Evaluating Xen for Router Virtualization

Norbert Egi[§] Adam Greenhalgh[‡] Mark Handley[‡] Mickaël Hoerdt[§] Laurent Mathy[§] Tim Schooley[‡]
[§]Computing Dept., Lancaster University,UK
{n.egi,m.hoerdt,l.mathy}@lancaster.ac.uk
[‡]Dept. of Computer Science,University College London,UK
{a.greenhalgh,m.handley,t.schooley}@cs.ucl.ac.uk

Abstract— **In this paper, we evaluate the performance of a software IP router forwarding plane inside the Xen virtual machine monitor environment with a view to identifying (some) design issues in Virtual Routers. To this end, we evaluate and compare the forwarding performance of two identical Linux software router configurations, run either above the Xen Hypervisor or within vanilla Linux. Even with minimal sized packets, we show that the Xen Dom0 privileged domain offers near native forwarding performance at the condition that the sollicitation to unpriviledged domains stay minimal, whereas Xen unprivileged domains offer very poor performance in every cases. This shows that an important design principle for virtual router platforms must be to handle all forwarding, for all virtual routers, onto the same forwarding engine, in order to avoid much detrimental per-packet context switching.**

I. Introduction

In the past few years virtualization has re-emerged as a hot topic in operating systems research, and it seems natural to wish to extend these benefits to network routing. If a single hardware platform can simultaneously perform the roles of multiple independent routers, then many practical applications are enabled. For example, a telecommunications provider might install one router in an office building, and support many separate small business customers on the same hardware platform, while allowing each to operate and configure their own virtual router.

Beyond the immediate commercial applications, virtual routers can be much more flexible. If we take advantage of the ability to isolate components through virtualization, we are no longer constrained to have the same layer 3 protocols in each virtual router. Indeed, in the wider community in projects such as the NSF GENI infrastructure project, or the proposed EU Fire initiative, it is felt that the current Internet Protocol suite and associated architecture may need to be revamped in the long term. An obvious way to support a new architecture is to have it run concurrently with the old one. Overlays are one possible scheme which have not seen general realization or use. An alternative approach is to remain agnostic to the layer 2 technology and run the different layer 3 protocols natively across the same layer 2 medium and to use virtualization and isolation in the network switching entities to separate the two.

In this paper we focus on the case of single commodity hardware platform using operating system virtualization to perform the roles of multiple independent edge routers. When designing a single platform virtual router a key decision is how to split the forwarding plane. In the following sections we introduce the options and compare them against native Linux forwarding.

Three enabling technologies have reached maturity at this point, making these ideas rather timely. The first of these is the support for hardware virtualization coupled with the increased resources available on commodity PC processors. This enables multiple concurrent operating systems to exist on a single platform, isolated from each other within their own separate domains. The second is Xen [**?**], which is a Virtual Machine Monitor, VMM, with its associated control software that makes the simultaneous running of multiple operating systems on a single system both feasible and manageable. The final enabling technology is XORP [**?**], an extensible open router platform. XORP is a routing software platform supporting the common routing protocols, designed with extensibility in mind. XORP's primary relevance is that it provides a flexible way to control forwarding on such virtual routers. In this paper though, we focus on the forwarding path rather than the control plane.

This paper is organized as follows : Section II presents the important aspects of Xen relative to virtual routing, Section III presents the experimental setup that we use to perform our evaluation, and Section IV presents our results. We conclude in section V.

II. Xen

Before we explore where to undertake packet forwarding in a virtual router we must first introduce the Xen networking model.

The Xen platform consists of a Xen *hypervisor* above the physical hardware and several *Domain*s, which represent the virtual machine, residing above the hypervisor. The hypervisor controls access to the physical machine's hardware for the guest domains. Rather than allowing guest domains to use privileged instructions directly, *hypercall*s are used to notify the Xen hypervisor about their need for executing a privileged instruction, at which point the hypervisor handles the request. For reliable and efficient hardware support, all the device drivers are kept in an *Isolated Driver Domain* (IDD, mainly known as *Domain 0*, or *dom0*) with special privileges. The devices in the driver domain (dom0) are accessed by the guest domains (domUs) via point-to-point links. See [**?**] for specific details.

