Performance Isolation Issues in Network Virtualization in Xen

Blazej Adamczyk, Andrzej Chydzinski
Institute of Informatics
Silesian University of Technology
44-100 Gliwice, Poland
{*blazej.adamczyk,andrzej.chydzinski*}*@polsl.pl*

***Abstract*—Resource virtualization has been known and used for a while as a mean of better hardware utilization and cost reduction. Recently, the idea of virtualization of networking resources has become of vital importance to networking community. Among other things, this is connected with the fact that the virtulization principle is built in many discussed Future Internet (*FI*) architectures. In this study we deal with the virtualization of networking resources offered by Xen virtual machine monitor. We are especially interested in the performance isolation across virtual network adapters. Firstly, we demonstrate several problems with the performance isolation. In particular, the results of a number of experiments in which the activity of one virtual machine influences the network performance of any other are presented. We also examine the fairness, predictability and configurability of the network I/O scheduler in Xen. Secondly, we propose solutions to the problems revealed by our experiments. In particular, we introduce prioritization into Xen Netback driver, add a verification mechanism to the output buffer and discuss possibilities of some other improvements.**

***Keywords*-performance isolation; Xen; virtualization; network scheduler.**

I. Introduction

The increasing number of different IT services are making the virtualization idea a very important aspect of computer science. Virtual Machine Monitors (VMMs) bring about the dynamic resource allocation and enable full utilization even of the most powerful servers, while still maintaining good fault isolation between virtual machines (VMs, also called domains in Xen). However, the services provided over the network may require a certain quality, which is not easy to ensure in a virtualized environment. Several VMs can share the same physical network interface as well as other hardware (processor, memory etc.) what likely makes one VM affect other VMs' performance. Therefore, the *performance isolation* is crucial in case of some applications and has to be carefully verified.

In this paper, we focus on Xen VMM, [2], which is one of the most popular virtualization platforms and an Open Source project. Firstly, we present a study of the network performance isolation between Xen virtual machines. Different test scenarios allowed us to identify several problems. Secondly, we carefully analyze the Xen CPU scheduler and the network I/O scheduler to find out their possible source and resolution method.

The motivation behind our study is the fact that virtualization of networking resources has recently became of vital importance to networking community. This is partly connected with the fact that the virtualization principle is built in many projects dealing with propositions on the Future Internet architecture, for example in 4WARD FP7, [3], FIA MANA, [4], AKARI, [5], PASSIVE FP7, [6], GENI, [7], IIP, [8]. As observed in [9], creating high-level abstractions of networking resources, that cover the underlying physical infrastructure and implementation may help to overcome several drawbacks of the current Internet architecture. For instance, virtualization allows coexistence of multiple networking technologies in the network layer and offers a possibility to deploy easily new architectures, protocols and services.

The remaining part of the paper is structured as follows. First, a short account of the literature connected with the subject is given in Section II. In Section III-A, Xen general architecture is overviewed. Then, a detailed Xen networking structure is presented in Section III-B. A description of the Xen schedulers is presented in Section III-C. Section IV-A describes the testing environment and its parameterizations. The results and discussion on them are contained in Section IV-B. Finally, propositions of methods for improving the network performance isolation in Xen are presented in Section V. Conclusions are gathered in Section VI.

II. State of the art

This study verifies that there are problems related to the performance and isolation of virtualized network resources. Network adapter sharing and scheduling without virtualization is well described in literature. Virtualized environments, however, introduce additional software layer what does not allow to apply directly the existing solutions. Considering virtualized environments several previous studies [10]–[16], [27]–[29] focus on analysis of the performance of I/O operations and some of them present partial solutions. Unfortunately, these studies do not examine isolation and manageability in the field of resource sharing in considered virtualization platforms. In [17], however, the authors tried to approach the performance isolation problem focusing on all kinds of resources. Unfortunately, this study was performed on older version of Xen with an older CPU sched-

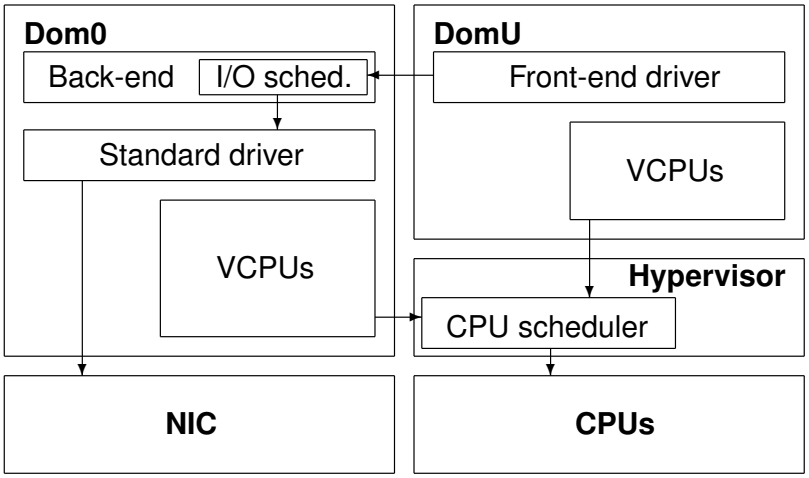

Figure 1. Xen architecture. Dom0 - Xen primary virtual machine, DomU - other Xen virtual machine, Hypervisor - main Xen operating system running directly on hardware, NIC - Network Interface Card, VCPU - virtual CPU.

uler implementation. They assumed that the main source of the problem is connected with CPU assignment and scheduling. As they have proven such general improvement idea can partially increase the performance isolation of all resources. We think, however, that to achieve really good performance isolation across virtual network adapters, the proposed CPU scheduler improvement is not the only change that has to be made because in the existing Xen networking implementation packet scheduling is performed randomly. We present that even on a low CPU utilization the problem is still noticeable and is related to the network scheduler itself. We have verified that applying a modified (for virtualization purposes) Weighted Round Robin (WRR) network scheduler improves the performance isolation and provides better control over virtual network devices.

