Optimizations of Large Receive Offload in Xen

Fumio Nakanjima and Hitoshi Oi*
The University of Aizu, Aizu Wakamatsu, JAPAN
{f.nkjm,oi}@oslab.biz

Abstract

Xen provides us with logically independent computing environments (domains) and I/O devices can be multiplexed so that each domain considers as if it has own instances of I/O devices. These benefits come with the performance overhead and network interface is one of most typical cases. Previously, we ported the large receive offload (LRO) into the physical and virtual network interfaces of Xen and evaluated its effectiveness. In this paper, two optimizations are attempted to further improve the network performance of Xen. First, copying packets at the bridge within the driver domain is eliminated.

The aggregated packets are flushed to the upper layer in the network stack when the kernel polls the network device driver. Our second optimization is to increase the number of aggregated packets by waiting for every other polling before flushing the packets. Compared to the original LRO, the first optimization reduces the packet handling overhead in the driver domain from 13.4 to 13.0 (clock cycles per transferred byte). However, it also increases the overhead in the guest domain from 7.1 to 7.7 and the overall improvement in throughput is negligible. The second optimization reduces the overhead in driver and guest domains from 13.4 to 3.3 and from 7.1 to 5.9, respectively. The receive throughput is improved from 577Mbps to 748Mbps.

Keywords Virtual Machine Monitor, Network, Performance Analysis, Xen

I. Introduction

System-level virtualization technologies provide multiple independent computing environments on a single physical platform. Each computing environment (virtual machine, or domain in Xen) can run its own instance of operating system (guest OS) and resource isolation between virtual machines (VMs) are guaranteed [1]. These advantages of virtualization come with performance overheads. With hardware supports for CPUs, switching between privileged and unprivileged execution modes is relatively fast [2], [3]. However, accesses to I/O devices are still costly operations and networking is one of most typical cases. In Xen, paravirtualization (PV) model is used for virtualizing I/O devices. With the PV device model, multiplexing and protection of a single I/O device among guest domains is possible. In addition, the interface of the I/O device for each guest domain can be abstracted. However, the performance degradation of network interface in the PV model is significant and various optimization attempts have been made [4], [5], [6], [13].

Large receive offload (LRO) is a technique to reduce the overhead of handling received message packet [7], [8]. Previously, we ported the LRO into a Xen virtualized system and reported the results of preliminary performance measurements [9]. In this paper, we apply two optimizations to LRO to improve the receive network performance of Xen. We analyze the experimental results in terms of throughput, number of clock cycles to process packets and also the latency incurred to the LRO.

The rest of this paper is organized as follows. In the next section, we explain the technical background and motivations of this work. In Section III, experimental results and their analysis are presented. Previous work related to this paper is presented in Section IV and the paper is concluded in Section V.

II. Background

In this section, we first present brief introductions to the architecture of Xen internal network and LRO. Next, two optimizations for further performance improvement are described.

A. Xen Internal Network Architecture

Fig. 1 shows the architecture of the Xen internal network. Each guest domain has zero or more virtualized

*Authors are ordered alphabetically

network interface called netfront. In the privileged domain handling physical network interface (driver domain), there is a counterpart for each netfront (netback). Netfront and corresponding netback exchange packets by sharing page frames pointed to by the descriptor ring. For event notification, virtual interrupt mechanism (event channel) through Xen hypervisor is used. This paravirtualized network architecture of Xen has advantages, such as device isolation or transparency to the guest domain, but it incurs high network performance overhead. In [6], it is reported that the per packet CPU overhead (in clock cycles) in Xen network is more than four times larger than that of native Linux.

