finished transmission.

IV. Quantifying the Effects of the VMM Scheduler and Domain 0

Since both the VMM scheduler and *Domain 0* can affect real-time I/O performance, in this section we examine which one is more important under typical situations. We studied the effect of the scheduler by pinning all guest domains to a single core and running extensive I/O. We compared the priority boost in RT-Xen versus the general boost in the default Credit scheduler. We then conducted a simple experiment to demonstrate the effect of *Domain 0*.

Experimental Setup

The experiments were performed on an Intel i7-980 six core machine with hyper-threading disabled. SpeedStep was disabled by default, and each core ran at 3.33 GHz constantly. We installed 64-bit CentOS with para-virtualized kernel 3.4.2 in both *Domain 0* and the guest domains, together with Xen 4.1.2 after applying the RT-Xen patch. We focused on the single-core case with every domain configured with one VCPU, and we dedicated core 0 to *Domain 0* with 1 GB memory, as *Domain 0* also works as the manager domain. Dedicating a separate core to handle communication and interrupts is a common practice in multi-core real-time systems research [2]. It is also recommended by the Xen community to improve I/O performance [20]. During our experiments we disabled the NIC and configured all the guest domains within a local IP address, focusing on local inter-domain communication (IDC) only. We also shut down all other unnecessary services to minimize incidental sources of interference. Data were then collected from the guest domains when the experiments were completed. Please note that *Domain 0* does not itself run other tasks that might interfere with its packet processing.

A. Effect of the VMM Scheduler: Credit vs. RT-Xen

The experiment presented in this section examines the effect of the VMM scheduler when all interference is coming from the same core. We booted ten domains and pinned all of them to core 1 (*Domain 0* still owns core 0). Each guest domain had 10% CPU share, which was achieved via the -c parameter in the Credit scheduler, and by configuring a budget of 1 and a period of 10 in the RT-Xen scheduler. We configured Domain 1 and Domain 2 with highest priority and measured the round-trip time between them: Domain 1 sent out 1 packet every 10 ms, and Domain 2 echoed it back. As in our previous work [4], [5], the *rdtsc* command was used to measure time. For each experiment, we recorded 5,000 data points. For the remaining eight domains, we configured them to work in four pairs and bounced a packet constantly between each pair. Note that all 10 domains were doing I/O in a blocked state, and thus they would all be boosted by the Credit scheduler. As expected, when Domain 1 or Domain 2 was inserted at the end of the BOOST category, the queue was already very long (with eight interfering domains thus creating a priority inversion). In contrast, the RT-Xen scheduler would always schedule domains based on priority.

Figure 3 shows a CDF plot of the latency with a percentile point every 5%. The solid lines show the results using the RT-Xen scheduler, and the dashed lines represent the Credit scheduler. The lines with diamond markers were obtained using the original kernel, and the lines with circles were obtained using our improved RTCA, which is discussed in Sections V and VI-A. We can clearly see that due to the general boost, the Credit scheduler's I/O performance is severely affected, growing from around 80 μs to around 160

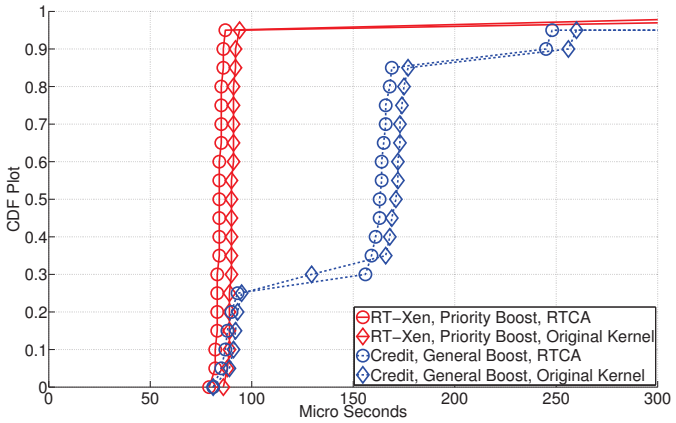


Fig. 3: Effect of the VMM Scheduler: Credit VS. RT-Xen

μs at 30%, and further extending to 250 μs at 90%. In contrast, the RT-Xen scheduler can limit the latency within 100 μs until the 95th percentile. We also noticed that when we were doing experiments, *Domain 0*'s CPU utilization stayed around 60%, indicating it was more than capable of processing the I/O load it was offered.

Summary*: RT-Xen can apply strict prioritization of VCPUs, preventing interference within the same core.*

B. The VMM Scheduler is not Enough

We have shown that by appropriately boosting the VCPU, we can deliver better I/O performance for high priority domains. However, as we have discussed earlier, *Domain 0* could also become a bottleneck when processing I/O, especially when there is lots of I/O from other cores.

