CESNET Technical Report 3/2008

Effects of Virtualisation on Behaviour and Performance Characteristics of Network Processing

David Antoš, Jiří Denemark, Jan Fousek

Received 6.3.2008

Abstract

This report compares overhead of IPv6 and IPv4 processing in operating systems virtualised with Xen and VServer and physical machines. The overhead is studied under various conditions: various higher-level protocols, runnig several virtual machines on a single physical one, under heavy computation load, etc.

Keywords: Xen, VServer, network performance, IPv4, IPv6

1 Introduction

Virtualisation is a promising way to deploy user controlled clusters and grids in a shared physical infrastructure. Numbers of distinct IP addresses are necessary for clusters of virtual machines—the required address space consumption grows typically several times, which is not feasible with the IPv4 addresses. If we require the machines to be addressed publicly and permanently, IPv6 must be used.

In this report, we study overhead of IPv6 processing in a virtual environment of the Xen [1] and VServer[1] virtualised systems. We compare it to IPv4 processing in the same environment and also with a native IPv4 and IPv6 deployment. We study various usage patterns and workloads with the goal to determine whether IPv6 can already be used within virtualised environment in a large computation infrastructure.

2 Virtualisation of the Network Stack

In this section, we describe architecture of network stack of virtual machine monitors Xen and VServer.

2.1 Xen and Networking

Xen [1] is a virtual machine monitor (VMM) that supports multiple guest operating systems on a single physical machine. Two modes of operation are possible: paravirtualisation, where the guest operating system must be aware of the virtualised environment and cooperate with the virtual machine monitor, and full virtualisation that allows running unmodified guest operating systems on hardware platforms that support hardware virtualisation extensions.

[1] `http://linux-vserver.org/`

In Xen terminology, virtual machines are called *domains*. A privileged domain called Dom0 is started by the Xen VMM on boot, it manages other domains and accesses hardware directly. Each guest operating system runs in a separate domain denoted by DomU (U standing for user).

We will briefly describe how network access is virtualised in Xen (see also Xen Networking[2]). As the Dom0 manages physical devices, the guest operating systems communicate with network interface cards via virtual interfaces in Dom0.

Virtual interfaces in Dom0 are connected with physical interfaces either by means of bridging on link layer, or they can be routed in Dom0 machine (possibly also with Network Address Translation). User domains see the exported virtual interfaces and treat them internally the same way as classical network interface cards. Connection of Dom0 virtual interfaces and interfaces seen by user domains can be seen as an abstraction of Ethernet wire.

The default Xen configuration uses bridging in Dom0 to allow all domains to appear on the network as individual hosts. Bridging can further be configured in domain 0 using *iptables* and *ebtables* (e.g., to prevent address spoofing).

When a packet arrives at the network card, it is handled by Dom0 driver. The packet is (still on level 2) passed to the bridge. The bridge distributes the packet the same way as a physical switch would, deciding about the final destination (i.e., the correct virtual interface) based on the packet's MAC address. The virtual interface puts the packet into the correct DomU. Sending from DomU proceeds analogically.

Xen also support direct export of the PCI device into the DomU. This is used, e.g., for providing direct access to fast low latency interfaces like Infiniband. We will not deal with these devices as their use limits the extent of available virtualisation.

2.2 VServer and Networking

VServer[3] does not create virtual network interfaces for virtual machines. Physical interfaces are shared among virtual machines. For each virtual machine an alias with appropriate IP address is created. The kernel assures processes from a virtual machine may only bind to its configured address. It creates the illusion of separate network interfaces.

3 Test Environment and Tools

This section describes hardware and tools used for testing.

3.1 Hardware, OS, Xen, and VServer Configuration

We used two machines for test, for brevity, we will call them 1 and 2.

Machine 1 is a Dual Core Opteron 280 at 2.4 GHz with 2 GB RAM, and Broadcom Tigon3 PCI-X network interface. Xen version 3.0.4-1 is installed, all Xen domains run Suse Linux 10.0 x86-64 with Linux kernel 2.6.16.33.

[2] http://wiki.xensource.com/xenwiki/XenNetworking

[3] http://linux-vserver.org/

This machine was used also for VServer measurements, version of VServer was 2.1.1-rc22 with IPv6 patches and VServer utilities 0.30.213. VServer virtual machines ran Suse Linux 10.0 x86-64.

For comparison purposes, we measured also physical machine performance without virtualisation, again with the same Linux distribution and kernel.

Machine 2 was used without virtualisation. The machine runs Intel Pentium Dual Core at 3 GHz with 1 GB RAM, and Broadcom Tigon3 PCI Express network interface. Linux kernel is 2.6.19 and the distribution is Gentoo Linux.

3.2 Network Measurement Tools

Iperf[4] is a network bandwidth measurement tool. *Iperf* measures TCP bandwidth with adjustable parameters, it can create UDP streams with specified bandwidth and measure packet loss.