The Xen network device block model connects an ethernet interface at each end of an I/O channel. The device channel interfaces found in dom0 are called *back-end* interfaces (`vifX.Y`, where `X` identifies domU while `Y` is unique for the interface among all the given domU's back-end interfaces) while the interfaces on the other side of the I/O channel, found in the domUs, are called *front-end* interfaces (`ethY`, where `Y` identifies the front-end interface of domU) and appear as the real network interfaces of the guest operating system running in domU. Packets passed to the interface on either side of the I/O channel will appear at the interface on the other side of the channel after going through the Xen hypervisor (see Figure 1).

Having examined the basic structure of Xen's networking subsystem we are now able to examine where we might undertake packet forwarding and how we might move packets between the network interfaces and the location of the packet forwarding code. There are two plausible locations to carry out packet forwarding for each virtual router instance :

- In virtual router's domU instance the operating system carries out normal forwarding. This is obviously simple as no changed are required for a normal operating system.
- All forwarding is carried out in dom0 for all the virtual routers on the box. This is more complex because virtual routers must be isolated from one another.

Simplicity of design suggests forwarding in domU is the most attractive solution but what are the costs associated with moving packets between dom0 and domU to carry out the forwarding and how does this scale with increasing numbers of domUs?

A. Xen Packet Forwarding Options

Xen provides three mechanisms to move packets between the network devices and the back-end interfaces: Bridging, Routing and Network Address Translation. These mechanisms are used to demultiplex the incoming traffic from the real (i.e. physical) interface to the guest domains and to multiplex the outgoing packets originated in the guest domains. By default, Xen uses bridging within dom0 to connect real and virtual interfaces, allowing transparency for applications and services running on guest domains.

Figure 1 shows the classical network internals in Xen's driver domain and a guest domain. This default Xen configuration uses software bridges within the driver domain to transfer the packets between the real interfaces residing in dom0 and the back-end interfaces associated to domUs. With this configuration all the guest domains can share the same network in return for an increased complexity of the driver domain, which is not an issue when dom0 only serves its basic duties, namely isolating the hardware devices and providing a safe access for the guest domains to these devices.

Upon arrival, a packet is handled by the device driver in dom0 and appears at first on `pethY`, the real interface of the physical device, which is also attached to the bridge. Hence, the bridge demultiplexes the packet and puts it on the proper back-end `vifX.Y` interface based on the destination guest domain's MAC address. The back-end interface hands the packet over to the Xen hypervisor, which transfer the packet to the corresponding front-end interface (in the destination domU). The procedure is the same for packets destined to an interface in dom0. Outgoing packets will traverse the same path, but the opposite way of course. In this configuration IP addresses are assigned only to the (virtual) `ethY` interfaces, which are seen as the real interfaces in each domain. In the rest of this paper, we will refer to this design setup as ”bridged”.

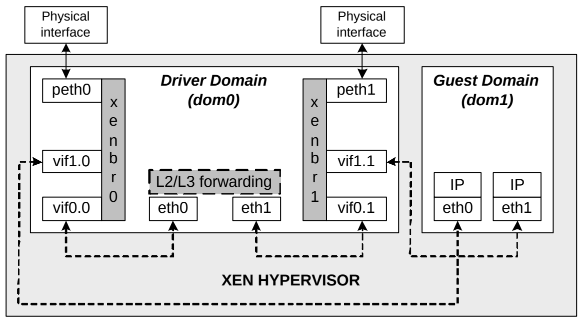


Fig. 1: Xen's classical network internals

An alternative to the classical bridging in dom0 is routing. In this latter configuration, the `ethY` interfaces of dom0 not only represent but are the real interfaces of the physical NIC (see Figure 2). Thus, it eliminates, from dom0's forwarding path, the I/O channels through the hypervisor as well as the bridges attached to each physical interface. As a consequence, packets forwarded from one to another physical interface in dom0 will traverse the same path as it would in native Linux. For sending packets to and from the domUs, IP addresses need to be assigned to the back-end interfaces too as opposed to the bridged setup. Packets destined to any of the domUs are routed from dom0's `ethY` interface to the appropriate `vifX.Y` back-end interface, from which point the hypervisor passes the packets to the `ethY` interface of the addressed domU, similar to the classical bridged configuration. Packets coming from the domUs cross the Xen hypervisor first and are routed to the outgoing interface determined by the routing protocol and based on the packets' destination address. In the rest of this paper, we will refer to this design setup as ”routed”.