A simple setup is used here to demonstrate the effect of *Domain 0*. We again pinned *Domain 0* to core 0, and dedicated core 1 and core 2 to Domain 1 and Domain 2, respectively, so the VMM scheduler would not matter. The same workload still ran between Domain 1 and Domain 2 and we measured the round trip times. For the remaining three cores, we booted three domains on each core with all of them doing extensive I/O, creating a heavy load on *Domain 0*.

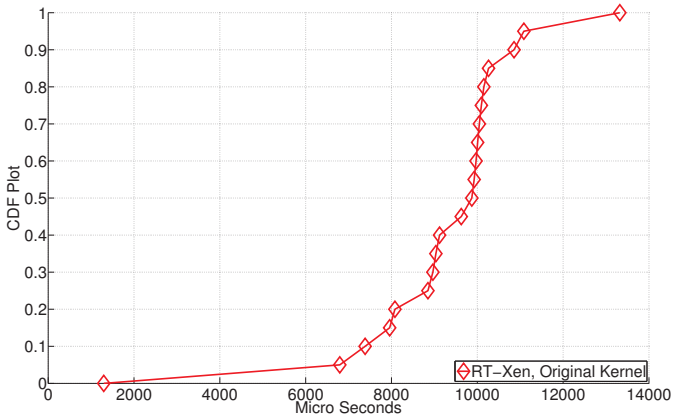


Fig. 4: Bottleneck in Domain 0

Figure 4 shows the CDF plot of the results with a sampling point every 5th percentile. Please note the larger x axis range in this figure. The latency grew from the μs level to more than 6 ms. This configuration represents the best the VMM scheduler can do, since all the interference came from *Domain 0*, and any improvement to the VMM scheduler thus cannot help.

V. Real-Time Communication Architecture

To address the limitations of *Domain 0*, this section presents a new *Real-Time Communication Architecture* (RTCA). We first discuss how we change the TX and softnet_data queues to make them more responsive and real-time aware. We then give a concrete example showing the packet processing order in the three queues, both in the RTCA and in the original version, along with a discussion of possible further improvements.

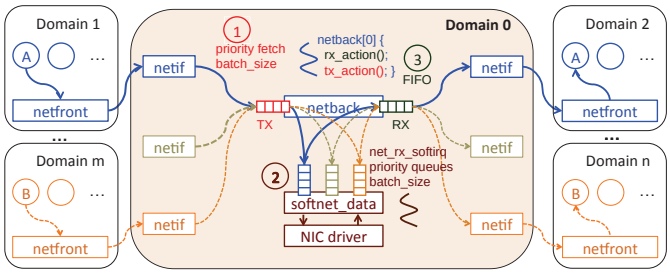
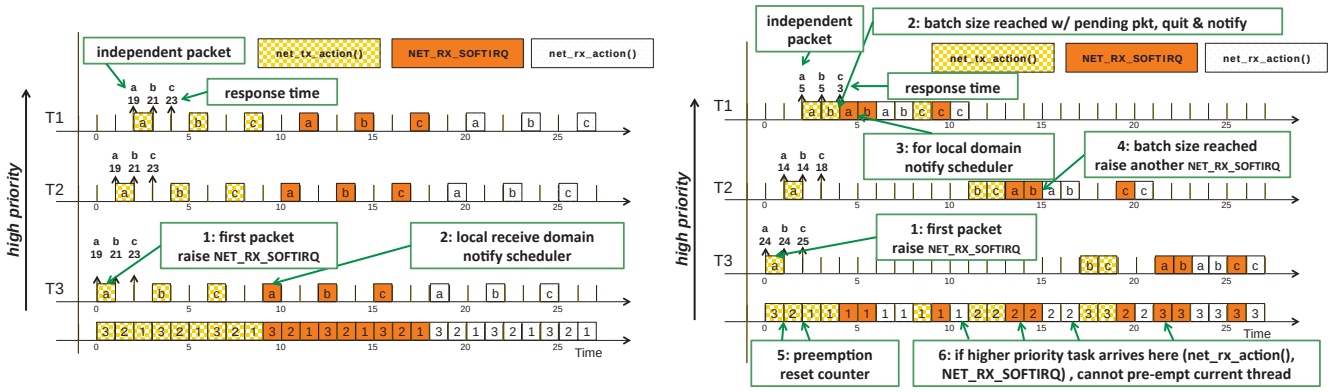


Fig. 5: RTCA: Real-Time Communication Architecture

Figure 5 shows the RTCA in Xen. Note that a key design principle is to minimize priority inversion as much as possible within *Domain 0*. We now discuss the changes we made to each of the three queues.