Generator7 is a UDP generator and receiver based on *RTPgen/RTPsink* toolset [4]. *Generator7* uses RTP-like time-stamps and sequence numbers to measure latency and jitter of the communication. Jitter is measured according to methods described in [5].

3.3 Machine Load Measurement Tools

We intended to use *xenmon* [3] to study CPU load in the virtual machine. *Xenmon* monitors total and average CPU time, domain waiting and blocked time, and I/O operations. Unfortunately, this tool turned out to report meaningless data on multiprocessor machines, especially in the case when the measured domain moves from one CPU to another during the measurement interval. This issue could be solved binding domains to specific CPUs, but as this would affect default Xen behaviour, we decided to use a different measurement tool.

Apart from the network load the only needed quantity was the CPU load, so we could use the *xentop* tool [2], which is a standard Xen replacement of the top command. The *xentop* takes into account domain interactions. The results are related to the physical machine.

4 Measurements

Throughput measurements have been performed with *Iperf* server running on machine 1 and a client on machine 2. Terms "receive" (Rx) and "transmit" (Tx) are used from the point of view of machine 1.

Tests run 10 seconds from client to server and 10 seconds in the opposite direction (by means of `iperf --tradeoff`). Seven trials have been collected and median value of them is presented.

For UDP, *iperf* was instructed to send a 1000 Mbit/s stream (i.e., `--bandwidth 1000M`). With such a value, some packets got lost, so the measured values are therefore the packets successfully delivered. Note that by fine-tuning the sent stream, we would be able to obtain slightly better results, but this would not reflect the stan-

[4] `http://dast.nlanr.net/projects/Iperf/`

dard user environment. We present received UDP values only as transmit values are similar.

All UDP tests have been performed for a series of packet sizes from 36 B to 1470 B, precisely 36 B, 60 B, 100 B, 200 B, ..., 1300 B, 1400 B, and 1470 B for IPv4, the last value for IPv6 being 1450 B because of longer headers.

4.1 Native Performance

In order to set a basis for our comparison, we first measured throughput of native machines without Xen in IPv4 and IPv6. As we can see from Figure 2, UDP throughput of IPv6 is slightly lower than of IPv4, which is caused by more difficult classification of longer addresses.

Quite expectably, the overhead of network processing is related to a number of packets per second rather than overall bandwidth. Therefore, using small UDP packets only very limited bandwidth can be achieved.

For TCP streams, we can see in Table 1 (rows denoted "Native") that the TCP throughput in IPv6 reaches about 98.7% of IPv4.

4.2 Xen DomU Performance

In this set of experiments, machine 1 runs Xen with a single user domain. The DomU uses a single CPU and 512 MB RAM. The measurements were performed in DomU. The memory assigned to DomU is sufficient for all tests (i.e., no swapping occurred). No CPU limits are set for Dom0, it can run on both processors.

