

# Achieving 10 Gb/s using Xen Para-virtualized Network Drivers

Kaushik Kumar Ram<sup>§\*</sup>, Jose Renato Santos<sup>‡</sup>, Yoshio Turner<sup>‡</sup>, Alan L. Cox<sup>§</sup>, Scott Rixner<sup>§</sup>  
<sup>‡</sup>Hewlett Packard Laboratories      <sup>§</sup>Rice University

## 1 Introduction

The high processing overheads incurred in Xen Para-virtualized Network Drivers limit overall I/O performance. For example, the number of processing cycles consumed for receiving network packets is 4.7 times higher in Xen than in native Linux. This high cost limits network throughput for Xen guests to only 2.9 Gb/s using a 10 gigabit NIC on a modern server which is able to achieve 9.3 Gb/s when running native Linux.

In a previous Xen summit we described several techniques to reduce the overhead of Xen para-virtualized (PV) network drivers. In particular we argued that the use of multi-queue NICs, (e.g. Intel VMDq) and the reuse of granted I/O buffer pages over multiple I/O operations could significantly reduce the overheads of Xen PV network drivers. Since then we have added support to multi-queue NICs in Xen netback drivers and evaluated its performance improvements when using a 10 Gigabit Intel NIC with VMDq support. In addition we have designed and implemented a novel grant reuse mechanism based on a software I/O translation table which allows grants to be reused over multiple I/O operations and does not require any communication with the driver domain when the guest needs to revoke an active grant.

The use of multi-queue NICs and grant reuse significantly reduce the overhead of driver domains. However, significant I/O virtualization overheads remain in the guest domains. To address these overheads, we also describe and evaluate additional I/O virtualization optimizations for guest domains.

## 2 Multi-queue NICs

Multi-queue NICs have multiple sets of transmit and receive queues. Each queue can be dedicated to a specific guest by assigning it a MAC address associated with the guest and then ensuring that buffers posted to the queues are allocated from memory belonging to that guest. This enables the NIC to place received packets directly into the corresponding guest's memory, avoiding the processing cost of demultiplexing packets in a software bridge and

the cost of copying packets from the driver domain to the guest.

Figure 1 compares the number of processor cycles per packet consumed in the guest domain and in the driver domain for different configurations. The first bar shows the processing cost for original Xen (*Xen Orig*) to process network packets on the receive side of a TCP connection; while the last bar shows the processing cost for *Linux*. Original Xen has significantly higher cost than Linux. Moreover the driver domain has significantly higher cost than the guest itself. The use of multi-queue NICs reduces the processing cost of the driver domain by 69% as illustrated by the second bar (*Xen MQ*) of Figure 1.

## 3 Grant Reuse based on a Software I/O Translation Table

Typically, the driver domain uses grants to map guest buffers in its address space. The need to map and unmap guest domain buffers in the driver domain on each I/O operation has high processing overhead that limits I/O performance. But when using multi-queue NICs, the device can place the packet directly into guest memory avoiding the need for the driver domain to access the network buffer. Thus the guest page does not need to be mapped in the driver domain. However, the driver domain still must verify that the granted page actually belongs to the guest and that it has been pinned by the hypervisor.

We propose an *I/O translation table* mechanism that allows the driver domain to ensure that I/O pages belong to the corresponding guest and have been pinned by the hypervisor without having to map them. An I/O translation table is a region of memory shared between the hypervisor and the the driver domain. It is indexed by the same grant references used to index the grant table and each entry has two fields as illustrated in Figure 2. The (*MA*) field is written by the hypervisor and contains the *machine address* associated with the grant. The *status (ST)* field is written by the driver domain and indicates the page is being used and thus the grant cannot be revoked. When the guest domain grants the driver domain access to a page, the guest updates its grant table and then issues a grant hypercall to add an entry in the I/O translation table. The

\*This work was performed while Kaushik Kumar Ram was an intern at HP Labs.

hypervisor adds the corresponding machine page address to the table after validating and pinning the page. The driver domain can then access the I/O translation table to determine the machine address of a granted page and use this address to program the device DMA directly without the need to issue a hypercall.

Since the guest page is not mapped, the driver domain cannot use netfilter rules to process network packets. In the future we plan to use packet header splitting functionality available in most modern NICs to separate packet headers from payload allowing the use of local driver domain memory for storing packet headers. Packet headers would need to be copied from driver domain to guest memory, but this should not add significant overhead since headers are small.

While the I/O translation table avoids the overhead of grant hypercalls in the driver domain, it adds additional overheads due to grant hypercalls in the guest to pin and unpin pages. However this overhead can be minimized by reusing grants over multiple I/O operations. We modified the socket buffer allocator in Linux to use a dedicated slab cache that create a pool of dedicated network buffers that are recycled over multiple I/O operations. The network buffers are allocated from the pool when needed for I/O operations and returned to it afterwards. A grant is created the first time the buffer is used but not revoked when the buffer is returned to the free pool. The grant is only revoked when the page is removed from the dedicated slab cache which can happen due to kernel memory pressure.

Figure 1 shows the improvement in processing cost due to the I/O translation table and grant reuse mechanisms. In particular, grant reuse reduces the driver domain cost by 53% compared to the multi-queue results, as shown in the third bar (*GRU*) of the Figure.