Algorithm 1 net_tx_action()

```
1:  cur priority = highest active netif priority
2:  total = 0
3:  counter = 0
4:  while schedule_list not empty &&
      counter < batch size && total < round limit do
5:    fetch the highest priority active netif device
6:    if its priority is higher than cur priority then
7:      reset counter to 0
8:      reset current priority to its priority
9:    end if
10:   enqueue one packet
11:   counter++, total++
12:   update information including packet order, total size
13:   if the netif device still has pending packets then
14:     put the netif device back into the schedule list
15:   end if
16: end while
17: dequeue from TX Queue to softnet_data queue
    raise NET_RX_SOFTIRQ for first packet
18: if schedule_list not empty then
19:   notify the scheduler
20: end if
```



(a) Original Kernel

(b) RTCA Kernel

Fig. 6: Packet Processing Illustration

Netback TX Queue: Algorithm 1 describes how we process the packets in the net_tx_action() function. Instead of a round-robin policy, we now fetch packets according to their priority, one at a time. We also make the batch size tunable for each netif individually, to make *Domain 0* more flexible and configurable for different system integrators. The packets are processed one at a time because during the processing of lower priority domains, a higher priority domain may become active and dynamically add its netif into the schedule list. Making a prioritized decision at each packet thus minimizes priority inversion. Note that due to other information kept separately in the netback driver about the packet order, neither splitting the queue nor simply reordering it is easily achievable without causing a kernel panic[1]. As a result, the TX Queue is dequeued in FIFO order. However, whenever a higher priority domain arrives, we reset the counter for it. Our evaluation in Section VI shows that with a batch size of 1, the system had suitable real-time latency and throughput performance. If that setting is used, the total size limit of 238 is unlikely to be reached, and so the total number of packets for a high-priority domain is unlikely to be limited by the previously processed lower priority domains.

Softnet_Data Queue: Since the packets coming from the TX Queue might be from different domains, we split the queue by priorities, and only process the highest priority one within each NET_RX_SOFTIRQ. The batch size is also a tunable parameter for each queue. Moreover, under a heavy overload, the lower priority queues can easily be filled up, making the total size limit for all the softnet_data queues easily reached. Therefore, we eliminate the total limit of 1000 packets for all domains, and instead set an individual limit of 600 for each softnet_data queue. Note that this parameter is also tunable by system integrators at their discretion.

Netback RX Queue: As the packets coming from the softnet_data queue are only from one priority level, there is no need to split this queue. Moreover, by appropriately configuring the batch size for the softnet_data queue (making it less than 238), the capacity of the RX Queue will always be enough. For these reasons, we made no modification to the net_rx_action() function. Please note that both the softnet_data and RX Queues are non-preemptible: once the kernel begins processing them even for the lower priority domains, an arriving higher priority domain packet can only notify the kernel thread and has to wait until the next round to be processed.

Without changing the fundamental processing architecture, we keep most of the benefits of Xen (for example, the existing rate control mechanism can be seamlessly integrated with our modifications), while significantly improving the real-time communication response time (as shown in Section VI) by an order of magnitude for higher priority domains, resulting in μs level timing that is suitable for many soft real-time systems.

Examples for Packet Processing

To better illustrate how our approach works, we show the packet processing order both in the RTCA (Figure 6b) and in the original kernel (Figure 6a), assuming that the guest domains always get the physical CPU when they want it (a perfect VMM scheduler). Both examples use the same task set, where three domains (T3, T2 and T1 with increasing priority) are trying to send three individual packets successively starting from time 1, 2, and 3. The lowest line of each figure shows the processing order for each domain, and the corresponding upper lines show the processing order for individual packets in each domain. To better illustrate pre-emption in the TX Queue, all three domains are configured with a batch size of 2 in the TX and softnet_data queues. The upper arrow shows the release of the packet, and the number above the arrow shows the response time for each packet.

Several key observations can be made here:

- The RTCA greatly improves the response times for packets in higher priority domains (from 19, 21, 23 to 5, 5, 3). Since unmodified Xen processes packets in a round-robin order, and uses a relatively large batch size

[1]As future work, we plan to examine how to address this remaining limitation.

for all three queues, the response time is identical for each domain; in contrast, the RTCA prioritizes the processing order and imposes a smaller batch size, resulting in better responsiveness for higher priority domains.
- Whenever the batch size is reached and there are still pending packets, or when the first packet arrives, either a softirq is raised or the scheduler is notified (points 1, 2, 3, and 4 in Figure 6b; points 1 and 2 in Figure 6a).
- In the RTCA, TX Queue processing is pre-emptive, and every time a high-priority domain packet arrives, the counter is reset (point 5 in Figure 6b).
- The softnet_data and RX Queue processing is non-pre-emptive: if higher priority tasks are released during their processing, only the scheduler is notified (point 6 in Figure 6b).