III. Xen virtualization architecture

A. Xen VMM

Different virtualization environments have been developed throughout the years. Xen, due to its unique architecture (Fig. 1), is one of the leading solutions. The core of Xen, which is responsible for control over all virtual machines, is a tiny operating system called Xen Hypervisor. Its main tasks are CPU scheduling, memory assignment and interrupt forwarding. In contrast to other VMMs, the virtualization of all other resources is moved outside the hypervisor. Such original approach has the following advantages:

- Device drivers are not limited to the hypervisor operating system because they are installed on a virtual machine (any OS),
- Device drivers, as the most vulnerable software, are isolated from the hypervisor, significantly increasing the stability,
- Distributed virtualization of resources allows creation of several driver domains, eliminating the single point of failure,
- Small hypervisor operating system is much more reliable, efficient and stable.

There are two main virtualization methods. The first one allows to run any kind of OS and emulates all the necessary hardware to create an impression that the guest system is running on a physical machine. Second approach is to run a modified guest operating system, which is ”aware” of being virtualized. The latter, called *paravirtualization*, is much more efficient, but limited to some operating systems only. Xen provides both methods, but performs much better in the paravirtualization mode, which will be the only method used further in this paper.

To make the I/O operations as fast as possible, Xen introduced also paravirtualized device drivers. Each guest domain (Xen VMs are also called ”domains”) has the front-end drivers installed. Such drivers, provided with Xen, are communicating with the back-end drivers running on a special driver domain (Dom0 in Fig. 1). All requests addressed to a certain hardware are first scheduled and processed by the back-end driver, then are sent to the standard device driver inside the driver domain and finally reach the hardware. Thanks to Xen internal page-flipping mechanism called XenBus [18], [19] such solution is much more efficient than the standard emulation technique.

B. Xen networking architecture

To perform analysis of the network-related problems it is necessary to explain Xen networking architecture in detail. For each virtual interface the back-end network driver, called *Netback*, creates a virtual network interface in domain0 called *vif*. All virtual interfaces which share

the same physical device are connected with it using a standard Linux level-2 bridge. The Netback process which is responsible for handling traffic of each virtual interface is scheduling this traffic and passing it to the bridge. The existing scheduling scheme implemented in Xen by default is a simple Credit Scheduler and will be described in section III-C2. Finally, the bridge passes the packets to the device driver output queue and the device driver sends the packets to the hardware. Figure 2 presents the outgoing traffic path through Xen virtualization platform.

Such solution has several advantages. Firstly, the administrator has direct control over virtual interfaces from within domain0. Secondly, it is possible to monitor and analyze the packets passing through these interfaces using standard tools like *tcpdump* [20] or *wireshark* [21]. Finally, the traffic can be filtered and manipulated on the bridge level by creating custom ethernet bridge rules using *ebtables* [22] utility. All the above can be applied for both in and outgoing network traffic.

C. Xen schedulers

The main goal of this study is to examine the network performance isolation across Xen guest domains. It means to check if activity of one virtual machine influences the network performance of any other. The resulting knowledge is of great importance from the perspective of many network-related applications.

There are two elements in Xen, which may influence such isolation, namely the CPU scheduler and the network I/O scheduler [23]. Despite the fact that the schedulers are very simple algorithms, their analytical analysis is still far from being solvable. Creating a mathematical model of such systems even with large approximation is a very complex task and very often proves to be impossible. In the following two sections a description of the two schedulers is given.

1) CPU Scheduler: The fundamental part of each multi-tasking operating system is the CPU scheduler. Its aim is to create an impression that all running processes are executed in parallel. Typically, there are much more processes than available physical CPUs and the processes have to share CPU time. The scheduler is responsible for this division.

Inside Xen VMM, the hypervisor is the main operating system running on the physical machine. It is responsible for scheduling physical CPU time among virtual machines. To make the process easier the term *virtual CPU (VCPU)* is introduced. Every VM in Xen can have multiple virtual processors. Also, every domain is running operating system with another scheduler, which divides a VCPU time among processes running inside the guest operating system. The hypervisor on the other hand, schedules the physical CPU time among VCPUs.

The newest version of Xen uses the *credit scheduler* [24], [25]. It assigns two parameters for each domain - *weight* and *cap*. The weight defines how much CPU time a domain gets comparing to other virtual machines. The cap parameter is optional and describes the maximum amount of CPU a domain can consume. Using this two parameters the number of credits can be calculated. As a VCPU runs, it consumes credits. While VCPU has existing credits, its priority is called *under* and it gets CPU time normally. When there are no credits left, the priority changes to *over*. Each physical CPU maintains its own local VCPU queue. In the first place, the VCPU tasks with priority *under* from the local queue are executed. Then, if there are no VCPUs with priority *under*, the scheduler looks for such tasks in other CPU queues. If there are no tasks with priority *under*, the tasks with priority *over* from the local queue are executed. The credit scheduler in Xen can be summarized in the following algorithm and diagram (Fig. 3):

1) Process preemption - the scheduler takes control over CPU.
2) Last taken VCPU inserted back into the local queue according to its credits number.
3) Have the highest priority VCPU from the local queue used all its credits?
   - No: Highest priority VCPU taken from the local queue.
   - Yes: SMP Load Balancing - highest priority VCPU taken from other CPU queues.
4) Switching context to the currently taken VCPU - the VCPU takes control over CPU.

Considering this CPU scheduler in the context of the network performance isolation, it is worth noticing that the scheduler operates on virtual CPUs only, so it should not have a strong impact on I/O performance. The network I/O scheduler, on the other hand, works as a kernel thread inside domain0 using its VCPUs so a fair VCPU scheduling scheme should not influence its performance. However, it may happen that one misbehaving VM will slow down the total responsiveness and performance of other domains. Also, as it was presented in [17], the Xen CPU scheduler does not take into account the amount of CPU consumed by the driver domain on behalf of other VM. This may also have an impact on the network performance isolation, as some domains may use more CPU time than they are allowed. Furthermore, a different type of I/O request (e.g., more demanding, like disk driver requests) can potentially slow down the driver domain and affect the network performance of other VMs.