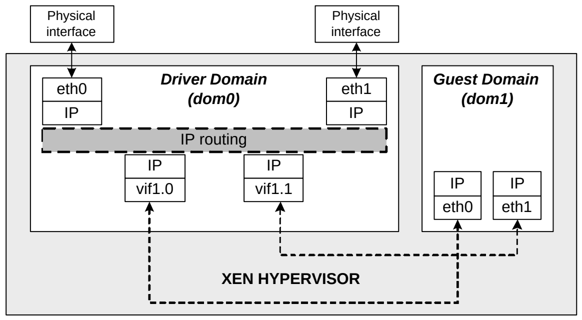


Fig. 2: Xen's routed network internals

Work has been done in the optimization of the network performance of Xen 2.0 VMM [**?**]. Three optimization categories were proposed. In superpage optimization, the number of TLB misses by guest domains is reduced with the use of new virtual memory primitives in the guest operating systems. I/O channel optimization includes mechanisms to make both transmit and receive between dom0 and domUs faster. High level virtual interface optimization consists of carrying out several TCP operations (e.g. scatter/gather I/O, TCP/IP checksum offload, and TCP segmentation offload (TSO)) in dom0, on behalf of guest operating systems . The evaluation of these methods show an insignificant impact of superpage and I/O channel optimizations by themselves and show a little impact when combined with high level optimization, which is the only method that gives a high performance hit to the system. Although the high level optimization is similar in spirit to our design concepts (i.e. performing functionalities in the driver domain on behalf of the guest domains in order to reduce the number of network packets the Xen hypervisor has to handle), the work actually carried out was targeted to servers (TCP flows) and did not consider the specifics of router forwarding.

III. Experimental setup

In the following experiments we aim to answer three questions which will inform our descision about where to forward packets within a virtual router. The first question is; how does dom0 forwarding compare with native linux forwarding and does increasing the number of domUs effect performance? The second question is; what is the impact of using the bridging and routing mechanisms to move packets from dom0 to the domUs? And the final question is; how does the forwarding performance of dom0 compare to that of a domU domain?

The experiments were carried out on the Heterogeneous Experimental Network (HEN) testbed[**?**], using a simple three node topology, as illustrated in Figure 3, consisting of a traffic generator (TG) on the left, a traffic sink (TS) on the right and the system under test (SUT) in the middle. Both TG and TS hardware configurations are Dell PowerEdge 1950 servers with two dual core Intel Xeon processors (2.66GHz each) and 2GB system memory. The SUT test was either a Sun Fire X4100 with one 2.2Ghz AMD Opteron(tm) Processor (single core) and 2GB of memory or a Dell 1950 servers described above. The Dell 1950 system's network interfaces are Gigabit Ethernet Intel(R) PRO/1000 NICs residing in PCI-Express x4 slot, while Sun Fire X4100 systems used the same network card but residing in a PCI-X 100 Mhz slot. The systems are interconnected via a Force10 E1200 non-blocking switching fabric.

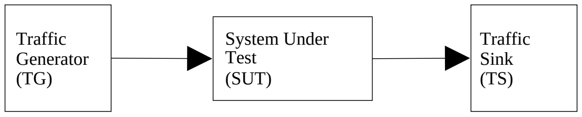


Fig. 3: Topology used

For generating and receiving traffic we used the Click modular router [**?**] software and its associated Linux kernel (2.6.16.33) polling patch. The Click modular router offers high performance traffic generation as well as measurement capabilities that we evaluated first before doing the actual performance measurements on the SUT. Figure 4 shows the performance over a direct link between TG and TS, when exchanging minimum 64 bytes frames. It can be seen (and it was verified) that loss-free linerate (e.g. 1.488.095 packets per second on a Gigabit Ethernet) was achieved between TG and TS. This rate also applies when the packets were categorized into more packet flows equally sharing the total bandwidth, up to 32 flows. This means that TG and TS are capable of saturating the ethernet link and that, on the three node topology, any loss observed with a rate below this maximum is happening in the SUT.

