Prioritizing 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 QoS-critical systems as virtual machines (domains) on a common, virtualized computing platform. Given the lower latency of local inter-domain communication (IDC) on the same host (compared to inter-host communication), system administrators may preferably colocate domains so that they can communicate locally. When multiple IDC flows contend on the same host, it is important to properly prioritize IDC flows among domains to meet their respective QoS requirements. This paper examines the limitations of IDC 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 IDC QoS under different conditions, and show that improving the VMM scheduler alone cannot effectively prevent priority inversion for local IDC. To address those limitations, we present RTCA, a *Real-Time Communication Architecture* within the manager domain in Xen, along with experimental results that demonstrate the latency of high-priority IDC 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 40 to 60 virtual machines (VMs) 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 systems on a common virtualized computing platform. Since VMs are called domains in virtualization, we will use the term domain from now on. When systems are integrated in this manner, a significant amount of network communication may become local inter-domain communication (IDC) within the same host. However, when multiple domains co-exist on a same host, it is important to properly schedule the processing of local communication to achieve latency differentiation among domains with different priorities and QoS requirements.

This paper addresses the problem of achieving latency differentiation in local IDC on a virtualized platform. We closely examine the latency of IDC flows in Xen [3], a widely used open-source virtual machine monitor that has been extended to support real-time domain scheduling [4], [5], and point out its key limitations that can cause significant priority inversion in IDC. We show experimentally that improving the VMM scheduler alone cannot achieve suitable latency differentiation for IDC due to significant priority inversion in the manager domain that handles packets sent between domains. To address that problem we have designed and implemented a *Real-Time Communication Architecture* (RTCA) within the manager domain that effectively handles local IDC within a host.

A key observation from our empirical studies is that both the VMM scheduler and the manager domain can affect IDC latency and incur priority inversion. While the former needs to schedule the right domain at the right time to send or receive packets, the latter should provide proper prioritization in processing packets from domains efficiently at run time. In this paper, we systematically study the impact of both the VMM scheduler and the manager domain, 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 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 appropriately prioritized 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.

- Identification of the key limitations of Xen for handling IDC flows with different priorities in both the manager domain and the VMM scheduler through a systematic experimental study.
- Design and implementation of RTCA, a *Real-Time Communication Architecture* that effectively prioritizes packet processing within the manager domain.
- Experimental results that show the combination of the RTCA and an existing real-time VMM scheduler can reduce the latency of high-priority IDC from ms to μs in the presence of heavy low priority IDC flows.

II. Background

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

978-1-4799-0590-4/13/$31.00 ©2013 IEEE

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. 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.

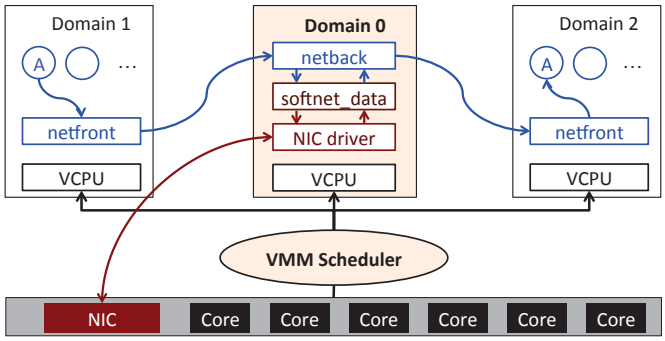


Fig. 1: Xen Communication Architecture Overview

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 Xen supported operating systems. 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 that 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 packets to the netfront driver in Domain 1; 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; when Domain 2 gets scheduled, its netfront driver 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 modifications to the guest domain besides the well supported Xen patch, and may even need to modify the applications 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 mechanisms 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 packets according to their QoS requirements. 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. Also, 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. Domains' VCPUs are divided into 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 the VCPU is placed into the UNDER category, while UNDER and OVER are scheduled in a round-robin manner. Ongaro et al. [11] studied 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 CPU and I/O interference, and on the order in which the domains were booted.