2) Network I/O scheduler: Looking at Xen architecture and analyzing its source code from the network performance isolation point of view, one can easily note that the most interesting part is the aforementioned Netback driver. It contains another scheduler, responsible for gathering all I/O requests sent to a certain physical network adapter. This network scheduler is not a complex mechanism and probably can be improved. Its only configuration parameter is the

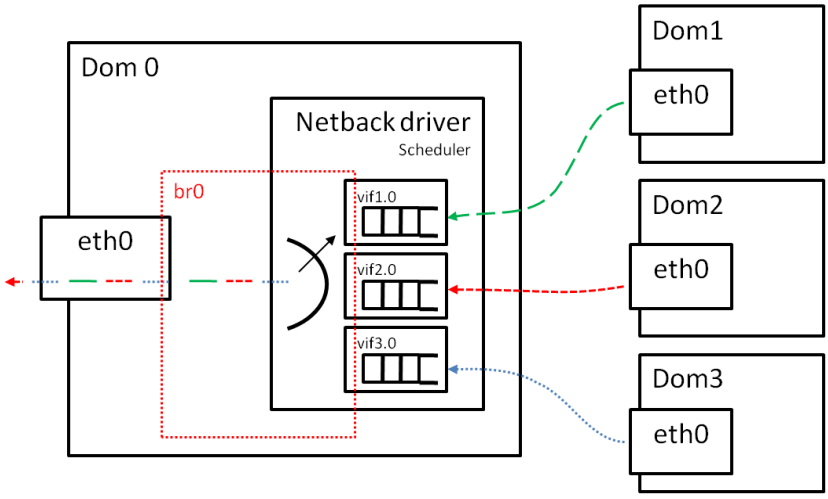

Figure 2. Xen networking architecture.

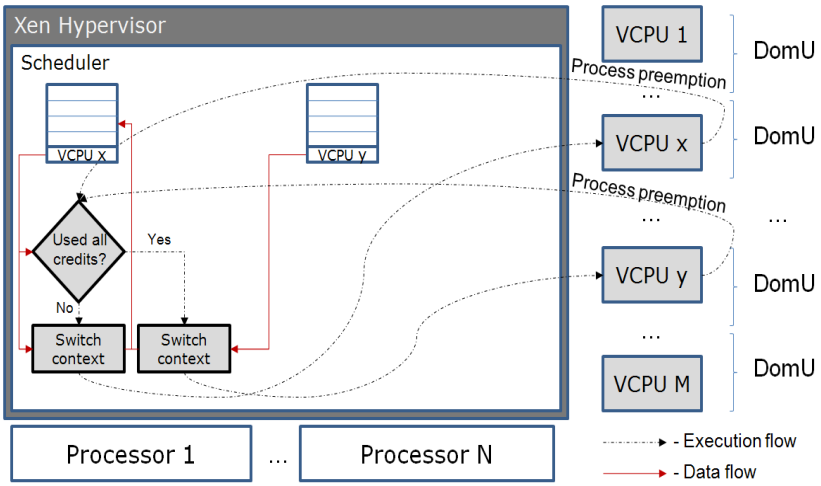

Figure 3. Xen CPU scheduler

maximum rate (parameter *rate*) - in fact it can be perceived as the credits number in the scheduler. The administrator can specify only the maximal throughput achieved by a certain virtual network adapter. Unfortunately, there is no way to prioritize and control the quality of service in more details.

The scheduler itself counts the amount of data sent/received in given periods. If *rate* has been reached, it sets a callback to process the request in next periods. Such solution is efficient, but does not guarantee any fair share or quality. In fact, a misbehaving VM can theoretically flood driver domain with requests because it processes all of them even those which are further rejected.

IV. Experiements

A. Experimental setup

To perform the tests, we installed Linux Gentoo with Xen 4.0.0 on Intel Quad Core 2 (2.83GHz), 4GB RAM, with hardware virtualization support. Two guest domains, each having 1 VCPU and 1GB of RAM, were created. Although there were separate physical CPU available for each VM, both VCPUs were pinned to the same physical CPU. Such configuration was used in order to check the influence of the CPU scheduler on the network performance. All network measurements were taken using *iperf* application. The UDP protocol transferring datagrams of 1500B to an external host over 100Mb link was used. We used the 100Mb link (instead of 1Gb) to demonstrate that the isolation problems are still present without a heavy CPU utilization. Only the outgoing

traffic was measured, as this was our main point of interest. The testing environment is presented in Fig. 4.

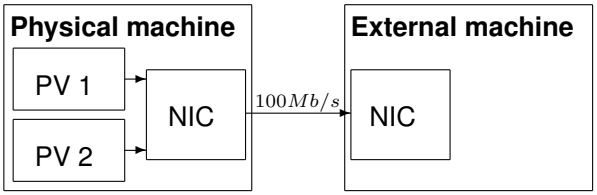

Figure 4. Testbed configuration. PV1, PV2 - Xen paravirtualized machines, NIC - Network Card Interface.

B. Results

In the first experiment, we observed how activity of one VM can affect the performance of another, when both VMs are configured with the same *rate* parameter. Four values of *rate* were used in different test runs: $25Mb/s$, $30Mb/s$, $35Mb/s$ and $40Mb/s$. In every run one machine started its transfer at the very beginning and the other started after 5s of delay. For every *rate* value, the experiment was repeated 10 times and the 0.95 confidence intervals were derived. The results are presented in Fig. 5.