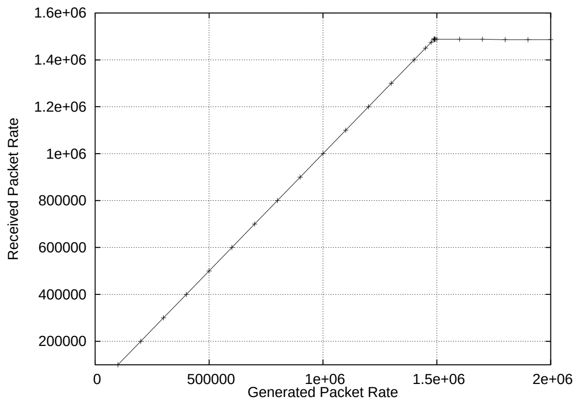


Fig. 4: TG to TS IP link performance

For each test, a constant bit rate of traffic was generated from TG to TS with an increasing packet rate between 100 kpps and 1500 kpps for the Dell 1950 SUT hardware or between 100 kpps and 1000 kpps for the Sun Fire X4100 hardware. We increased the rate with 100 kpps granularity and kept the packet at 64 bytes for all the experiments. Traffic was generated with each rate for 10 seconds and repeated three times. The rate for each 10 second burst was observed on the TS and the mean of the three readings was calculated.

The packets were forwarded by the SUT, using the various setups of native Linux version 2.6.16.33 and Xen version 3.0.4-1, all using the same kernel configuration. The vanilla e1000 network driver furnished in the kernel was used with NAPI[**?**] enabled on each network interfaces of the SUT. The arriving traffic in the traffic sink was examined and the number and rate of the packets were calculated and stored for later processing.

IV. Results

In this section, we present the results of our experiments and show that the default bridged scheme used in Xen for inter-domain packet transfer has significant costs for dom0's forwarding performance. We show the effects on performance of having multiple domUs forwarding traffic either alone or in conjunction with dom0.

We examine the performance of bridging, routing and a hybrid bridged routing scheme similar to Xen but running under native Linux. We then examine bridged and routed configurations within the Xen platform. We give our explanations of performance hits where applicable.

A. Native Linux Forwarding Performance

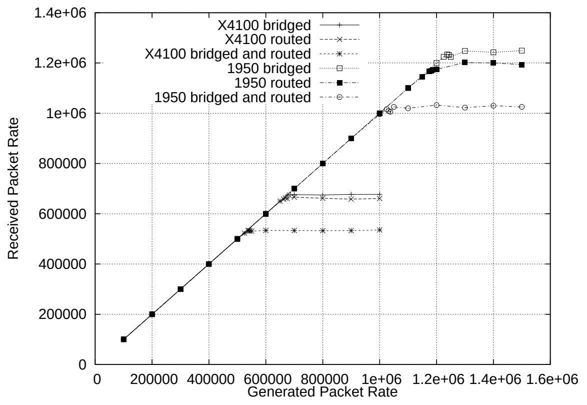
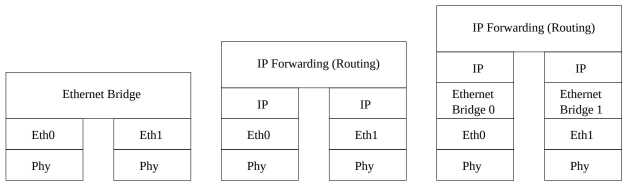


Fig. 5: Bridged native Linux vs routed native Linux vs routed and bridged native Linux

The measured performance of native Linux constitutes the reference numbers which the following Xen experiment results are compared with. They show the possible performance achievable on a general purpose hardware architecture like the Dell 1950 and the Sun Fire X4100. In the following section, we present only the results gathered with the Sun Fire X4100 hardware platform as the observed patterns were very similar on the other hardware architecture.

Figure 5 shows the performances of all the setups that we tested: a bridged setup (Figure 6(a)), a routed setup (Figure 6(b)) and a hybrid of the bridging and routing solutions (Figure 6(c)). The difference in the performance between simple bridged and routed is very small as opposed to the hybrid configuration which shows a 30% lower forwarding rate on both hardware configurations.



(a) Bridging only (b) Routing only (c) Hybrid of Briding and Routing

Fig. 6: Protocol stack of different configurations described in the paper

The protocol stacks showed on Figure 6 illustrate that with the hybrid setup, the complexity involved is more relevant. When a bridge is set up, Linux implements a logical interface that forwards the packets between the device driver and the IP layer whenever it is needed (i.e. the incoming packet is destined to the host itself or the host sends a packet out on a network interface attached to the bridge). The packet handler for this logical interface is implemented as a basic hook executed within a scheduled NAPI interrupt of the hardware network driver. This hook checks whether a physical interface is bridged when it receives a packet. If this interface is bridged, the bridging hook is called and the packet is passed to the upper kernel layer directly. NAPI is designed so that interrupts are allowed as fast as the system can process them [**?**]. NAPI uses the hardware DMA ring of the network card to store packets. When the system load reaches its limit, packet processing is poll-driven instead of interrupt driven. We conclude that packets are dropped in the DMA ring of the network card because of the overhead introduced by the bridge in the receive/send path. Why and where the software bridge is costly still need to be investigated in more detail.