C. RT-Xen Scheduler

Our previous work, RT-Xen [4], [5], allows users to prioritize the CPU resources for domains, and thus delivers the desired QoS for CPU-intensive tasks running atop each domain. For the studies presented in this paper, we use the deferrable server scheduler: whenever the domain still has budget but no tasks to run, the remaining budget is preserved for future use. With that policy in the RT-Xen scheduler, if a domain with pending packets to send has the highest priority and also has budget left, it will be scheduled first. For incoming packets, the VMM scheduler is notified (via the *wake_up()* function), and it compares the destination domain's priority with the currently running one: if the destination domain has higher priority, it will immediately interrupt the current domain and be scheduled instead; otherwise it will be inserted into the run queue according to its priority. In Section VI-A, we show that applying RT-Xen can greatly improve IDC prioritization when compared to the default Credit scheduler. However, when interfering IDC flows come from other cores, RT-Xen alone cannot provide correctly prioritized IDC performance.

D. IDC in Domain 0

To explain how IDC is performed in *Domain 0*, we now describe how Linux processes packets, how the softirqs and kernel threads behave, 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 *Domain 0*. 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 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.

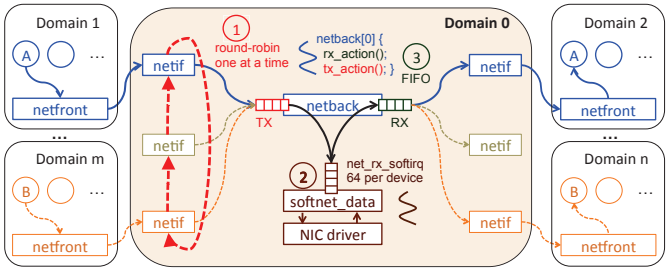


Fig. 2: Xen Communication Architecture in Domain 0

As shown in Figure 1, Xen uses the netfront and netback drivers to transmit packets between guest and manager domains. Figure 2 illustrates 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 Linux 3.4. 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 the netback device. [1]

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 quantum 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 server-based algorithm [19] 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. RTCA 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 quantum 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 a packet, if *Domain 0* finds that its destination is a local domain, it bridges the packet to the RX Queue in the corresponding netback device; if it is processing the first packet, it also notifies the scheduler to schedule the kernel thread. Note that a 1000 packet limit applies for the backlog device [20]. 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 a function 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 processing of 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* can impact IDC performance. This section describes qualitatively the limitations of both the VMM scheduler and *Domain 0* for prioritized IDC. The next section will present an empirical study to quantify the impacts of their limitations on performance of prioritized IDC flows.

A. Limitations of the VMM schedulers

The default Credit scheduler has two major problems when handling prioritized traffic: (1) it schedules VCPUs with outgoing packets in a round-robin fashion; and (2) for incoming packets, it applies a general boost to a blocked VCPU regardless of its priority. Note that boosting the priority of a low priority VCPU to receive a packet can introduce priority inversion when a high priority VCPU is running.

[1]Linux (version 2.4 and after) provides a traffic control tool [18]. A priority qdisc can be used to prioritize packets. However, the priority qdisc only works within one device, while in the IDC, traffic belonging to different domains already delivered to different *netif* devices.

The RT-Xen scheduler [4], [5] applies a strict priority policy for VCPUs for both outgoing and incoming packets, and thus can 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 to another local domain is less than 10 μs. Consider a case where one packet is bouncing between two domains on the same core: if these two domains run 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 consumed, resulting in a 50% share for each regardless of their budget and period configuration. This clearly violates the resource isolation property of the VMM scheduler. We address this limitation by providing dual time resolutions: μs for CPU time accounting, and ms for VCPU scheduling. The dual resolution provides better resource isolation, while maintaining 1ms as an appropriate scheduling quantum for real-time applications. For all the evaluations in this paper we use this improved version of the RT-Xen scheduler.