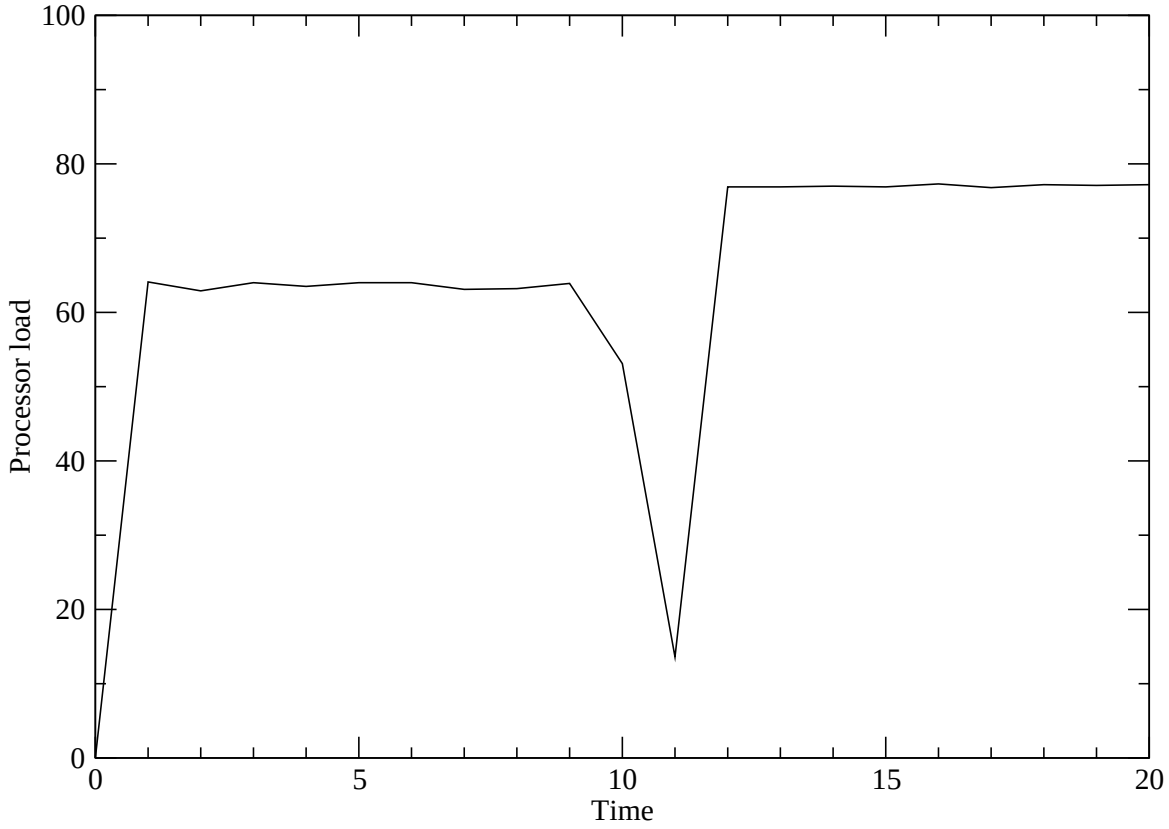

Figure 1. CPU Load in a Single Measurement

Before we present the results, we will show how CPU loads were obtained. Experiment for each packet size lasted for 20 seconds, half of the time receiving on machine 1, and the remaining 10 seconds transmitting from machine 1. CPU load was obtained in 1 second intervals with *xentop*. An example of such measurement is presented in Figure 1. As the CPU loads increase in the beginning of each part of the experiment and they are very stable afterwards, we computed the resulting CPU load as a median of middle 5 seconds of the transmit part of the experiment.

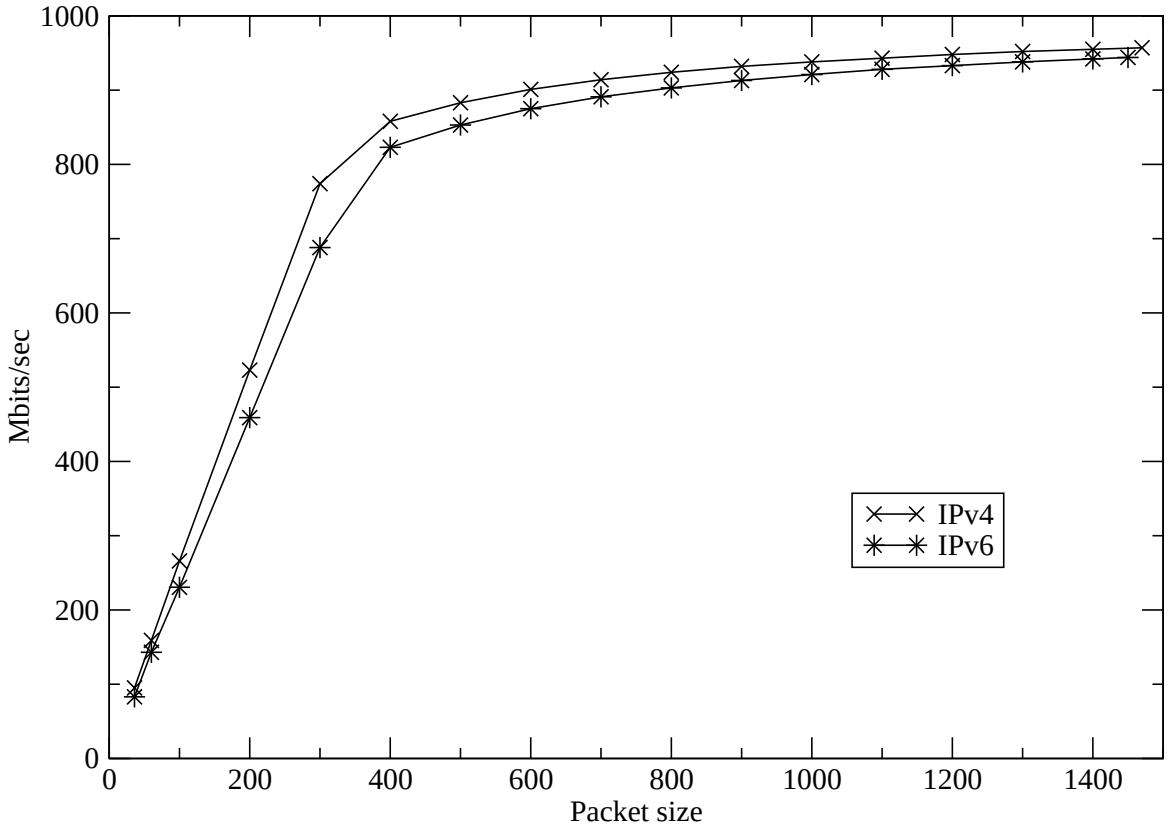

Figure 2. UDP Throughput in Native Linux

Figure 2 and Figure 3 show loss of throughput of UDP traffic in DomU compared to native Linux configuration for packets smaller than approximately 800 B. For longer packets, the throughput is not limited by the Xen environment. The difference of IPv6 and IPv4 performance is in similar relationship for both native and DomU setups, therefore the overhead of Xen is the limiting factor.