As this bridge implementation is used to connect the real and virtual network interfaces in dom0, it is very likely that this performance hit will be observed on the Xen hosts too.

B. Xen Forwarding Performance

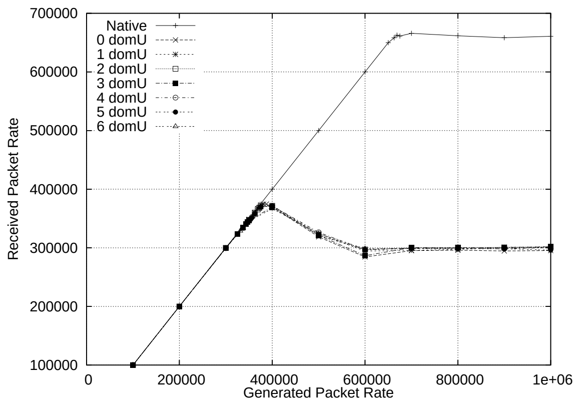


Fig. 7: Dom0 bridged forwarding performance with different numbers of domUs vs native Linux

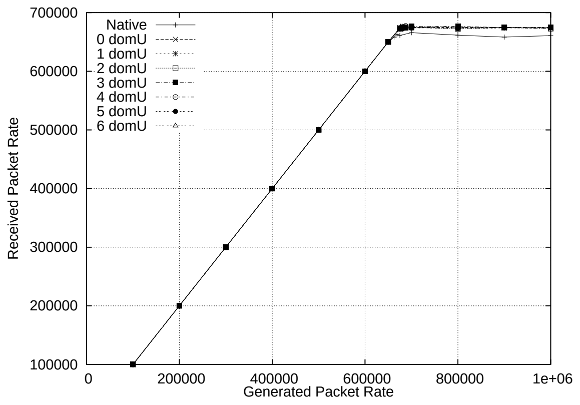


Fig. 8: Dom0 routed forwarding performance with different numbers of domUs vs native Linux

We start the evaluation of Xen with the results of the performance tests on the forwarding in the privileged driver domain (dom0). We only present the results for the Sun Fire X4100 architecture as the results observed on Dell's 1950 presented a similar pattern here too. It is worth noting that even by using Xen with the support of SMP (e.g. support for multi-processors) and allowing Xen to automatically allocate the domUs to each core of the Dell CPUs the results remained similar.

Figure 7 and 8 show the forwarding rate of dom0 with an increasing number of guest domains in the bridged and routed setups, respectively. In both figures the forwarding rate of native Linux for the same setup is plotted as well to illustrate the performance differences between Xen's driver domain and native Linux.

Both figures show that the number of running guest domains does not have any influence on the performance of dom0's forwarding performance if the domUs are not used to forward packets. Nevertheless, we can observe a high performance impact in Figure 7. This is due to the complex bridged I/O communication channel architecture explained in section II. Figure 8 shows that this performance hit can be avoided if the I/O communication channel between dom0 and the domUs is moved to the IP layer.

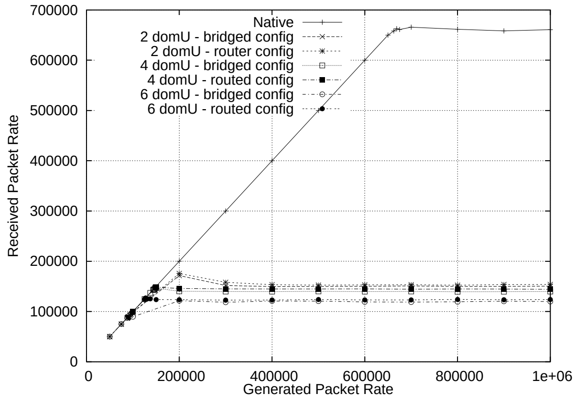


Fig. 9: DomU-only aggregated forwarding performance in bridged and routed setup with different numbers of domUs vs native Linux