VI. Evaluation

This section focus on comparing the original *Domain 0* kernel and the RTCA. As we discussed in Section V, the RTCA can be configured with different batch sizes, which we consider here. We first repeated the experiments in Section IV-A to see the combined effect of the VMM scheduler and *Domain 0* kernel. After that, we focused on *Domain 0* only and showed the latency and throughput performance under four levels of interference workload. Finally, we used an end-to-end task set similar to one in [21] to evaluate the combined effect of the VMM scheduler and *Domain 0* on the end-to-end performance of IDC.

A. Interference within the Same Core

We repeated the experiments in Section IV-A with the RTCA using a batch size of 1 (which as later experiments show, gives better latency performance). For brevity and ease of comparison, we plotted the results in Figure 3, where the lines marked by circles show results obtained using the RTCA. A key observation is that the difference between the two dashed lines (and similarly, between the two solid lines) is small. This indicates that when *Domain 0* is not busy, the VMM scheduler plays a more important role, which is to be expected since the RT-Xen scheduler can effectively prevent priority inversion within the same core, and thus the interference from other VCPUs is much less.

***Summary**: When* Domain 0 *is not busy, the VMM scheduler dominates the I/O performance for higher priority domains.*

B. Interference from Other Cores

Our subsequent experiments focus on showing the effect of *Domain 0*. We use *Original* to represent the default communication architecture in contrast to the RTCA in *Domain 0*. These experiments ran on a six core machine, and all the cores were used.

Figure 7 shows the setup, where three of the cores were dedicated to *Domain 0* and the two highest priority domains respectively, so they would always get the CPU when needed, thus emulating the best that a VMM scheduler can do. On each of the remaining three cores, we booted up three interference

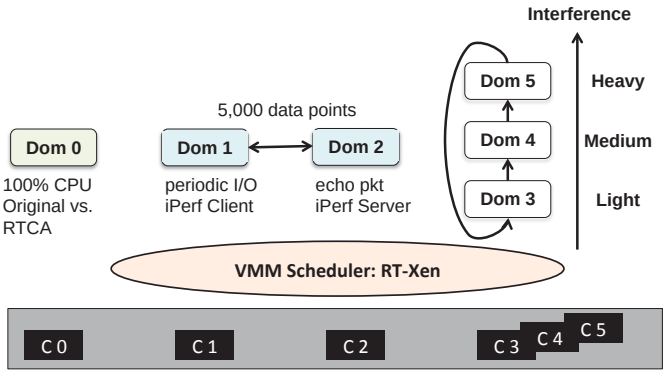


Fig. 7: Experiment with Interference from Other Cores: Setup

domains, and gave each domain 30% of the CPU share. They all performed extensive I/O (sending packets to other domains). The interference is classified according to four levels, with *Base* being no interference, *Light* being only one active domain per core, *Medium* being two active domains per core, and *Heavy* having all three of them active.

As we discussed earlier, the batch size also may affect performance. Therefore, for the *RTCA*, we also examined three batch sizes: 1, as it represents the most responsive *Domain 0*; 64, as this would be the default batch size for the bridge queue; and 238, as this is the maximum batch size for the TX and RX Queues on our hardware. For the *Original* case, we kept everything as defaulted (batch size of 64 per device, total budget of 300 per round).

1) Latency Performance: Similar to the experiments in Section IV-A, the same periodic workload was used to measure the round-trip time between Domain 1 and Domain 2. Table I shows the median, 75%, and 95% values among 5000 data points. All values larger than 1000 μs (1 ms) are bolded for ease of comparison.

From those results, several key observations can be made:

- With the *Original* kernel, when there is even *Light* interference, the latency increases from about 70 μs to more than 5 ms.
- In contrast, the RTCA performs well for soft real-time systems: except with a batch size of 238, 95% of the data points were under 500 μs. This indicates that by prioritizing packets within *Domain 0*, we can greatly reduce the (soft real-time) latency.
- The smaller the batch size, the better and less varied the results. Using a batch size of 1 results in around 70 μs round trip time for all cases; with a batch size of 64, the latency grew to around 300 μs when there is interference; and with a batch size of 238 would vary from 2 to 3 ms. This is due to the increasing blocking times in all three queues, as discussed in Section III. As a result, using a batch size of 1 makes the system most responsive.