B. Limitations of Domain 0

Domain 0 also has the following major limitations in terms of real-time IDC performance. As was discussed in Section II-D, the TX, softnet_data, and RX queues are shared by all guest domains, resulting in a round-robin scheduling policy with a quantum of 1 regardless of the domain's priority. We show in Section VI that even under a light interference workload from other cores (which cannot be prevented by any VMM scheduler), the IDC latency for high priority domains is severely affected. Another limitation is that the TX, softnet_data, and RX queues are processed in a fixed order regardless of the priority of the current processing packets. 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 3.0 (and later versions) switched to using one kernel thread to process both TX and RX queues, with the RX Queue always being processed first. This introduces another problem that the higher priority packets may need to wait until a previous lower priority one has finished transmission. Finally, the priority inversion is exacerbated by large and mismatched queue sizes. The TX and RX queues have total processing sizes of 238 with a quantum of 1 for each domain, while the softnet_data queue has a total processing size of 300 with a quantum of 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.

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

We ran a series of experiments to evaluate the impacts of the VMM scheduler and *Domain 0* on IDC performance. 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. 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 [21]. During all experiments we disabled the NIC and configured all the guest domains within a local IP address, focusing on IDC only. We also shut down all other unnecessary services to minimize incidental sources of interference. *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 ms and a period of 10 ms 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. 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 constantly bounced a packet between each pair. Note that all 10 domains were doing IDC 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 already had a long backlog with eight interfering domains, thus creating a priority inversion. In contrast, the RT-Xen scheduler would always schedule domains based on their priorities.

Figure 3 shows a CDF plot of the IDC latency between the pair of high priority domains 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 a modified *Domain 0* with our new *Real-Time Communication Architecture* (RTCA), which will be discussed in Sections V and VI-A. We can clearly see that due to the general boost,

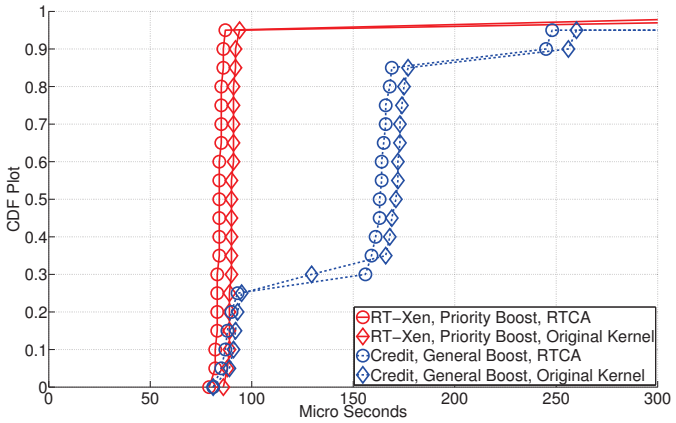


Fig. 3: Effect of the VMM Scheduler: Credit VS. RT-Xen

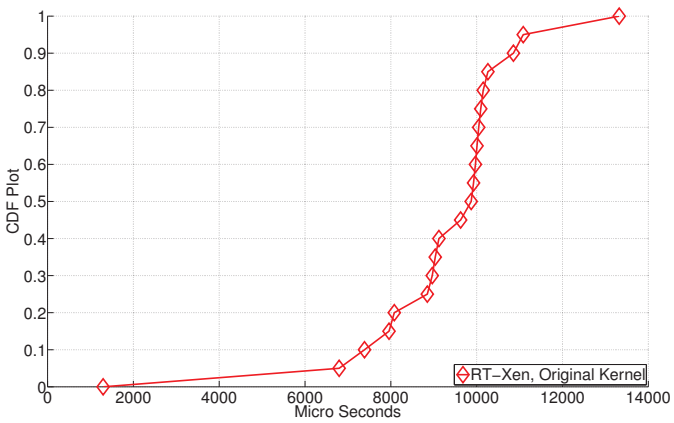


Fig. 4: Bottleneck in Domain 0

the IDC latency between the high priority domains under the Credit scheduler is severely affected by the interfering IDC between low priority domains, growing from around 80 μs to around 160 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 IDC load it was offered.

B. The VMM Scheduler is Not Enough

We have shown that scheduling VCPUs based on priorities can deliver better IDC performance for high priority domains. The experiment presented in this subsection shows that *Domain 0* can also become a bottleneck when processing IDC, especially when significant contention from low priority domains exists.

In this experiment, 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 intensive IDC, creating a heavy load on *Domain 0*.