The throughput loss for smaller packets is caused by saturating the CPU for both domains, as we can see in Figure 4 and Figure 5. The figure shows CPU utilisation depending on packet size for IPv4 and IPv6. Both protocols behave in a similar manner. The point where CPU utilisation gets under 100% correlates exactly to the packet length where DomU and native Linux throughput are equal.

CPU load of Dom0 for packets under 800 B shows interesting change when going from IPv4 to IPv6. It is about 10% smaller for IPv6. We suggest that this is caused by the fact that DomU, using 100% CPU itself, is not able to generate sufficient data stream to saturate Dom0 in the IPv6 case. This observation also correlates with the UDP throughput.

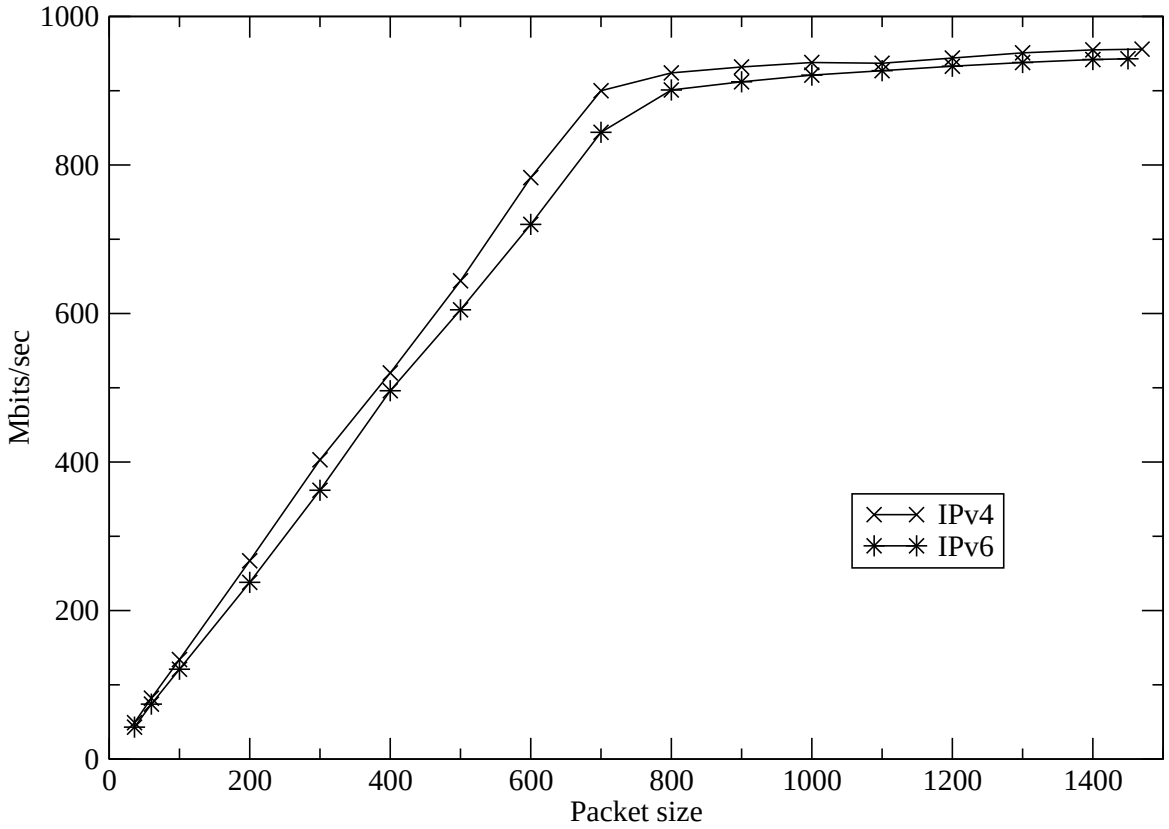

Figure 3. UDP Throughput in Xen, Single DomU

We have no explanation of the performance anomaly of DomU for packets of 1100 B to 1400 B. The increase in CPU load does not correspond to the basic principle that Xen overhead should correlate to packet number and size. Explaining the anomaly would require deeper study of Xen internals (e.g., buffer sizes and behaviour) and is left for future work.

Table 1. Native, DomU, and VServer TCP Throughput (Mbit/s)

	Rx	**Tx**
Native IPv4	936	936
Native IPv6	924	923
Xen DomU IPv4	936	932
Xen DomU IPv6	923	923
VServer IPv4	936	936
VServer IPv6	922	923

TCP throughput (Table 1) for a single DomU is practically the same as for native machines. Due to TCP segmenting, packets are large enough to minimise Xen overhead, making the behaviour similar to UDP for large packets.