***Summary**: By reducing priority inversion in* Domain 0*, RTCA can effectively mitigate impacts of low-priority traffic on the latency of high-priority IDC.*

TABLE I: Effect of Interference from Other Cores: Latency (μs)

	Median				75th percentile				95th percentile			
Domain 0	Original	RTCA			Original	RTCA			Original	RTCA		
		1	64	238		1	64	238		1	64	238
Base	68	70	71	71	69	72	72	72	71	74	74	74
Light	**5183**	60	64	64	**5803**	61	115	90	**6610**	66	261	324
Medium	**9621**	61	216	**2421**	**9780**	63	272	**2552**	**11954**	68	363	**3404**
Heavy	**9872**	69	317	**3661**	**10095**	71	347	**4427**	**11085**	76	390	**4643**

2) Throughput Performance: The previous experiment shows that using a batch size of 1 results in the best latency. However, a smaller batch size also means more frequent context switches, resulting in larger overhead and potentially reduced throughput. This experiment measures throughput under the same settings.

We kept the interference workload as before, and used iperf [22] (which is widely used in networking) in Domain 1 and Domain 2 to measure the throughput. Domain 2 ran the iperf server, while Domain 1 ran the iperf client using a default configuration, for 10 seconds. For each data point, the experiments were repeated 10 times, and we plotted the mean value with confidence intervals (one standard deviation). For completeness, results using the original kernel are also included.

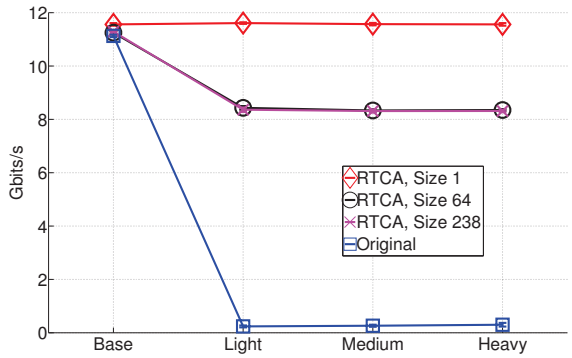


Fig. 8: Interference form Other Cores: Throughput

Figure 8 shows the results. As expected, under the *Base* case, the original kernel and the RTCA performed about the same at 11.5 Gb/s. When there was interference, the throughput with the original kernel dropped dramatically to less than 1 Gb/s due to priority inversions in *Domain 0*. The RTCA with size 1 provided constant performance, since 1 is already the minimum batch size we can have. The blocking time stays relatively constant regardless of the interference level. This also indicates that in IDC, the context switching time is almost negligible. The size 64 and size 238 curves overlap with each other, and all performed at about 8.3 Gb/s under interference. This is to be expected as a larger batch size enables lower priority domains to occupy more time in *Domain 0*, making the blocking time for the non-preemptable sections much longer.

Summary*: A small batch size leads to significant reduction in high-priority IDC latency and improved throughput under interfering traffic.*

C. End-to-End Task Performance

Our previous experiments used I/O micro benchmarks to evaluate both the original kernel and the RTCA in terms of latency and throughput. However, in typical soft real-time systems, a domain may run a mixed task set containing both CPU and I/O intensive tasks, and other domains may compete for CPU resources as well as I/O resources. Our previous work [4], [5] showed that by using RT-Xen schedulers, we can guarantee sufficient availability of CPU resources. This section studies the combined effects of the VMM schedulers and the *Domain 0* communication architecture on end-to-end tasks.

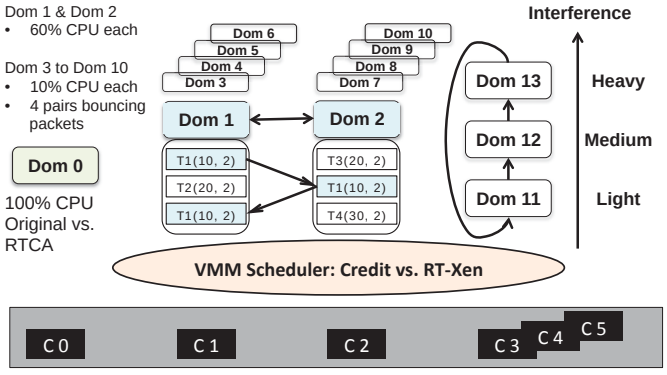


Fig. 9: Experiment with End-to-End Tasks: Setup

Figure 9 shows the setup. *Domain 0* runs on a separate core and is always idle. Domain 1 and Domain 2 are given highest priority and are pinned to cores 1 and 2, respectively, each with 60% of the CPU share. A end-to-end task set similar to [21] ran on them, where task 1 initiated a job every 10 ms, and each job ran for 2 ms and sent a packet to another task in Domain 2. Once Domain 2 received that packet, the task did another computation for 2 ms and sent a packet back to Domain 1. The receiving task in Domain 1 did another 2 ms computation and then finished. This end-to-end task model represents typical distributed tasks in a real-time system, e.g., where a sensor senses the environments, does some computation to compress the data, and sends it back to the server. The server records the data and sends the corresponding command to the sensor, and the sensor may do some computation (for example, adjusting the sampling frequency). Domain 1 also contains a CPU intensive periodic task, and Domain 2 contains two of them. Interested readers