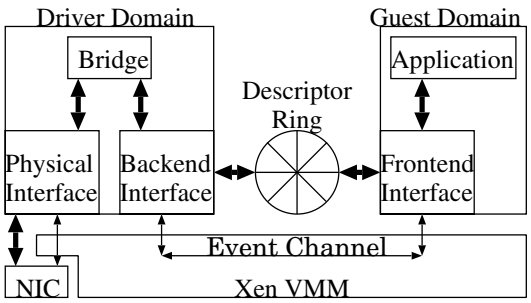


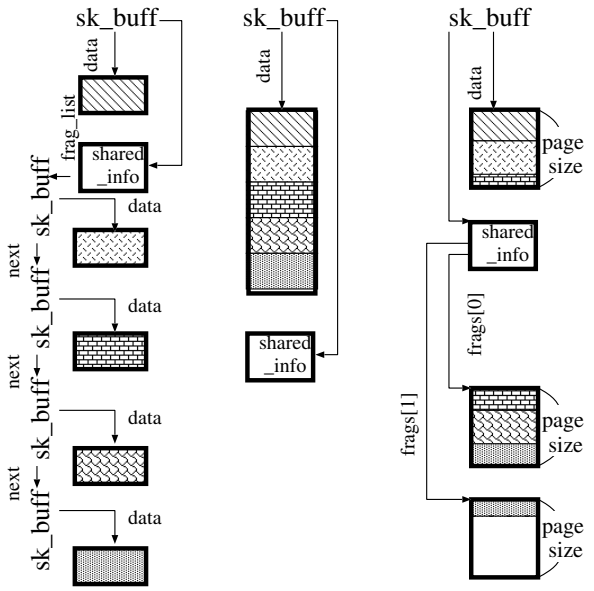
Fig. 1. Xen Internal Network Architecture

B. Large Receive Offload

Large receive offload (LRO) was initially implemented by hardware on the Neterion 10GbE Ethernet card [7]. Jan-Bernd Themann implemented it in software as a patch for the Linux kernel [8] and it has been adopted to the kernel tree from version 2.6.24. LRO combines multiple received TCP packets and passes them as a single larger packet to the upper layer in the network. By reducing the number of packet processing operations, the CPU overhead is lowered and the performance improvement is expected.

C. Further Optimizations

In our previous work, we ported the skb-mode LRO into the physical and virtual NICs of a Xen virtualized system and measured its effectiveness [9], [10]. In the previous work, we noticed of two possible points of further optimization. First, the packets in the same TCP stream are aggregated with a chain of `sk_buff` structures (Fig. 2 (a)). These aggregated packets have to traverse the bridge and then to the backend interface within the driver domain (Fig. 1). However, since the backend cannot handle the aggregated packets, the packets have to be copied into a continuous memory region at the bridge (Fig. 2 (b)). When the aggregated packets arrive at the backend interface, they are transmitted to the corresponding guest domain. This inter-domain communication is performed by copying the packets to the memory pages for which the driver domain is granted access by the guest domain (Fig. 2 (c)). We have modified the bridge so that it can directly pass the aggregated packets to the backend without copying; i.e. the step of Fig. 2 (b) is eliminated.



(a) After LRO (b) Bridge (c) Backend

Fig. 2. Flow LROed Packets in the Driver Domain. (a) Five packets are aggregated by sk_buff structures. (b) At the Bridge, packets are copied to a continuous memory region. (c) At the backend, these packets are copied to memory pages for transmission to the frontend.

In the NIC with LRO, packet aggregation takes place at every polling from the kernel. However, the results obtained in our prior work (Table I) indicate that the number of packets combined in a single operation is quite small. This small number of aggregated packets (LRO rate) implies the small benefit of LRO because LRO is designed to reduce the per-packet overhead, such as header inspection, by handling multiple packets in a single operation. Our second optimization is to perform the aggregation at every another polling. This option should increase the number of aggregated packets, leading to a lower packet handling overhead at the expense of increased latency.

Configuration	Linux	Xen	LRO
Throughput	715.1	569.0	606.0
LRO Rate	1	1	2.65

TABLE I. Summary from [10]. Throughput are in Mbps. LRO rates stand for the numbers of packets aggregated in a single operation.

III. Experimental Results and Analysis

In this section, we present the results and analysis of the experiments. First, the hardware and software environments used for the experiments are described. Next, we present the measurement results of network receive throughput with and without LRO and two optimizations presented earlier.