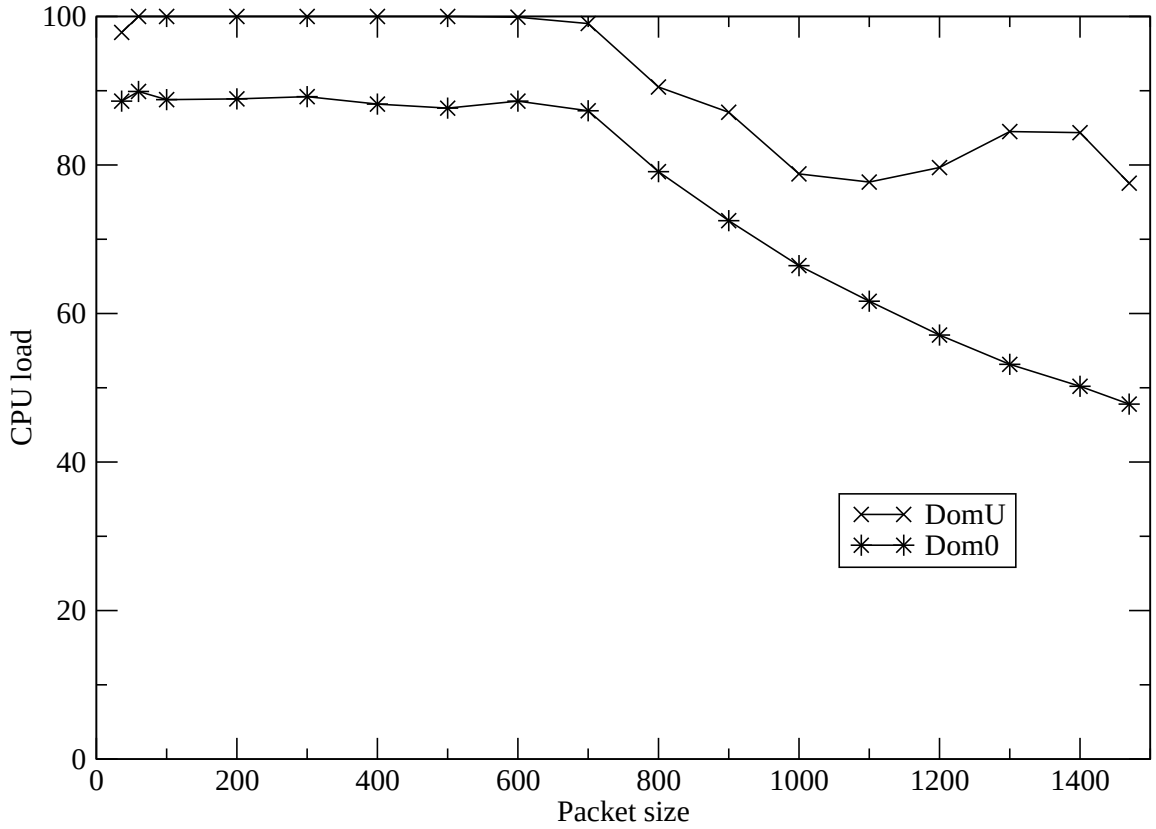

Figure 4. CPU Load for UDP Depending on Packet Size in Single DomU Configuration in Xen—IPv4

4.3 VServer Virtual Machine Performance

Similarly to Xen test described in the previous section, experiment for each packet size lasted 20 seconds, half of the time receiving on machine 1, the other half transmitting from machine 1.

VServer performance (Figure 6) is fully comparable to performance of the native machine. Surprisingly, VServer reaches higher throughput than the physical machine for packets between 200–300 B. We have currently no explanation of this phenomenon, we suggest it can be caused by timing dependencies in the kernel.

TCP throughput is practically the same as in the native machine for both IP protocol versions.

4.4 Double DomU Interactions in Xen

Similar throughput measurements were repeated with two user domains in Xen. In this configuration, each user domain used 512 MB RAM and one CPU (without explicit binding CPUs to domains, it is the task of Xen scheduler). Dom0 runs on both processors. All domains had the same weights, which is the default configuration. We measured both user domains in parallel in order to study fairness of Xen scheduling under heavy network operations.

Figure 7 and Figure 8 show throughput of UDP traffic for parallel user domains for both protocols. The domains share the resources fairly and this behaviour is very similar for both cases.