can refer to our previous papers on RT-Xen [4], [5] for task implementation details. For interference within the same core, we booted another eight domains grouped into pairs to bounce packets between each other. They were given 10% CPU share each and configured with lower priority. On the remaining three cores, a similar setup to that in Section VI-B was used to generate IDC interference from other cores. For the RTCA, since the results given in Section VI-B already showed that using a batch size of 1 resulted in the best performance, we did not try other batch sizes. Each experiment ran for 10 seconds.

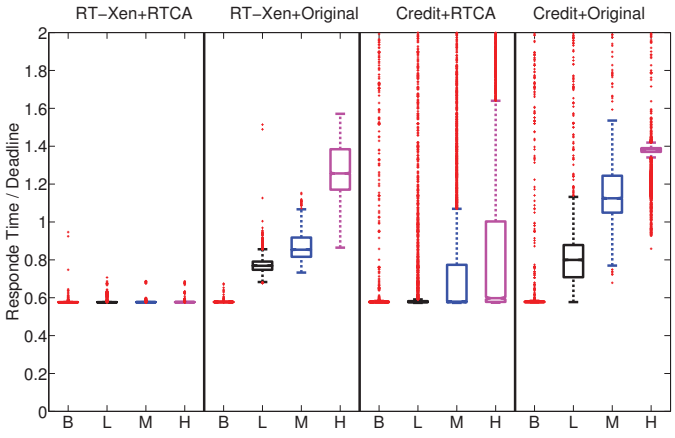


Fig. 10: Box Plot of $\frac{ResponseTime}{Deadline}$ for Task 1

Figure 10 shows a box plot of the ratio $(ResponseTime)/(Deadline)$ for task 1 under different interference levels, with B indicating the *Base* case, L the *Light* case, M the *Medium* case, and H the *Heavy* case. On each box, the central mark represents the median value, whereas the upper and lower box edges show the 25th and 75th percentiles separately. If the data values are larger than $q_3 + 1.5 * (q_3 - q_1)$ or smaller than $q_1 - 1.5 * (q_3 - q_1)$ (where q_3 and q_1 are the 75th and 25th percentiles, respectively), they are considered outliers and plotted via individual markers. For clarity of presentation, any job whose $(ResponseTime)/(Deadline)$ is greater than 2 is not shown here (note here that if $(ResponseTime)/(Deadline)$ is larger than 1, it means the job has missed its deadline).

Starting from the left, the "RT-Xen+RTCA" combination delivers constant performance under all cases. This shows that by combining the two improved subsystems, we can prevent interference from both the same core and other cores. The "RT-Xen+Original" combination misses deadlines under heavy interference. The results confirm that when I/O is involved, *Domain 0* cannot be simply treated as a black box due to the possible priority inversion within it. The "Credit+RTCA" combination performs slightly better than the second combination, but still has lots of outliers even under the *Base* case. This is due to the BOOST contention from Domain 3 through Domain 10. The "Credit+Original" combination performs the worst, as the interference comes from all Domains.

We are also interested in the periodic CPU intensive tasks in Domain 1 and Domain 2, since they could represent important tasks as well. Therefore, we also studied CPU intensive tasks' performance. Our previous paper [4] has shown that the Credit scheduler is not suitable for the real-time CPU intensive tasks, and here we see the same observation. When the Credit scheduler is used, the deadline miss ratio for tasks 2, 3, and 4 all exceed 95% regardless of the *Domain 0* kernel, while when the RT-Xen scheduler is used, no deadline is missed. This is expected as the interference in *Domain 0* make little difference for CPU intensive tasks. Table **??** shows the results.

***Summary**: By combining the RT-Xen VMM scheduler and the RTCA* Domain 0 *kernel, we can deliver real-time performance to both CPU and I/O intensive tasks.*

VII. Related Work

The order in which packets are processed in the *manager domain* has rarely been discussed before. Most prior work treats the manager domain as a black box and focuses on improving the default VMM scheduler [10]–[13], [23]. To our knowledge, this work is the first to detail and modify the exact packet processing order in *Domain 0*.