Firstly, we can see that the actual rate is always a little smaller than *rate* parameter. As for the performance isolation, it is not too bad for low values of *rate*. However, with growing *rate*, the confidence intervals are getting larger and larger - in sample runs we can observe stronger variations of the throughput achieved by each VM. For the value of *rate* equal to 35Mb/s, the performance isolation becomes rather weak (although only about 60 percent of the total bandwidth is consumed). It is also worth to mention that a single VM throughput is stable even above $80Mb/s$ what proves that this effect is in fact a performance isolation issue.

Thus the only way to achieve a good isolation is to limit virtual adapters by far, which is not a satisfactory solution. Also, it is worth mentioning that having only the upper limit parameter is not enough in many cases. It would be much better to have any means to prioritize certain virtual adapter or even to have a minimum rate parameter and a scheduler satisfying these requirements.

In the second experiment, different *rate* values per each VM were used. Fig. 6 shows results for $rate = 30Mb/s$ in one VM, and $rate = 40Mb/s$ in another. The isolation problem still remains but, what is worth noticing, both VMs affect each other similarly.

In the presented two experiments the performance isolation problem was either mild or moderate, depending on the configuration. In the following two experiments, we will demonstrate more severe performance isolation issues.

In the third experiment, we verified how Xen divides available bandwidth among two VMs when the maximal rate is not set. A sample path of the throughput achieved by each

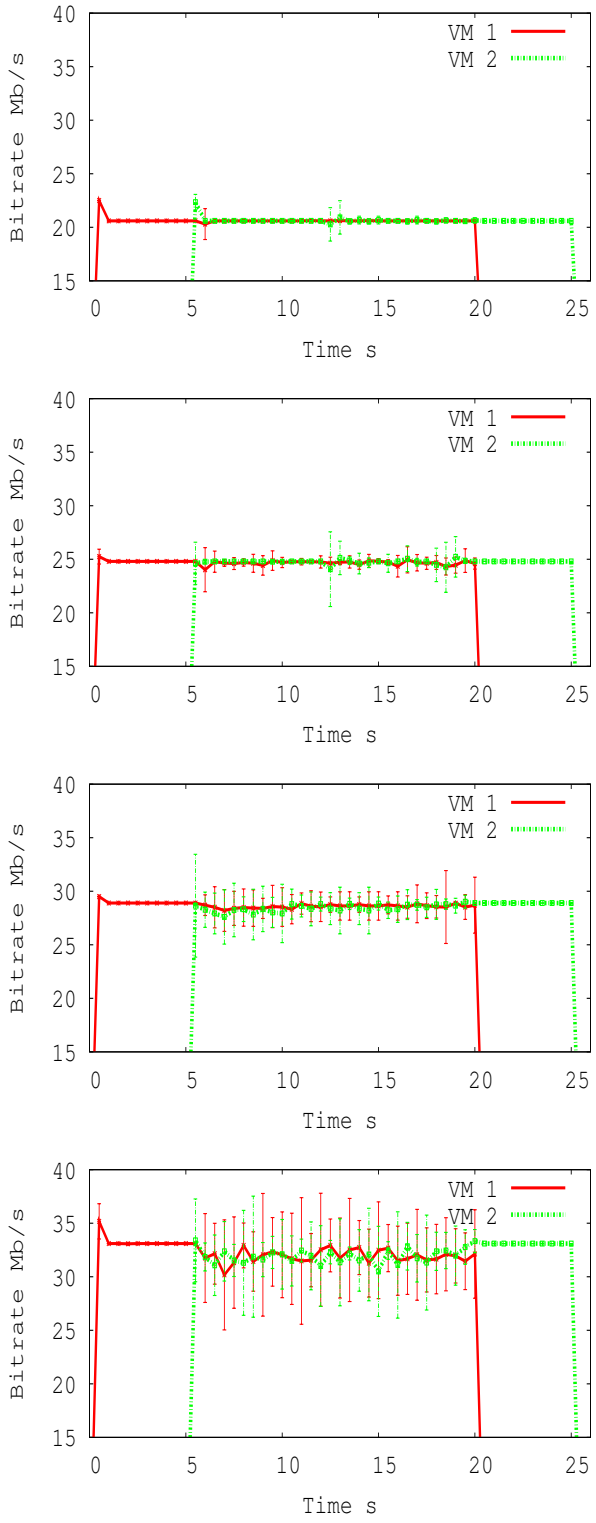

Figure 5. The throughput per VM for different values of *rate* parameter, namely for $25Mb/s$, $30Mb/s$, $35Mb/s$ and $40Mb/s$, counting from the top.

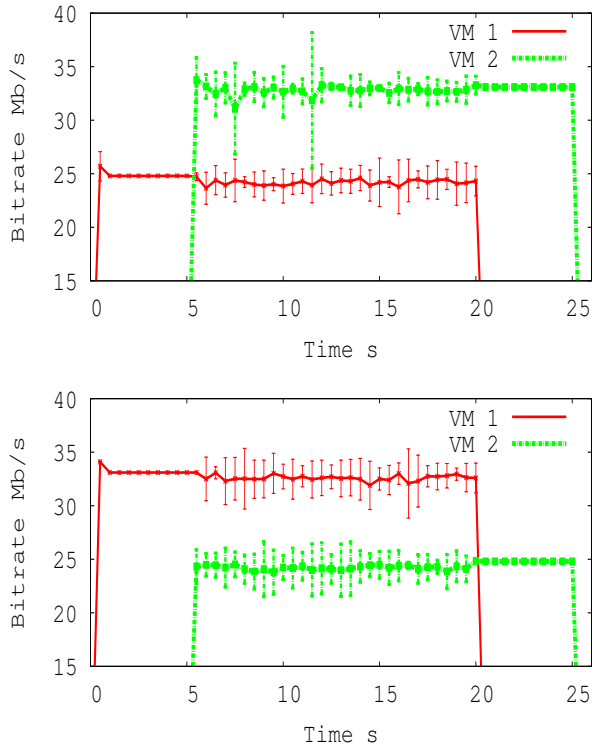

Figure 6. Total throughput per VM for different values of *rate* parameter ($30Mb/s$ and $40Mb/s$.