A. Experimental Environment

Table II shows the details of the machine used as the message receiver. This machine has a dual-core Xeon processor, 2GB of memory and a Gigabit NIC. However, only one core is activated throughout the experiments in this paper. This is because we only run the guest domain for the message receiver on the receiver machine to see the effects of LRO and optimizations in this paper. Changes in the network performance when another guest domain is consuming CPU time can be found in [10][1]. The base operating system is Linux kernel 2.6.18 and Xen 3.1.1 is used for virtualization. The machine used as the message sender has an AMD Athlon 64x2 and is also based on Linux 2.6.18. It is connected to the receiver machine by a cross-cable. Depending on the type of NIC, fragmented packets are linked either by the `next` pointer in the skb or by the `frags` array. LRO operations performed in these types are called skb-mode and page-mode, respectively. The former type is used in the r8169-based NIC of the receiver machine. For the measurements of message throughput and response time, we use the TCP_STREAM_TEST and TCP_RR tests of netperf suite [11], respectively. Each test is run for five minutes. For the TCP_STREAM_TEST, a single message size of 1KB is used because, from our experiences in [10], the throughput is almost constant for the message sizes of 256B or larger. To obtain the packet handling overhead in number of clock cycles, we use oprofile [12]. For the measurements of native Linux, we actually use the same system as Xen cases, but the `netserver` (the receiver program of the netperf suite) is run inside the driver domain so that the results are not affected by the overhead of Xen internal network architecture described in Section II-A.

[1]Optimizations discussed in this paper were not applied in [10]

Component	Description
CPU	Xeon 3GHz
Memory	2GB
NIC	1Gbps
Operating System	Linux 2.6.18
VMM	Xen 3.1.1
Network Measurement	Netperf 2.4.4
Guest Domain	
vCPU	1
Memory	512MB

TABLE II. Benchmarking Environments

B. Results and Analysis

Abbrev.	System
Linux	Native Linux
Xen	Unmodified Xen
LRO	Xen with LRO
BO	LRO with Backend Optimization
DA	LRO with Delayed Aggregation

TABLE III. System Abbreviations

In this section, we compare the results of measurements on the systems listed in Table III. Table IV shows the throughput results obtained by the TCP_STREAM_TEST. On the native Linux, 721Mbps is achieved, but on the original Xen, it is 536Mpbs, which is only 74% of the Linux. With LRO, throughput is improved to 577Mpbs which is about 8% increase from that of Xen[2]. The backend optimization (BO) is not effective and we cannot see any throughput improvement over Xen. On the contrary, the delayed aggregation (DA) option achieves the throughput of 748Mbps, which is about 40% improvement over the original Xen. Moreover, it is 4% higher than native Linux.

System	Linux	Xen	LRO	BO	DA
Throughput	721.1	535.9	577.2	576.5	747.5
vs Linux (%)	100.0	74.3	80.0	80.0	103.7
vs Xen (%)	134.6	100.0	107.7	107.6	139.5

TABLE IV. Throughput Measurement Results in Mbps. vs Linux and vs Xen rows represent the relative ratios against Linux and Xen, respectively.

[2]These results are different from those presented in Table I. We consider the differences are due to the fact that we use a difference machine for the message sender. In [10], we used a machine with a dual-core Pentium-D, while we use an AMD Athlon 64x2 based machine in this paper. From our experiences, the combination of the mother board and the NIC significantly affects results of this type of test.

Table V shows the number of CPU clock cycles divided by the number of transferred bytes. In Xen-based systems (Xen, LRO, BO and DA), they are broken down into the clock cycles in the driver domain and those in the guest domain. In terms of overall clock cycles, Xen incurs about 32% of overhead, which is considered to be the cause of 26% performance degradation against Linux. With LRO, clock cycles in the driver and guest domains are reduced to 93% and 82% of Xen, respectively. While the BO options reduces the overhead in the driver domain from 13.4 to 13.0, it also increases that of the guest domain from 7.1 to 7.7. From the measurements using oprofile. we have found that `skb_copy_bits`, which copies the data from skb to the kernel buffer, contributes the most of the reduction in the driver domain, as expected in this option. The increase in the guest domain, which we did not expect, is mostly due to `__copy_to_user_ll`, which is invoked from `netserver` (the receiver side of netperf) and copies the received packets from the kernel to the user space. No further details have been obtained so far, but one possibility is that the elimination of copy at the bridge has altered the alignment of the packets which results in the extra latency of `__copy_to_user_ll` execution.