Another branch of related work concentrates on establishing a shared memory data channel between two guest domains via the Xen hypervisor. As a result, the time spent in the *manager domain* is avoided, and the performance is improved close to the level of native socket communication. The most typical approaches include [6]–[9]. IVC [6] provides a user level communication library for High Performance Computing applications. XWay [7] modifies the AF_NET network protocol stack to dynamically switch between TCP/IP (using the original kernel) and their XWay channel (using shared memory). XenSocket [8] instead maintains another new AF_XEN network protocol stack to support IDC. For security reasons, only one data channel is provided (only a *trusted* domain can read data directly from an *untrusted* domain). XenLoop [9] utilizes the existing Linux netfilter mechanism to add a *XenLoop* module between the IP layer and the netfront driver: if the packet is for IDC, the *XenLoop* module directly communicates with another *XenLoop* module in the corresponding domain. A *discover* kernel module in the *manager domain* is also needed to establish the data channel.

In sharp contrast, the RTCA does not require any information about the guest domain or the application. In principle, as long as the guest domain is supported by Xen, real-time properties can be enforced, but the approaches above are constrained by using Linux as a guest domain. Furthermore, the RTCA naturally supports live migration and can be easily integrated with existing or improved rate control mechanisms in Xen. Finally, the RTCA is a more general approach that can be extended to other devices like the NIC, and also to other similar VMM architectures [24].

Our previous work on RT-Xen [4], [5] provides a real-time VMM scheduling framework within Xen. It provides a suite of real-time schedulers (e.g., deferrable and periodic servers). However, all of its work is done in the hypervisor. In contrast, the RTCA reduces priority inversion within *Domain 0*.

TABLE II: Deadline Miss Ratio fro Periodic CPU Task

Kernel	Original								RTCA (batch size 1)							
Scheduler	Credit				RT-Xen				Credit				RT-Xen			
Interference	B	L	M	H	B	L	M	H	B	L	M	H	B	L	M	H
Task 2	98.38%	99.68%	98.34%	99.6%	0	0	0.02%	0	99.22%	98.62%	99.9%	99.7%	0	0	0	0
Task 3	99.3%	98.9%	98.84%	98.6%	0	0	0	0	98.86%	99.4%	99.54%	98.16%	0	0	0	0
Task 4	99.28%	98.89%	98.83%	98.59%	0	0	0	0	98.86%	99.4%	99.52%	98.14%	0	0	0	0

There also has been research on real-time virtualization in other frameworks besides Xen. Bruns et al. [25] compare the thread switching times and interrupt latencies using the L4/Fiasco microkernel. Danish et al. [26] describe the scheduling framework for the *Quest* OS, which uses a priority-inheritance bandwidth-preserving server policy for communication management. Parmer et al. [27] provide hierarchical resource management (HRM) to customize different subsystems of various applications. Cucinotta et al. [28] focus on scheduling in the KVM, which is an integrated VMM.

Prior research also has focused on providing real-time communication guarantees to local and distributed system tasks. Rajkumar et al. [29] designed a resource kernel which provides timely, guaranteed and protected access to system resources. Ghosh et al. [30] later extended it to maximize overall system utility and satisfy multi-resource constraints. Kuhns et al. [31] provided a real-time communication subsystem to support end-to-end, prioritized traffic and bounded communication utilization of each priority class using middleware. The RTCA complements prior work in that it focuses on IDC within the same host, and reduces key sources of priority inversion in the manager domain.

VIII. Conclusion

As computer hardware becomes increasingly powerful, there is a growing trend towards integrating complex, legacy real-time embedded systems using fewer hosts. Virtualization has received significant attention as an attractive systems technology to support integration of embedded systems of systems. This paper addresses the open problem of supporting *local inter-domain communication* (IDC) within the same host. It examines the real-time IDC performance of Xen, a widely used open-source virtual machine monitor that has recently been extended to support real-time domain scheduling. We show experimentally that improving the VMM scheduler alone cannot guarantee real-time performance of IDC, and to address that problem we have designed and implemented a *Real-Time Communication Architecture* (RTCA) within the manager domain that handles communication between guest domains in Xen. Empirical results demonstrate that combining the RTCA and a real-time VMM scheduler can reduce the latency of high-priority IDC significantly in the presence of heavy low-priority traffic by effectively mitigating priority inversion within the manager domain.

References

[1] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending Networking into the Virtualization Layer," *HotNets*, 2009.

[2] B. Brandenburg and J. Anderson, "On the Implementation of Global Real-time Schedulers," in *Real-Time Systems Symposium (RTSS)*, 2009.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Symposium on Operating Systems Principles (SOSP)*, 2003.

[4] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: Towards Real-time Hypervisor Scheduling in Xen," in *International Conference on Embedded Software (EMSOFT)*, 2011.