VM in time is presented in Fig. 7. Surprisingly, sometimes one virtual machine gets the total throughput and the other's throughput decreases to 0. Moreover, there are long periods when one VM dominates the other by far. Therefore, we have in fact no performance isolation at all in this case.

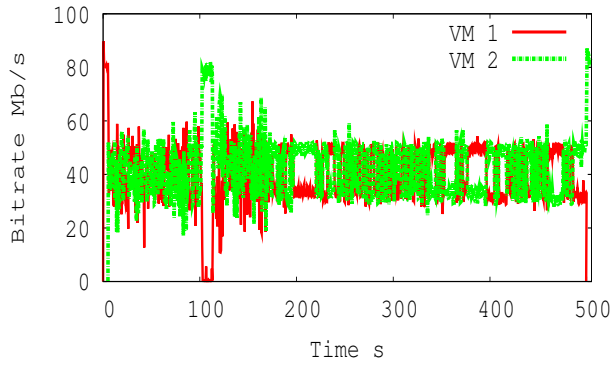

Figure 7. Sample throughput processes in time for two separate VMs without limits

In the fourth set of tests, we wanted to verify if a very abusive virtual machine can take more bandwidth than others. This time we wanted to check the performance isolation of the network I/O scheduler only, therefore we pinned one physical CPU to each VM.

In the first test, one domain was trying to transfer data over one connection using full available speed, while the second domain was using two connections, both of them trying to achieve full available speed. In the next test, the second domain was using three connections at full available speed.

The results are presented in Fig. 8. As it can be observed, the more abusive domain is, the better throughput it achieves. Naturally, if the rate parameter had been set, the overactive domain would never have crossed the maximum rate. In the lower ranges however, the problem remains.

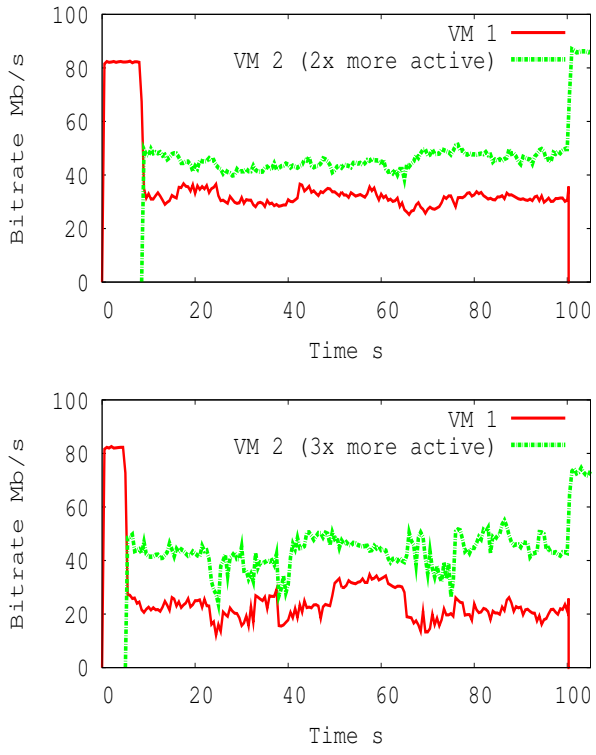

Figure 8. Bandwidth division with one overactive VM.

In the last experiment, we wanted to check if non-network I/O requests can influence the network performance isolation of another domain. During the experiment one VM was constantly sending datagrams at full speed, while the second VM was performing some extensive disk operations (*fio* tool was used for this purpose). The results are presented in Fig. 9; t_0 and t_1 are points in time when the extensive disk operations were initiated and finished, respectively.

We can see that other I/O request can also have a strong impact on the network performance. This is probably caused by driver domain not being able to process all the I/O requests. Block device access is being handled by separate block device back-end drivers. Disk operations are much more demanding in the driver domain than the Netback

drivers because disk is used by many crucial system components and when a disk request is in a blocked state waiting for a response all other mechanisms using disk are blocked as well causing the whole system to perform badly.

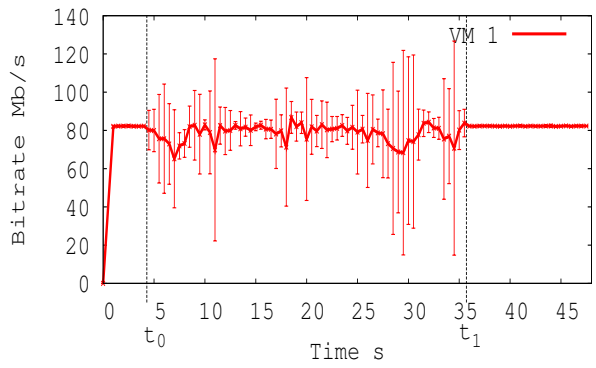

Figure 9. Disk I/O influence on network performance. (t_0 - disk I/O start, t_1 - disk I/O finish)

V. Improvements

After detailed analysis of the problem, we have gathered some ideas on how to modify Xen to improve the network performance isolation. Currently, in the driver domain several Netback kernel threads can be running, depending on the number of VCPUs. Furthermore, several virtual network adapters are mapped with one Netback kernel thread dynamically and this single Netback thread schedules the work using a simple round-robin algorithm, additionally taking into account *rate* parameter (omitting adapters, which used up all their bandwidth in the current period). Our idea is to introduce two additional parameters for every virtual adapter, namely *priority* and *min rate*. To implement the former, it would be necessary to change the round-robin mechanism to a more advanced priority based queue. Of course, we have to remember that the algorithm should not increase significantly the time complexity. The *min rate* parameter could use the same prioritization mechanism, assigning higher priorities to interfaces, which have not yet achieved the minimum rate. Depending on the results, it may be also necessary to introduce a user level application for maintaining the niceness level of each Netback thread inside the driver domain, according to actual needs.