Figure 4 shows a CDF plot of the results with a sampling point every 5th percentile. Please note the larger x axis range in this figure, compared to Figure 3. The IDC latency between the high priority domains grew from the μs level to more than 6 ms. Since all the interference occurs within *Domain 0*, any improvement to the VMM scheduler thus cannot help. Therefore, it is important to introduce prioritized IDC packet processing in *Domain 0*.

V. Real-Time Communication Architecture

To address the limitations of *Domain 0*, this section presents a new *Real-Time Communication Architecture* (RTCA) for *Domain 0*. The goal of the RTCA is to support packet processing based on domain priorities while reducing priority inversion. The priorities are based on domains instead of flows due to the domain is the unit provided to the end customer.

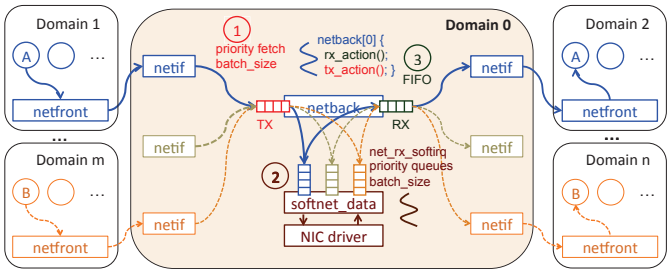
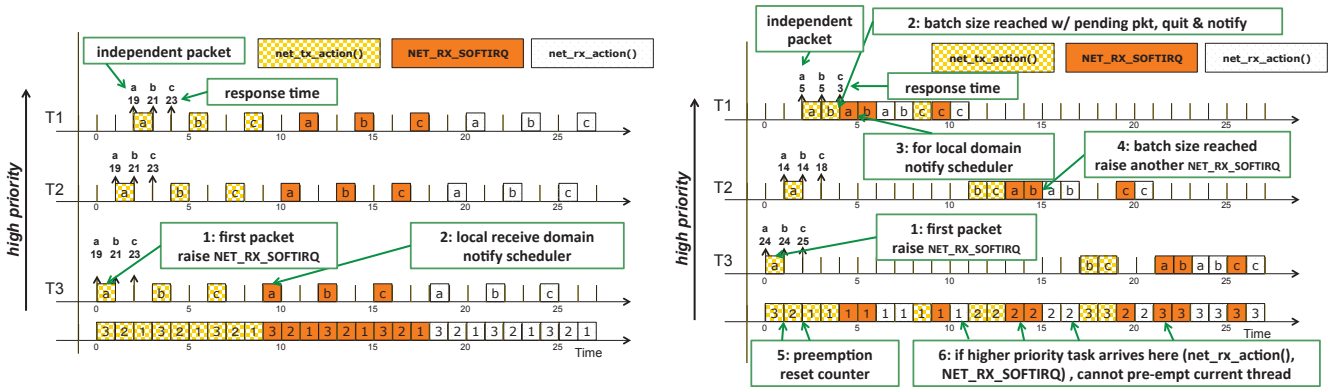


Fig. 5: RTCA: Real-Time Communication Architecture

Figure 5 shows the RTCA in *Domain 0*. We now discuss the changes we made to each of the three queues.

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 priorities, one at a time. Within one round, we fetch up to a *batch size* number of packets, and we also make the batch size tunable to make IDC QoS more configurable for different system integrators. A counter is initialized to 0 and used to keep track of how many packets are processed within one round. 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[2]. As a result, the TX Queue is dequeued in FIFO order. However, whenever a higher priority domain arrives in one round, we reset the counter so that the performance for higher priority domains will not be affected. Section VI shows that with a batch size of 1, the system achieves suitable IDC latency and throughput for high priority domains. This is caused by the longer blocking time for each packet when the batch size increases. If a batch size of 1 is used, the total size limit of 238 is unlikely to be reached,