[5] J. Lee, S. Xi, S. Chen, L. T. Phan, C. Gill, I. Lee, C. Lu, and O. Sokolsky, "Realizing Compositional Scheduling through Virtualization," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2012.

[6] W. Huang, M. Koop, Q. Gao, and D. Panda, "Virtual Machine Aware Communication Libraries for High Performance Computing," in *Supercomputing*, 2007.

[7] K. Kim, C. Kim, S. Jung, H. Shin, and J. Kim, "Inter-domain Socket Communications Supporting High Performance and Full Binary Compatibility on Xen," in *Virtual Execution environments (VEE)*, 2008.

[8] X. Zhang, S. McIntosh, P. Rohatgi, and J. Griffin, "Xensocket: A High-Throughput Interdomain Transport for Virtual Machines," *Middleware*, 2007.

[9] J. Wang, K. Wright, and K. Gopalan, "XenLoop: A Transparent High Performance Inter-VM Network Loopback," in *High Performance Distributed Computing (HPDC)*, 2008.

[10] S. Govindan, A. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and Co.: Communication-aware CPU Scheduling for Consolidated Xen-based Hosting Platforms," in *Virtual Execution environments (VEE)*, 2007.

[11] D. Ongaro, A. Cox, and S. Rixner, "Scheduling I/O in Virtual Machine Monitors," in *Virtual Execution environments (VEE)*, 2008.

[12] M. Lee, A. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting Soft Real-Time Tasks in the Xen Hypervisor," in *Virtual Execution environments (VEE)*, 2010.

[13] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing Performance Isolation Across Virtual Machines in Xen," in *Middleware*, 2006.

[14] L. Cheng, C. Wang, and S. Di, "Defeating Network Jitter for Virtual Machines," in *Utility and Cloud Computing (UCC)*, 2011.

[15] Xen Wiki, "Credit-based cpu scheduler," http://wiki.xensource.com/xenwiki/CreditScheduler.

[16] K. Ramakrishnan, "Performance Considerations in Designing Network Interfaces," *Selected Areas in Communications*, 1993.

[17] J. Salim, R. Olsson, and A. Kuznetsov, "Beyond Softnet," in *Proceedings of the 5th Annual Linux Showcase and Conference*, 2001.

[18] "Xen Configuration File Options," http://www.xen.org/files/Support/XenConfigurationDetails.pdf.

[19] K. Salah and A. Qahtan, "Implementation and Experimental Performance Evaluation of a Hybrid Interrupt-handling Scheme," *Computer Communications*, 2009.

[20] Xen Wiki, "Xen common problems," http://wiki.xen.org/wiki/Xen_Common_Problems.

[21] J. Liu, *Real-Time Systems*. Prentice Hall PTR, 2000.

[22] C. Hsu and U. Kremer, "IPERF: A Framework for Automatic Construction of Performance Prediction Models," in *Workshop on Profile and Feedback-Directed Compilation (PFDC)*, 1998.

[23] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee, "Task-aware Virtual Machine Scheduling for I/O Performance," in *Virtual Execution environments (VEE)*, 2009.

[24] L. Xia, Z. Cui, J. Lange, Y. Tang, P. Dinda, and P. Bridges, "VNet/P: Bridging the Cloud and High Performance Computing through Fast Overlay Networking," in *High-Performance Parallel and Distributed Computing (HPDC)*, 2012.

[25] F. Bruns, S. Traboulsi, D. Szczesny, E. Gonzalez, Y. Xu, and A. Bilgic, "An Evaluation of Microkernel-Based Virtualization for Embedded Real-Time Systems," in *Euromicro Technical Committee on Real-Time Systems (ECRTS)*, 2010.

[26] M. Danish, Y. Li, and R. West, "Virtual-CPU Scheduling in the Quest Operating System," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2011.

[27] G. Parmer and R. West, "Hires: A System for Predictable Hierarchical Resource Management," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2011.

[28] T. Cucinotta, G. Anastasi, and L. Abeni, "Respecting Temporal Constraints in Virtualised Services," in *COMPSAC*, 2009.

[29] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, "Resource Kernels: A Resource-centric Approach to Real-time and Multimedia Systems," *Readings in Multimedia Computing and Networking*, 2001.

[30] S. Ghosh, J. Hansen, R. Rajkumar, and J. Lehoczky, "Integrated Resource Management and Scheduling with Multi-resource Constraints," in *Real-Time Systems Symposium (RTSS)*, 2004.

[31] F. Kuhns, D. Schmidt, and D. Levine, "The Design and Performance of a Real-Time I/O Subsystem," in *Real-Time Systems Symposium (RTSS)*, 1999.