1) Prioritization: The very first step to solve all the aforementioned problems is to introduce a prioritization mechanism into Xen's Netback driver. This will allow for better control over virtual interfaces and additionally schedule the packets in more predictable way thus preventing guests domains to flood the backend driver with requests what should result in improved isolation. To achieve such functionality we implemented the simple *Weighted Round Robin* algorithm [26]. We decided to use the WRR because of its simplicity, low complexity and to present that even the basic scheduling scheme can improve the performance isolation by far comparing to the native random scheduler. The actual implementation is presented in Algorithm 1.

Algorithm 1 The implemented version of WRR scheduler

```
min = infinity
for each vif do
    vif.weight = vif.priority/mean_pkt_size
    if min > vif.weight then
        min = vif.weight
    end if
end for
for each vif do
    vif.packets_to_serve = vif.weight/min
end for
while true do
    for each vif do
        if vif.has_packets_to_send() then
            counter = 0
            while counter < vif.packets_to_serve do
                event = vif.wait_for_event()
                if event = new_packet_to_send then
                    vif.process_packet()
                    set timer to end of transmission time
                else if event = timer_elpased then
                    counter++
                end if
            end while
        end if
    end for
end while
```

In a virtualized environment where a packet passes several virtual adapters before it reaches the actual real interface and each interface has its own input buffer, the WRR scheduler has to be modified to guarantee that the scheduled packets will not be dropped before they reach the wire. Dynamic and real-time priority assignment in this scheduler was created by additional Linux kernel *sysctl* parameters, i.e., *prioritize* and *priorities*. The first parameter defines whether to use the *WRR* scheduler or not. Second parameter is an array of the actual priority values for each virtual adapter.

Each *vif* has a separate queue of data to transfer and a *priority*. The latter corresponds to the weight in the implemented WRR algorithm. Total bandwidth available at the physical link is shared proportionally between all active virtual interfaces according to their weights. Because the packets are scheduled at the virtual driver level, they are processed almost immediately. This may cause wrong behaviour when the queue gets empty and after some time recieves a new packet while the last one is still transmitting. In standard WRR implementation the new packet would be transmitted because of the blocking send operation. This is why we had to introduce a waiting mechanism (in means

of a timer) so that all transmitted packets are actually sent before switching to the next queue.

To test the prioritization we performed simple experiment where two VMs transmit data to an external host. In the meantime the priorities were changed every second. At the beginning VM 1 had much bigger priority, in the end VM 2 was favored in the same proportion (i.e., $30/1$). The results are presented in Figure 10.

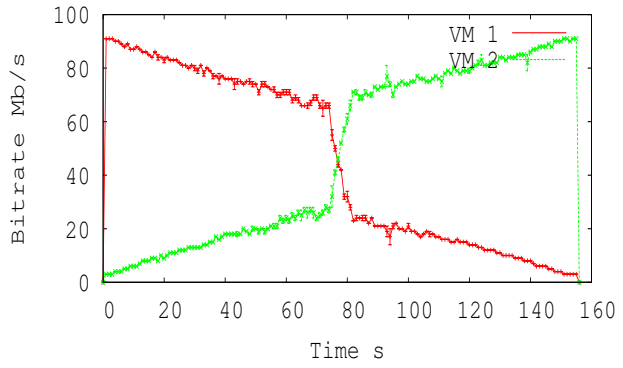

Figure 10. Results of the improved scheduler for changing priorities of each VM.

2) Buffer overflow in Domain 0: After implementing the WRR scheduling scheme the results in terms of performance and isolation were better but still not satisfactory especially at high throughputs (i.e., around the output device bandwidth). Gathering a small sample of output traffic at the physical device led us to the conclusion that the WRR scheduler is working correctly for most of the time however sometimes the packets from different *vifs* are not sent in correct amounts (according to WRR weights).

Knowing the above and that the scheduler itself is implemented correctly, it became obvious that the output traffic is being distorted after scheduling. Furthermore, looking at Xen networking architecture (see section III-B), one can easily notice that the scheduling is applied before packets get to the bridge and finally to the physical device output queue. Normally, when the output queue is getting full the driver informs higher layers and stops packet transmission using *netif_stop_queue* function. Xen Netback driver implementation lacks a mechanism of verification the output buffer of physical device before sending data to bridge. This results in distorted scheduling and larger amount of packet drops.

We modified the Netback module adding such verification. After detecting that the physical device output queue is full the packet is left in the *vifs* output queue. Of course this may lead to situation when the buffer of virtual interface gets full as well what finally results in *netif_stop_queue* at domainU level what is desired and makes the *vif* operation more similar to the operation of real hardware drivers.

3) Improved scheduler results: After applying the modification we have repeated the experiments to verify if the