## 4 Guest Optimizations

In addition to reducing the processing cost in the driver domain we also propose additional optimizations to reduce the cost in the guest domain. First, we extend netfront with LRO (Large Receive Offload) processing which aggregates arriving packets into a much smaller number of larger-sized packets, and then passes the large segments to the network stack. Second, we use software prefetching to bring a packet into the processor cache while we are processing the previous packet. This minimizes the cache miss penalty for accessing packets in netfront. Finally, we reduce the size of buffers used to receive packets. For historical reasons, netfront allocates full page buffers for each data packet. However half page (2KB) buffers are large enough to accommodate standard sized Ethernet packets. By using a smaller buffer size we reduce the TLB working set for packet buffers which reduces the

number of TLB misses and improves performance.

Figure 3 shows that, in combination, these three guest domain optimizations reduce guest domain processing cost by 30%, from 4721 to 3266 processor cycles per packet. This final cost is only 11% higher than the processing cost of Linux (2927 packets per cycle), The remaining gap between Xen and native Linux is caused by two main factors: higher processing cost in the virtual device driver compared to the physical device driver, and additional processor cycles spent in the hypervisor for event delivery between the guest and driver domains.

## 5 Impact of Optimizations

Figure 4 shows the impact of all optimizations on the I/O throughput that is achieved by guest domains. The first bar, Xen Orig, shows that Xen without our optimizations achieves only 2.9 Gb/s throughput to a single guest. The bottleneck resource in this case is the 100% saturated driver domain processor. In comparison, non-virtualized Linux, shown in the last bar, Linux, achieves 9.31 Gb/s using 100% of a single processor. The second bar, Xen Opt (1 Guest), corresponds to Xen with multi-queue support, grant reuse, and the guest domain optimizations. With these optimizations, the achieved throughput increases 180% reaching 8.2 Gb/s for a single processor guest.

By removing the driver domain bottleneck we achieve better scalability and higher data rates when running multiple guests which can more efficiently utilize the multiple cores available in modern systems. The third bar in Figure 4, Xen Opt (2 Guests), shows that full 10 GbE line rates can be achieved when running two guests on different physical processors. In this scenario the bottleneck resource is the link bandwidth.

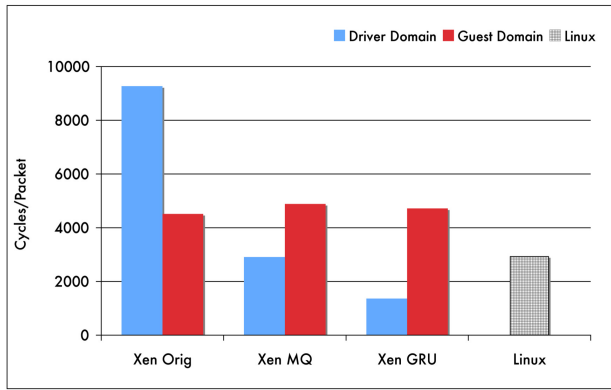


Figure 1: Packet Processing Cost

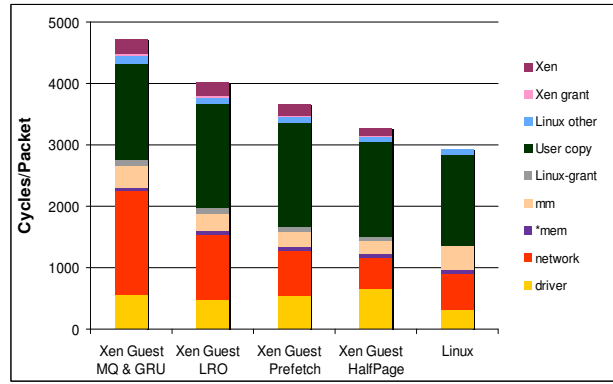


Figure 3: Guest domain cost breakdown in cycles/packet

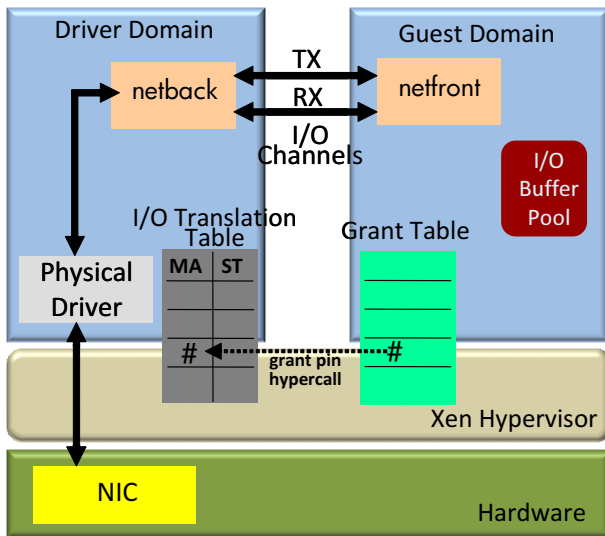


Figure 2: I/O Translation Table

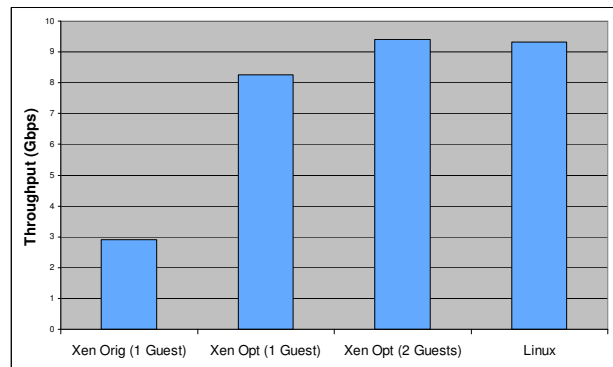


Figure 4: Impact of the optimizations on throughput