The “Sum” graph is the sum of the throughput. The aggregate throughput

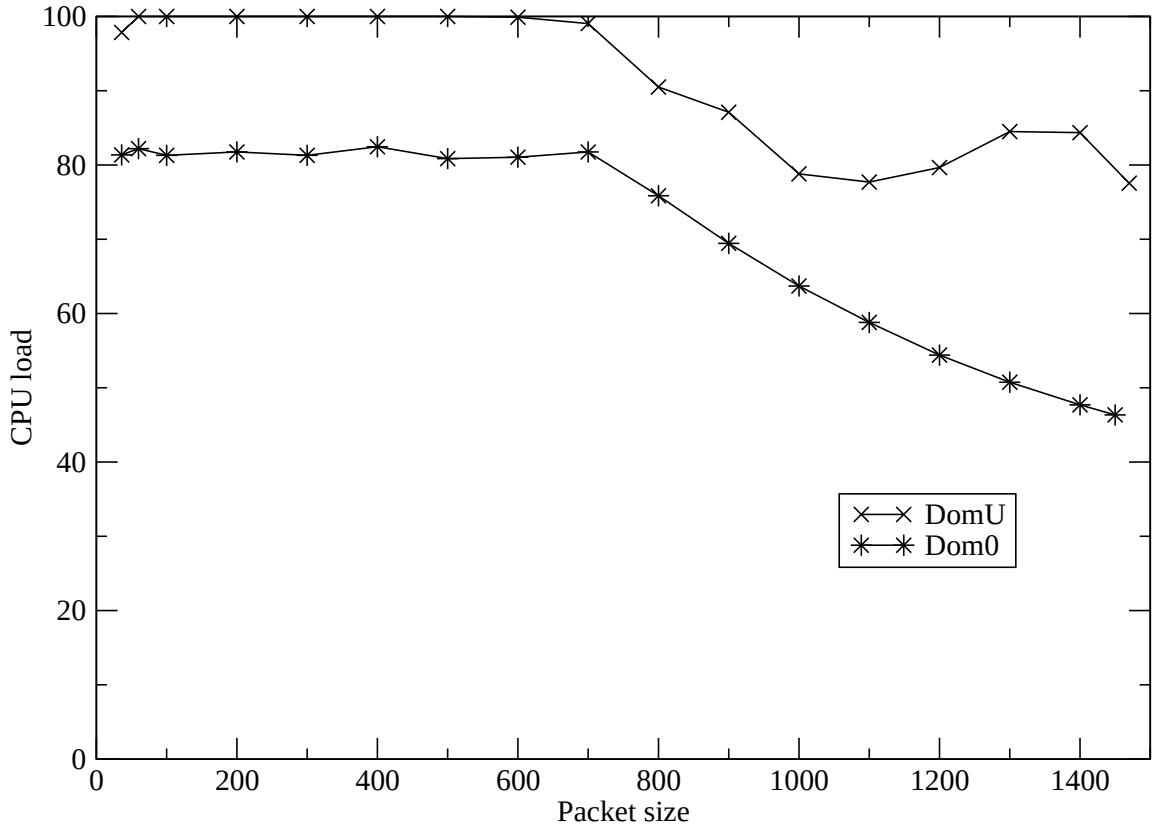

Figure 5. CPU Load for UDP Depending on Packet Size in Single DomU Configuration in Xen—IPv6

of two concurrent transmissions heading to separate DomUs is, in case of small packets, even higher then for a single DomU. This is caused by different distribution of CPUs among Dom0 and DomUs. For the single DomU case, the distribution is roughly 1:1 and for two DomUs it is 1:2 in favour of processing in DomUs.

CPU loads (Figure 9 and Figure 10) again drop from full utilisation between 600 B and 700 B packet lengths. Note that the maximum value is 200% because of two processors.

TCP throughput reached 456 and 462 Mbit/s for the user domains (918 Mbit/s in total) for IPv6. In comparison, user domains created IPv4 TCP streams of 463 and 466 Mbit/s (929 Mbit/s in total).

4.5 Interactions of Two VMs in VServer

A pair of VMs in VServer accesses the network concurrently and compete for network access (Figure 11 and Figure 12). Throughputs of both VMs are presented to compare fairness. While the overhead of Xen (cf. Section 4.4) is bigger for short packets, it tends to behave more fairly than VServer. Bandwidth is fairly distributed for packets longer than 300 B.

The difference between IPv4 and IPv6 behaviour is not significant.