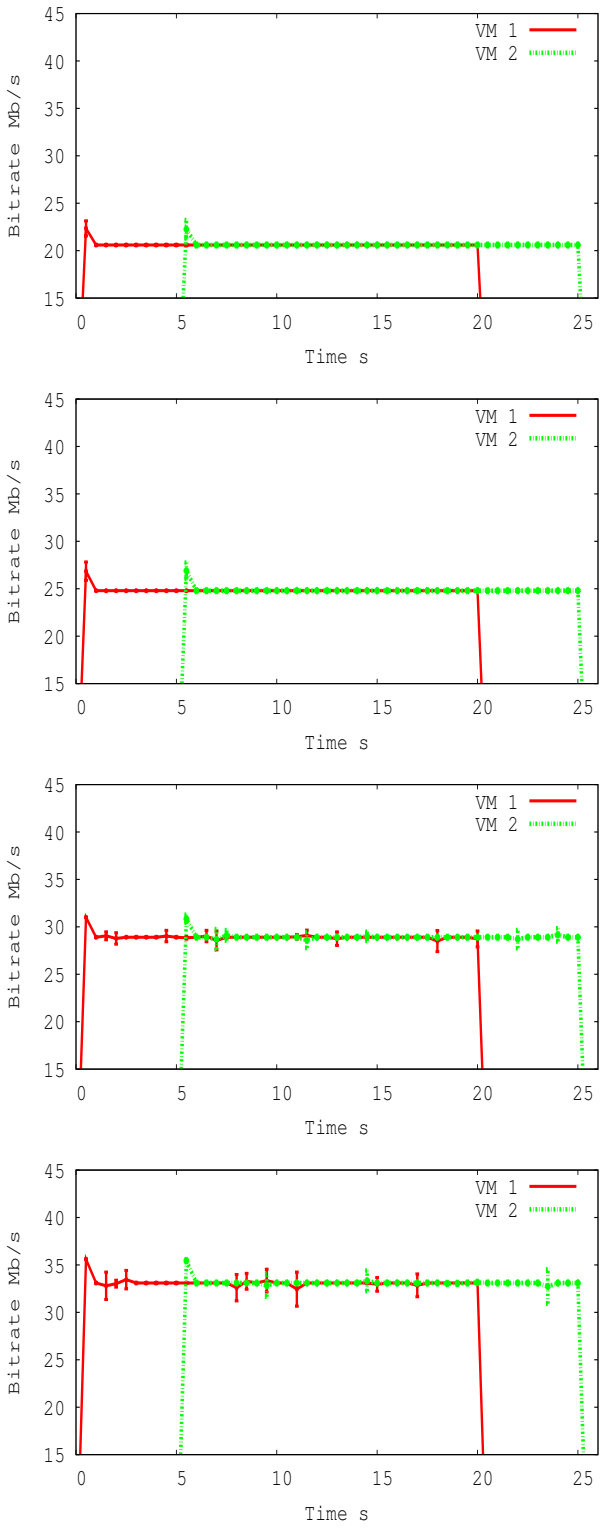

Figure 11. Results of the improved scheduler taking into consideration output buffer for *rate* parameter equal to $25Mb/s, 30Mb/s, 35Mb/s$ and $40Mb/s$.

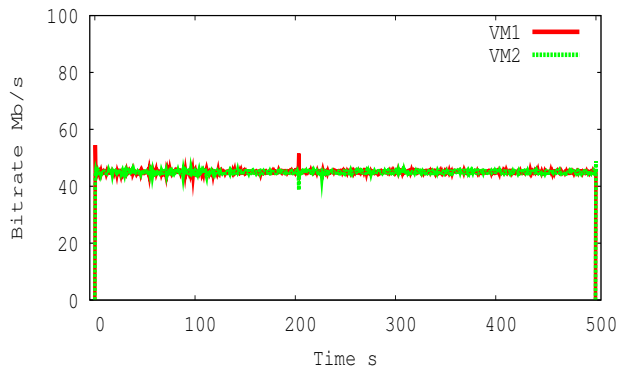

Figure 12. Results of the improved scheduler for one long measurement without $rate$ parameter.

modified scheduler indeed provides better isolation. Figures 11 and 12 present the results.

Comparing Figures 5 and 11 it can be easily seen that the WRR scheduler makes the bandwitdh sharing fair and stable. There does not seem to be any influence of one virtual interface on another. There are, however, some fluctuations (even if only one *vif* is transmitting) which were not noticed in the unmodified version what makes us think this is a minor problem which can be resolved and will be a subject of our further investigation.

Considering Figures 7 and 12 one can see that the modified Netback driver again makes the bandwidth sharing fair and predictable. Both virtual machines get more or less the same result and none is favored nor discriminated. In our opinion, this experiment shows the most significant benefits of applying our modification. In unmodified Xen environment there is no actual network scheduling what makes the outcome (both bandwidth and delays) dependent only on CPU scheduling and assignment. Application of WRR scheduler influences the way virtual interfaces send data and thus require CPU. This, as we can see, significantly decreases the CPU scheduling influence. Unfortunately, the disk I/O influence is still not fully addressed by our solution as we changed only the packet scheduling mechanism. Probably a good idea to minimize this effect would be to additionally use the solution proposed in **??** but unfortunately we could not verify this as we used a newer version of Xen.

It is worth to notice that the applied packet scheduler by improving the fairness and isolation can also positively influence the delays. We think that for our testing scenario, where the packet sizes were constant and the most important factor was the throughput, the WRR algorithm was a good choice. However, for other scenarius and use cases it might be better to implement a different scheduling scheme which may improve the interesting parameters. For example, when a real IP networks are concerned the packets have random sizes and thus it would be good to choose the Deficit Round Robin (*DRR*) scheduling or in case when fairness is concerned the Weighted Fair Queueing could be used. Of course, the more complex scheduling algorithm is used the more overhead is caused by the networking layer thus some schedulers may be hard to implement. In case of our implementation the overhead is minimal and does not affect the overall system performance.

Both the prioritization and buffer modification presented above may be of great use for system administrators who are providing services to external clients and wish to have good control over network resources and at the same time maintain the performance isolation at higher level.

4) Further improvements: Prioritization and the aforementioned buffer modification brings a lot of new possibilities and improves the performance isolation by far. Nevertheless, in high CPU utilization scenarios it may be not sufficient. We may think of much more complicated mechanisms. Virtualization makes the problem very complex, as three different schedulers may affect the isolation: CPU Scheduler, Domain 0 VCPU Scheduler and Netback I/O Scheduler. To achieve best results it might be necessary to synchronize all schedulers. Thus, partial solutions providing the *minimal rate* parameter for given virtual interface may prove very valuable. Further, a modification proposed in [17] may also help to increase the performance isolation taking the aggregate CPU consumption into consideration. Finally, we would like to test the scalability of our solution on a better hardware with more VMs running. All these are subjects of our future study.

VI. Conclusion

Xen is a powerful and stable virtualization platform, what accompanied with its Open Source formula makes it one of the most interesting VMMs, especially for research purposes. However, when the network virtualization is considered, the weak point of Xen is its lack of proper performance isolation. We demonstrated this using five sets of tests. The problems with isolation are caused by several factors mostly connected with CPU and I/O schedulers. We proposed the Netback driver modification using WRR algorithm to provide prioritization. We have also briefly presented an idea for future improvements.