System	Linux	Xen	LRO	BO	DA
Driver	NA	14.5	13.4	13.0	3.3
vs Xen (%)	NA	100.0	92.8	89.6	22.8
Guest	NA	8.7	7.1	7.7	5.9
vs Xen (%)	NA	100.0	82.2	88.2	67.7
Total	17.5	23.2	20.6	20.6	9.2
vs Linux (%)	100.0	132.3	117.5	117.9	52.4
vs Xen (%)	75.6	100.0	88.8	89.1	39.6

TABLE V. Receiving Overhead in Clock Cycles per Byte.

As shown in Table VI, the DA option aggregates nearly twice the number of packets of LRO. As a result, the DA option reduces the overhead in the driver and guest to 1/4 and 2/3 of those in Xen, respectively. In terms of total overhead, the DA option reduces it to 52% and 40% of Linux and Xen, respectively.

System	LRO	DA
LRO Rate	2.55	4.71

TABLE VI. LRO Rates (Number of Aggregated Packets)

One of the concerns with the DA option is the extra latency incurred by its delaying aggregation. First, we measured the response time of non-aggregated packet using the TCP_RR test of netperf. The results in Table VII show the average numbers of synchronous request-response transactions per second. All Xen-based systems are about 17% lower than the native Linux. However, we hardly see differences among them and we consider that the effect of aggregation is isolated from the latency-sensitive traffic.

When the DA option is used, the latency increase due to aggregation is unavoidable because flushing the aggregated packets takes place at every other polling from the kernel. We measured the latency from the aggregation to the flush of the first group of packets using the time stamp counter (TSC) and it was $33.17\mu Sec$. This latency coincides with the following calculation. From Table VI, we see that the DA option waits for about two more packets to arrive before flushing. From $MTU = 1500B$ and throughput of 720Mbps, the packet arrival rate is around 60K packets per second, which means that a packet arrives every $16.7\mu Sec$. Whether this skew in packet arrival rate matters or not is a problem of application type and user expectation. One option is, however, to make the aggregation rate as a system parameter that is turned by the administrator.

System	Linux	Xen	LRO	BO	DA
Trans./Sec	6090	5036	5010	4999	5015

TABLE VII. Results of Netperf TCP_RR Tests. Each value represents the number of request-response transactions per second

.

IV. Related Work

Menon et. al. evaluated three optimization techniques for the network performance in Xen [4]: the virtual network interface, the data path between guest and driver domains, and the virtual memory. They redefined the virtual network interface so that it could utilize the hardware support from modern NICs such as TCP segmentation offloading (TSO). Menon et. al. analyzed the overhead of TCP receive operation and applied two optimizations [5]. In principle, their first optimization, receive aggregation, is equivalent to Themann's LRO patch [8]. It seems that they were only aware of the hardware implementation of LRO ([7]) at the time of writing and implemented the same mechanism from scratch. Both LRO and their receive aggregation reduce the per-packet overhead, such as header inspection. Their second optimization, acknowledgment offload, reduces the overhead of transmitting ACK packets and should also be applicable to LRO-based optimization.

In [1], the design of Xen is described in detail. In the paper, they compared the network performance of Xen with native Linux, VMware and User-mode Linux. Santos et. al. analyzied the Xen receiving network path by breaking down the CPU time into functional groups,

such as network device driver or grant copy [6]. They applied various optimization techniques to these categories and reduced the overhead. Some of the techniques they used (such as reducing the cost of grant operation) should be applicable to our cases and we plan to do so. When multiple domains on a single platform are configured to form a multi-tier system, inter-domain communication traffic can be large and it could be the performance bottle neck of the system. Jian Wang et. al. proposed XenLoop which is a inter-domain communication channel [13]. It bypasses the regular virtual network interfaces in Fig. 1, yet provides transparency in the sense that the application programs can still use standard TCP/IP APIs.