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



(a) Original Kernel

(b) RTCA Kernel

Fig. 6: Packet Processing Illustration

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
```

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 can 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-preemptable: 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.

Notably, without changing the fundamental architecture of *Domain 0*, we keep the benefits of compatibility with the original Xen features (for example, the existing rate control mechanism can be seamlessly integrated with RTCA), while significantly improving the IDC latency between high priority domains (as shown in Section VI) by an order of magnitude, resulting in μs level timing that is suitable for many soft real-time systems. Therefore, while the RTCA does not completely eliminate priority inversion, it can be highly effective in improving IDC prioritization for soft real-time applications.

Examples for Packet Processing

To better illustrate how RTCA 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 need it (e.g., via 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

TABLE I: Effect of Interference from Multiple 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**

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:

- RTCA effectively reduces the IDC latency of the packets between the higher priority domains (from 19, 21, 23 to 5, 5, 3, respectively). Since (unmodified) Xen processes packets in a round-robin order, and uses a relatively large batch size 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 faster IDC 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 focuses 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 address 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 under four levels of interference workload. Finally, we used an end-to-end task set to evaluate the combined effect of the VMM scheduler and *Domain 0* on the end-to-end performance of IDC. All the experiments used the same setup as in Section IV.

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 IDC performance for higher priority domains.*

B. Interference from Multiple Cores

The subsequent experiments focus on showing the effect of *Domain 0* when it becomes the performance bottleneck. We use *Original* to represent the default communication architecture in contrast to the RTCA in *Domain 0*.

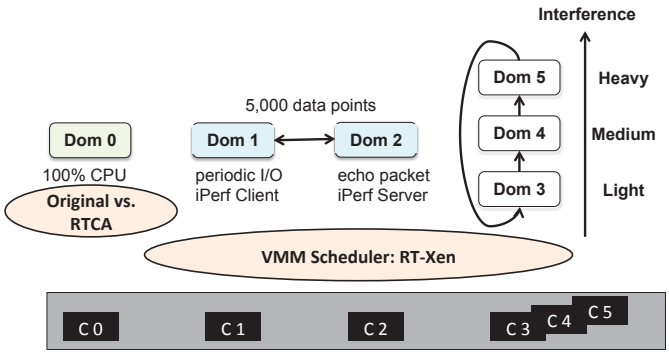


Fig. 7: Experiment with Interference from Multiple Cores

Figure 7 shows the setup, with three cores 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 domains, and gave each domain 30% of the CPU share. They all performed intensive IDC (constantly sending UDP packets to other domains). Interference was generated at four levels in different experiments, 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 can affect the performance of RTCA. Therefore, we examined three batch sizes: 1, as it represents the most responsive *Domain 0*; 64, as this would be the default batch size for the softnet_data 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 (process 64 packets per device per time, and 300 packets per round).

1) Latency: Similar to the experiments in Section IV-A, the same periodic workload was used to measure the round-trip time between the high priority domains, 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, even under *Light* interference, the latency increases from about 70 μs to over 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 IDC latency between high priority domains under interfering IDC between low priority domains.
- 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 times for all cases; with a batch size of 64, the latency grew to around 300 μs under interference; and with a batch size of 238 would reach to above 3 ms. This is due to the increasing blocking times caused by priority inversion in all three queues, as discussed in Section III. As a result, using a batch size of 1 makes the system most responsive to high priority IDC.

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

2) Throughput: 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 in Section VI-B1, and used iperf [22] (which is widely used in networking evaluations) 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. For completeness, results using the original kernel are also included.

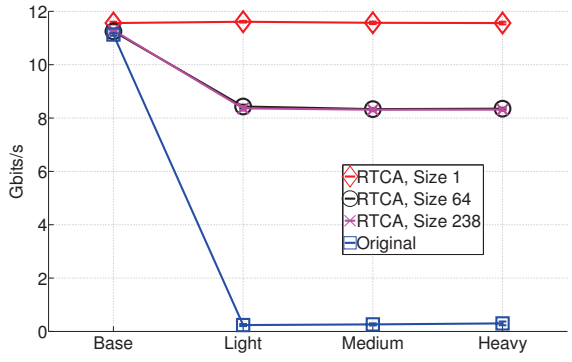