We evaluate the forwarding performances of guest domains alone, as follows. Figure 9 shows the aggregate forwarding performances with increasing number of domUs running on the Sun Fire X4100 architecture. The traffic load is equally distributed amongst the domUs. First, it is clear that guest domains have a significantly lower forwarding rate than that of dom0. With 6 domUs the achieved aggregated throughput is roughly 6 times lower than that of dom0. Second, as we increase the number of domUs, the aggregated performance decreases independently of the I/O channel setup. This is as one would expect: as each guest domain requires CPU time to handle its packets far more context switches are required leaving less time for actual work to be done resulting in greater packet loss. This effect is amplified as the number of forwarding guest domains increases. The bottleneck is the Xen hypervisor here, as the performances observed in both routed and bridged setup are very similar.

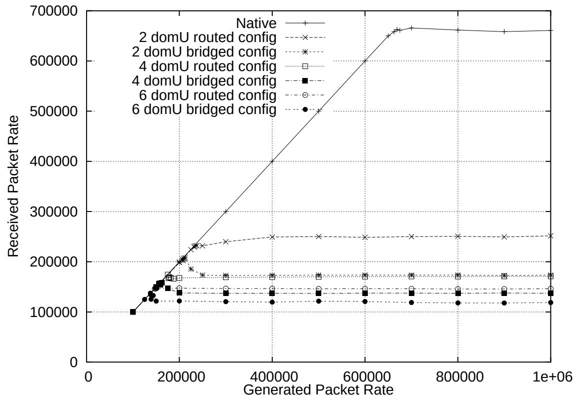


Fig. 10: Dom0 and domU aggregated forwarding performance in bridged and routed setup with different numbers of domUs vs native Linux

Figure 10 shows the aggregated forwarding performance of Xen when traffic is forwarded both by dom0 and an increasing number of domUs, starting from 2 domUs up to 6 domUs. As one would expect, the aggregate performance of Xen decreases as the number of domUs increases, again due to the extra context switches.

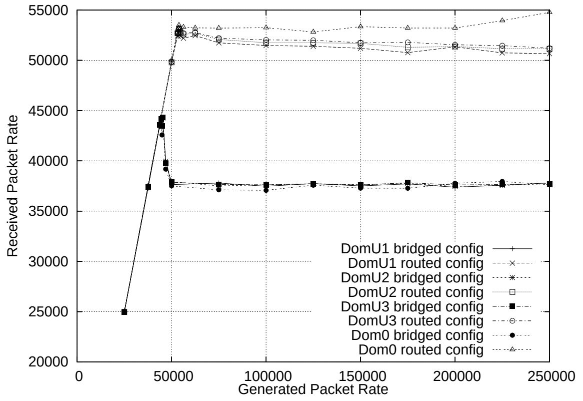


Fig. 11: Dom0 and domUs flow breakdown forwarding performance in bridged and routed setup with 3 domUs

Figures 11 and 12 show the per-domain performance when forwarding in dom0 and 3 and 6 domUs respectively. We observe how the hypervisor is allocating more resources to the priviledged dom0 domain but still does not ensure a guaranteed resource reservation mechanism: when the number of domUs increases and if traffic is flowing both through domUs and dom0, the performance of dom0 is deeply impacted even if the resources are allocated equally amongst the domains.

Again, the routed I/O channel is offering almost 100% more performance than the bridged one and the difference between dom0 and domU's performances is more pronounced as the Xen hypervisor is less sollicited with the bridged I/O channel.

In Figure 12 we observe that with the bridged I/O channel, dom0 is getting better performances than the domUs, whereas

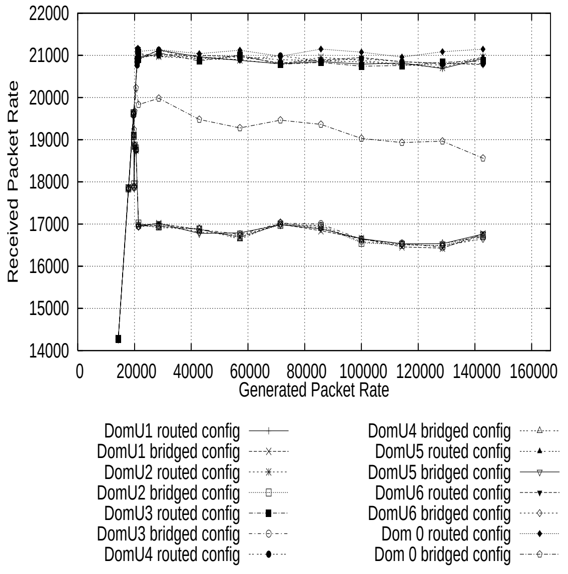


Fig. 12: Sun Fire X4100 Dom0 and domU fbw breakdown forwarding performance in bridged and routed setup with 6 domUs