V. Conclusions and Future Work

In this paper, we reported our attempts of improving the network receive throughput of a Xen-based system. Starting from our previous work of porting large receive offload (LRO) to Xen, we optimized two points of its internal operation. The first option was to eliminate the packet copy at the bridge in the driver domain. This option reduced the overhead in the driver domain, but also increased the overhead in the guest domain. Consequently, the first option did not lead to an overall performance gain. The second option was to defer the flushing of aggregated packets to every other polling. This option improved the throughput by 30% from the previous implementation of LRO on Xen.

The topics of further investigation include the following. First, as mentioned above, the option of eliminating packet copy increased the overhead in the guest domain, which offset the overhead reduction in the driver domain. Identifying and solving this increased overhead should make this option useful. The timing of flushing the aggregated packets is a trade-off between the delay and the reduction in the packet handling overhead. Making this timing as a system parameter that is tuned manually or automatically according to the system load is another topic. We measured the network performance in a standalone environment (i. e. only the guest domain for the network receiver was running on the platform). Evaluating the performance under a consolidated environment, such as the models proposed by VMmark [14] and Vconsolidate [15], is also necessary.

References

[1] Paul Barham et. al., "Xen and the Art of Virtualization," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP'03)*, pp164–177, October 2003.

[2] "AMD Virtualization," `http://www.amd.com/virtualization` .

[3] "Intel Virtualization Technology in Computing," `http://www.intel.com/technology/virtualization/` .

[4] Aravind Menon, Alan Cox and Willy Zwaenepoel, "Optimizing Network Virtualization in Xen ," in *Proceedings of the 2006 USENIX Annual Technical Conference*, pp15–28, May–June 2006.

[5] Aravind Menon and Willy Zwaenepoel, "Optimizing TCP Receive Performance," in *Proceedings of the USENIX 2008 Annual Technical Conference*, pp85–98, 2008.

[6] Jose Renato Santos, et. al., "Bridging the Gap between Software and Hardware Techniques for I/O Virtualization," in *Proceedings of the Usenix '08*, pp29–42, June 2008.

[7] Leonid Grossman, "Large Receive Offload implementation in Neterion 10GbE Ethernet driver," in *Proceedings of the Ottawa Linux Symposium*, pp195–200, July 2005.

[8] Jan-Bernd Themann, "[RFC 0/1] lro: Generic Large Receive Offload for TCP traffic," Linux Kernel Mailing List archive, `http://lkml.org/lkml/2007/7/20/250`.

[9] Takayuki Hatori and Hitoshi Oi, "Implementation and Analysis of Large Receive Offload in a Virtualized System," in *Proceedings of the Virtualization Performance: Analysis, Characterization, and Tools (VPACT'08)*, April 2008.

[10] Hitoshi Oi and Fumio Nakajima, "Performance Analysis of Large Receive Offload in a Xen Virtualized System," in *Proceedings of 2009 International Conference on Computer Engineering and Technology (ICCET 2009)*, Vol. 1, pp475–480, Singapore, January 2009.

[11] "Netperf: a network performance benchmark," Hewlett-Packard Company, Feb. 15, 1995.

[12] "OProfile - A System Profiler for Linux (News)," `http://oprofile.sourceforge.net/news/` .

[13] Jian Wang, Kwame-Lante Wright and Kartik Gopalan, "XenLoop: A Transparent High Performance Inter-VM Network Loopback," in *Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC'08)*, pp109–118, June 2008.

[14] VMmark, `http://www.vmware.com/products/vmmark/`

[15] Jeffrey Casazza, Michael Greenfield and Kan Shi, "Redefining server performance characterization for virtualization benchmarking," in *Intel Technology Journal*, Vol. 10, Issue 03, August 2006.