Fig. 8: Interference from Multiple Cores: Throughput

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 of high priority IDC with the original kernel dropped dramatically to less than 1 Gb/s due to priority inversions in *Domain 0*. The RTCA with batch size 1 provided steady performance as the blocking time due to priority inversion stays relatively constant regardless of the interference level. This also indicates that in local IDC, the context switching time is insignificant. 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 consume more time in *Domain 0*, making high priority IDC performance worse.

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

C. End-to-End Task Performance

The previous experiments used micro benchmarks to evaluate both the original *Domain 0* and the RTCA in terms of latency and throughput. However, in typical soft real-time systems, a domain performs both computation and communication workloads. This section studies the combined effects of the VMM schedulers and the *Domain 0* communication architecture on end-to-end tasks.

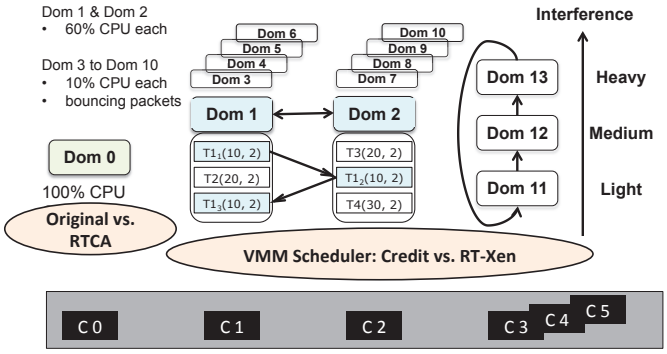


Fig. 9: Experiment with End-to-End Tasks

Figure 9 shows the setup. *Domain 0* runs on a dedicated core. Domain 1 and Domain 2 are given the highest priority and are pinned to cores 1 and 2, respectively, each with 60% of the CPU share. Task $T1$ is an end-to-end task consisting of three subtasks, $T1_1$ and $T1_3$ in Domain 1 and $T1_2$ in Domain 2. A new instance of $T1$ is released every 10 ms. For each instance, $T1_1$ first ran for 2 ms and sent a packet to $T1_2$ in Domain 2. Once Domain 2 received that packet, $T1_2$ ran for 2 ms and sent a packet back to Domain 1. $T1_3$ received the packet and ran for 2 ms and completed the instance of end-to-end task. Domain 1 also contains a local periodic task $T2$, and Domain 2 contains two local periodic tasks, $T3$ and $T4$. To simulate interference within the same core, we booted four pairs of other domains with each pair bouncing packets between each other. Each of the eight interfering domains was

given 10% CPU share and assigned a lower priority. On the remaining three cores, a similar setup to that in Section VI-B was used to generate IDC interference from multiple 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.

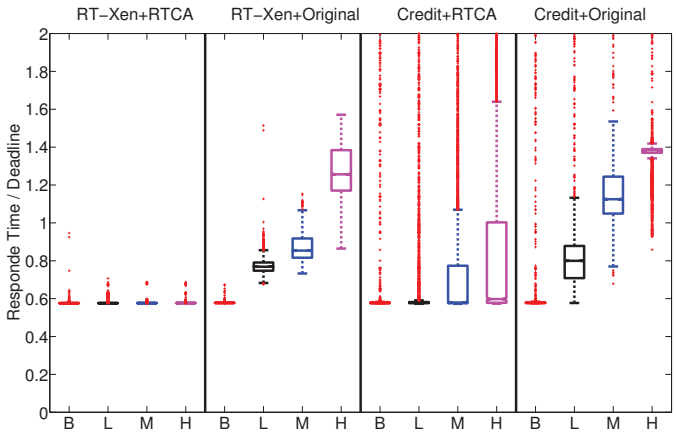


Fig. 10: Box Plot of Normalized Latency for Task T1

We use a metric called the normalized latency, defined as the ratio between the response time of a task to its deadline. A task meets its deadline if its normalized latency is within 1; otherwise it misses its deadline. Figure 10 shows a box plot of the normalized latency for then end-to-end task $T1$ 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 normalized latency is greater than 2 is not shown here (note here as well that if normalized latency is larger than 1, it means the job has missed its deadline).

Starting from the left, the "RT-Xen+RTCA" combination consistently meets its deadline, with the median normalized latency below 0.6. This shows that by combining the two improved subsystems, we can effectively alleviate interference both from the same core and from other cores. The "RT-Xen+Original" combination misses deadlines under heavy interference. The results confirm that when IDC is involved, *Domain 0* cannot be simply treated as a black box due to the possible priority inversion within its communication subsystem. The "Credit+RTCA" combination performs slightly better than the second combination, but still incurs a large number of deadline misses (denoted as outliers in Figure 10) 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 T1 suffers interference from all the other domains.

***Summary**: By combining the RT-Xen VMM scheduler and the RTCA* Domain 0 *kernel, we can deliver end-to-end real-time performance to tasks involving both computation and communication even under interference from low-priority IDC flows.*

VII. Related Work