VII. Acknowledgments

This work is partially funded by the European Union, European Funds 2007-2013, under contract number POIG.01.01.02-00-045/09-00 ”Future Internet Engineering”. This is extended version of the paper [1], presented during the International Conference on Cloud Computing, GRIDs, and Virtualization, Rome, September 25-30, 2011.

REFERENCES

[1] B. Adamczyk, A. Chydzinski: On the performance isolation across virtual network adapters in Xen, in Proceedings of the International Conference on Cloud Computing, GRIDs, and Virtualization. Rome, September 25–30, 2011, pp. 222–227.

[2] 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 Proceedings of the 19th ACM Symposium on Operating Systems Principles, New York, 2003, Vol. 37, pp. 164–177.

[3] The 4WARD Project, http://www.4ward-project.eu/index.php, 30-12-2011

[4] A. Galis, et al., Management and Service-aware Networking Architectures (MANA) for Future Internet. System Functions, Capabilities and Requirements, Position Paper, Version V6.0, 3rd May 2009.

[5] AKARI Architecture Design Project, http://akari-project.nict.go.jp/eng/index2.htm, 30-12-2011

[6] The PASSIVE Project, http://ict-passive.eu/about/, 30-12-2011

[7] Global Environment for Network Innovations Project, http://www.geni.net/, 30-12-2011

[8] Future Internet Engineering, http://iip.net.pl, 30-12-2011

[9] T. Anderson, L. Peterson, S. Shenker, J. Turner: Overcoming the Internet Impasse through Virtualization, Computer, Volume 38, Issue 4, April 2005, pp. 34–41.

[10] P. Padala et al.: Adaptive control of virtualized resources in utility computing environments, ACM SIGOPS Operating Systems Review, Vol. 41, No. 3, 2007, pp. 289–302.

[11] Y. Song, Y. Sun, H. Wang, and X. Song: An adaptive resource flowing scheme amongst VMs in a VM-based utility computing, in Proceedings of the 7th IEEE International Conference on Computer and Information Technology (CIT), 2007, pp. 1053–1058.

[12] J. Liu, W. Huang, B. Abali, and D. K. Panda: High performance VMM-bypass I/O in virtual machines, in Proceedings of the annual conference on USENIX, 2006, Vol. 6, pp. 3–3.

[13] V. Chadha, R. Illiikkal, R. Iyer, J. Moses, D. Newell, and R. J. Figueiredo: I/O processing in a virtualized platform: a simulation-driven approach, in Proceedings of the 3rd International Conference on Virtual Execution Environments, 2007, pp. 116–125.

[14] D. Ongaro, A. L. Cox, and S. Rixner: Scheduling I/O in virtual machine monitors, in Proceedings of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 2008, pp. 1–10.

[15] G. Liao, D. Guo, L. Bhuyan, and S. R. King: Software techniques to improve virtualized I/O performance on multi-core systems, in Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Jose, California, 2008, pp. 161–170.

[16] S. R. Seelam and P. J. Teller: Virtual I/O scheduler: a scheduler of schedulers for performance virtualization, in Proceedings of the 3rd International Conference on Virtual Execution Environments, 2007, pp. 105–115.

[17] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat: Enforcing Performance Isolation Across Virtual Machines in Xen; In Proceedings of the 7th ACM/IFIP/USENIX Middleware Conference, 2006, pp. 342–362.

[18] Y. Xia, Y. Niu, Y. Zheng, N. Jia, C. Yang, and X. Cheng: Analysis and Enhancement for Interactive-Oriented Virtual Machine Scheduling, in Proceedings of the IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008, Vol. 2, pp. 393–398.

[19] Xen Wiki, http://wiki.xensource.com/xenwiki/XenBus, 29-06-2011.

[20] Van Jacobson, Craig Leres and Steven McCanne: tcpdump, Lawrence Berkeley National Laboratory, University of California, Berkeley, http://www.tcpdump.org, 30-12-2011.

[21] Gerald Combs, et al.: Wireshark, http://www.wireshark.org/about.html, 30-12-2011.

[22] B. De Schuymer, et al.: ebtables, http://ebtables.sourceforge.net/, 30-12-2011.

[23] J. Matthews, E.M. Dow, T. Deshane, W. Hu, J. Bongio, P.F. Wilbur, and B. Johnson: Running Xen: A Hands-on Guide to the Art of Virtualization; Prentice Hall; April 2008.

[24] L. Cherkasova, D. Gupta, and A. Vahdat: Comparison of the three CPU schedulers in Xen, SIGMETRICS Performance Evaluation Review; September 2007, Vol. 35, No. 2., pp. 42–51.

[25] G. W. Dunlap: Scheduler development update, Xen Summit North America 2010, http://www.xen.org/files/xensummit_intel09/George_Dunlap.pdf, 29-06-2011.

[26] A. K. Parekh and R. G. Gallager: A generalized processor sharing approach to flow control in integrated services networks: The single-node case; IEEE/ACM Transactions on Networking; 1993, Vol. 1, pp. 344–357.

[27] G. Somani and S. Chaudhary: Application Performance Isolation in Virtualization in Cloud Computing; CLOUD 09. IEEE International Conference, 2009; pp. 41 48.

[28] N. M. M. K. Chowdhury and R. Boutaba: Network virtualization: state of the art and research challenges; Communications Magazine, IEEE, vol. 47, no. 7, pp. 20 26, Jul. 2009.

[29] P. Yuan, C. Ding, L. Cheng, S. Li, H. Jin, and W. Cao: VITS Test Suit: A Micro-benchmark for Evaluating Performance Isolation of Virtualization Systems; in e-Business Engineering (ICEBE), 2010 IEEE 7th International Conference on, 2010, pp. 132 139.