with the routed I/O channel the performance difference between dom0 and the domUs is almost invisible. The whole system is so stressed that the Xen hypervisor is no longer able to reserve resource for the priviledged domain. With the bridged I/O channel and because packets forwarded even by the priviledged domain have to pass through the hypervisor, as illustrated in Figure 1, Xen is still able to privilege dom0, but it does not prevent dom0 forwarding performance to degrade compared to the case without forwarding in domUs.

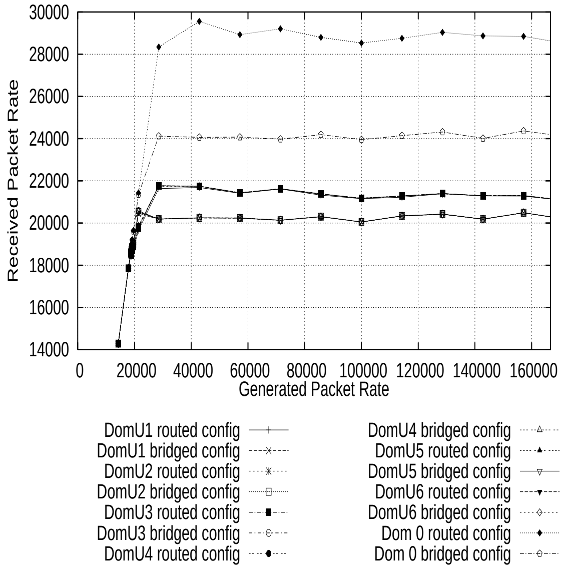


Fig. 13: Dell 1950 Dom0 and domU fbw breakdown forwarding performance in bridged and routed setup with 6 domUs

The same behaviour were not observed under the Dell 1950 architecture, as the hypervisor has more resources to dispatch. We can see on Figure 13 that with both routed and bridged I/O channel Xen is still able to privilege dom0, even if the performance drop compared to native Linux is still very remarkable.

V. Conclusion

Our main goal in running these experiments was to establish whether the forwarding perfomance of Xen is suitable for a virtual router platform.

We presented preliminary results for a performance evaluation of the forwarding path of a software router running under Xen virtualisation software. Those results showed that even with minimal packet size, Xen priviledged domain is able to forward packets as fast as native Linux at the important condition that the guest domains are not forwarding at the same time. Those results showed too that the performances of the unpriviledge domains are very low compared to native Linux, and that to build an efficient virtual router based on Xen, this must be taken into account.

Our result indicate that CPU saturation is a main feature of PC-based virtual routers, and that to avoid context-switching overheads, the virtualization platform (e.g. Xen) so all forwarding is handled in the privileged domain. In other words, domUs should only host the control (slow) path of its associated virtual router, while the corresponding forwarding should be "migrated" to dom0.

Although our current work has focused on PC-based solution using Xen, we believe that these general characteristics and recommendations will apply to any virtual router platform, and that the need for a single forwarding engine capable of forwarding for multiple virtual routers is a pre-requisite to achieve high performance. Our future work will tackle technique to achieve this.

References

[1] 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 *19th ACM Symposium on Operating Systems Principles*. ACM Press, October 2003.

[2] M. Handley, O. Hodson, and E. Kohler, "XORP: an open platform for network research," in *HOTNETS02*, October 2002.

[3] A. Menon, A. L. Cox, and W. Zwaenepoel, "Optimizing network virtualization in xen," in *Proceedings of the USENIX'06 Annual Technical Conference*, Boston, Massachusset, USA, June 2006, pp. 15–28.

[4] "Heterogeneous experimental network," http://www.cs.ucl.ac.uk/research/hen/.

[5] E. Kohler, R. Morris, B. Chen, J. Jahnotti, and M. F. Kasshoek, "The click modular router," *ACM Transaction on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[6] J. H. Salim, R. Olsson, and A. Kuznetsov, "Beyond softnet," in *5th Annual Linux Showcase and Conference*, November 2001.