This work presents a comprehensive study of IDC prioritization by studying both the VMM scheduler and the *Domain 0* kernel in Xen. Related work is thus largely divided into two categories. For the VMM scheduler, prior work focuses on improving the default Credit or SEDF schedulers [10]–[13], [23], and treats *Domain 0* as a black box. 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. 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 they are still scheduled in the default credit queue. Lee et al. [12] patched the Credit scheduler so it will boost not only blocked VCPUs, but also active VCPUs (so that if a VCPU also runs a background CPU intensive task, it can benefit from the boost as well). However, none of these approaches strictly prioritizes VCPUs. When multiple domains are doing I/O together, they are all scheduled in a round-robin fashion. To our knowledge, the work presented in this paper is the first to detail the communication architecture within *Domain 0*, and to evaluate the combined effect for IDC using different combinations of both the VMM scheduler and *Domain 0* under different cases.

Other research [6]–[9] concentrates on establishing a shared memory data channel between two guest domains via the Xen VMM. As a result, the time spent in the *manager domain* is avoided, and the performance is improved close to that of native socket communication. IVC [6] provides a user level communication library for High Performance Computing applications. XWay [7] modifies the AF_NET network protocol stack to switch dynamically 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. 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. These approaches [6]–[9] all require modification to the guest domain (beyond the well-supported default Xen patch) and/or the applications running atop it [6], [8]. In sharp contrast, RTCA does not require any modification to the guest domain or the application or the Xen hypervisor, and can be seamlessly integrated with the rate control mechanism within Xen. In addition, the insights from this work can be extended to the NIC or other similar VMM

architectures [24], making it more general and appropriate for real-time applications.

There also has been research on other virtualization platforms besides Xen. Cucinotta et al. [25] focus on improving the Linux scheduler for KVM [26]. Although this can benefit I/O performance, it cannot entirely prevent interference from other domains. The COMPOSITE OS [27] provides a hierarchical resource management mechanism: when a NIC DMAs data into main memory and triggers an interrupt on the CPU, the device driver classifies the packet and directs it to the corresponding subsystem (similar to guest domains in Xen). However, how to deal with interference is not addressed. The *Quest* OS [28] contains two kinds of VCPUs: Main VCPU and I/O VCPU. The Main VCPU is like a VCPU in Xen to process CPU intensive tasks, while the I/O VCPU is like a device driver in *Domain 0* in Xen: each is created per device and scheduled with a priority-inheritance bandwidth-preserving server policy. The I/O VCPUs are co-scheduled with the Main VCPU, while in this paper, *Domain 0* always owns a separate core. This paper complements related work in that it focuses on the widely used Xen architecture, and focuses on local IDC instead of network I/O. In addition, we provide a comprehensive study by combining the VMM scheduler and *Domain 0*.

VIII. Conclusion

Virtualization has received significant attention as an attractive technology to support integration of QoS-critical systems. This paper addresses the open problem of supporting local *inter-domain communication* (IDC) within the same host. It examines the IDC performance of Xen, a widely used open-source virtual machine monitor that recently has been extended to support real-time domain scheduling. We show through both analysis and experiments that improving the VMM scheduler alone cannot achieve effective latency differentiation for IDC due to significant priority inversion with the manager domain. To address this limitation, we have designed and implemented a *Real-Time Communication Architecture* (RTCA) within the manager domain to achieve effective prioritization among IDC flows. 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] "Linux Advanced Routing and Traffic Control," http://www.lartc.org/.

[19] "Xen Network Configuration," http://xenbits.xen.org/docs/unstable/misc/xl-network-configuration.html.

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

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

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

[26] "KVM: Kernel Based Virtual Machines," http://www.linux-kvm.org.

[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] 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.