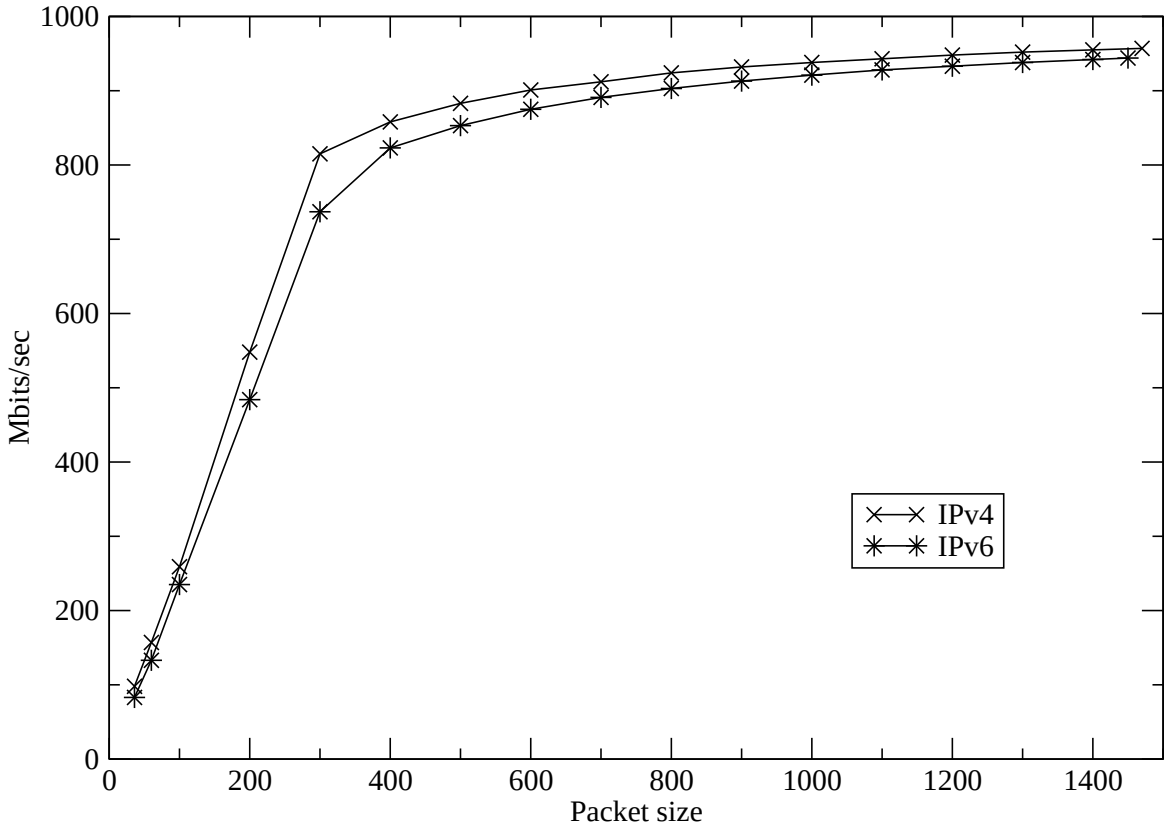

Figure 6. VServer Performance

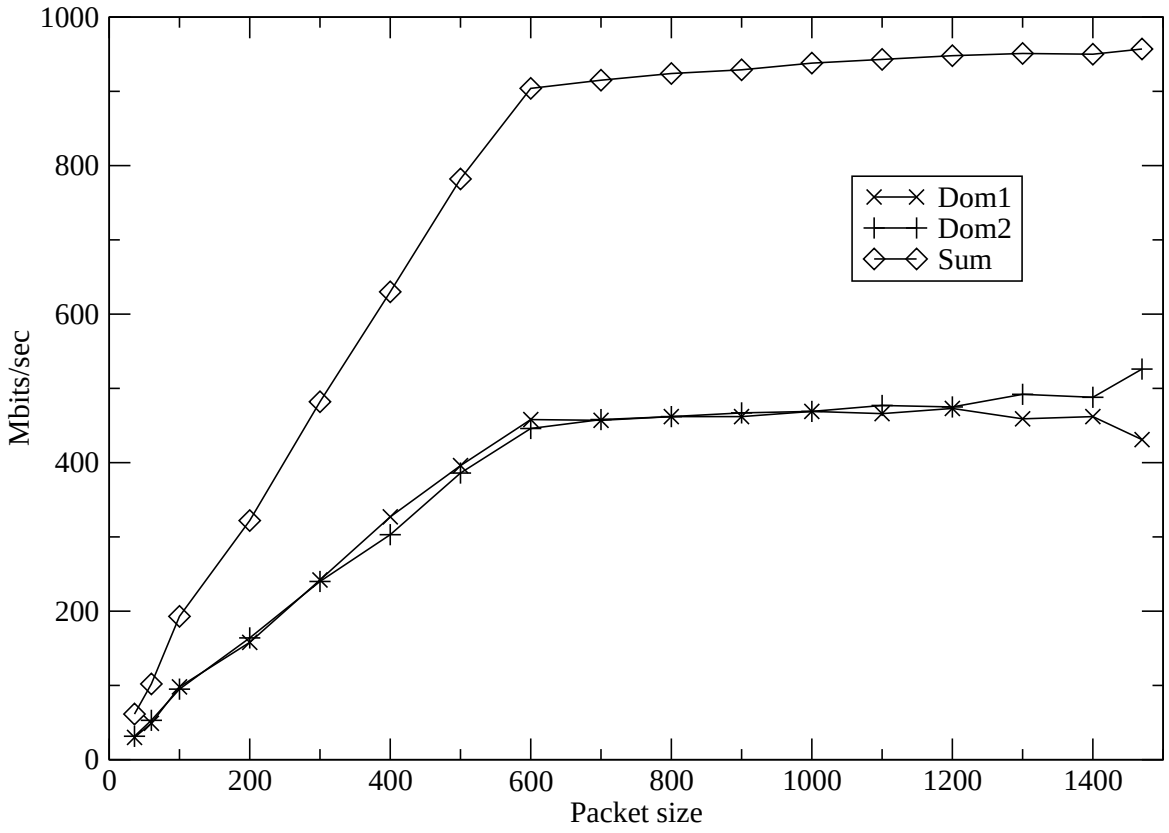

Figure 7. UDP Throughput in Domains for Double DomU Interactions—IPv4

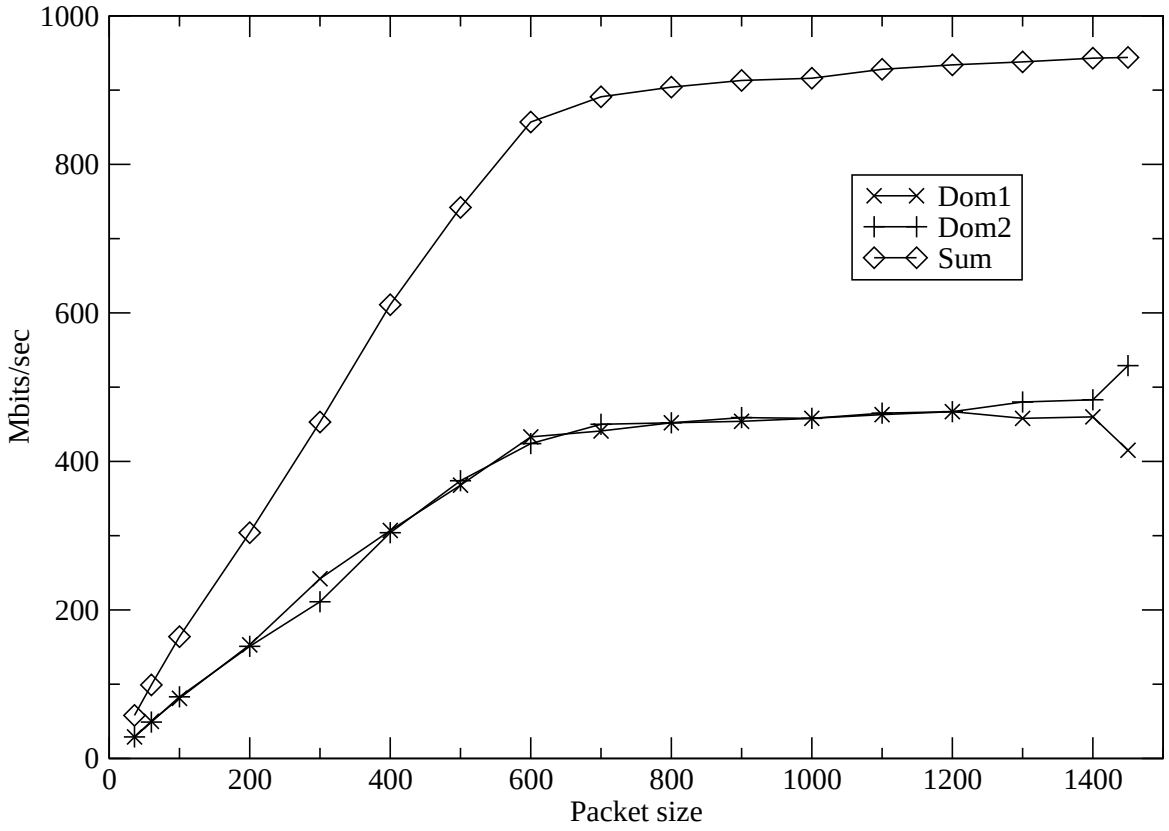

Figure 8. UDP Throughput in Domains for Double DomU Interactions—IPv6

4.6 Effects of Heavy CPU Load in Xen

The purpose of this test was to study how throughput and CPU load behaves when another domain is heavily CPU loaded. One user domain (Dom2) performed the measured network operations, while another user domain (Dom1) was stressed with *burnK7* program from the *cpuburn* utilities.

Cpuburn[5] programs are designed to load CPUs heavily for the purpose of system testing, with the goal to maximise heat production of the CPU, stressing the CPU, motherboard, and power supply. Running *cpuburn* tools utilises the CPU to 100%.

In Figure 13 and Figure 14, we show UDP throughput with another domain heavily CPU loaded in comparison with the throughput in a single DomU (described in Section 4.2) for both protocols. Again, the character of performance loss is similar and does not depend on the protocol.

CPU loads (Figure 15 and Figure 16) are quite fairly shared while the measured networking domain is 100% loaded. For longer packets, the load of networking domain decreases, allowing the burning domain to use more CPU. The increase of networking domain load for longest packets would require further investigation (it is likely to have the same cause as the performance anomaly described in Section 4.2).

TCP throughput reached 903 Mbit/s in this configuration for IPv6 and 895 Mbit/s

5 `http://pages.sbcglobal.net/redelm/`

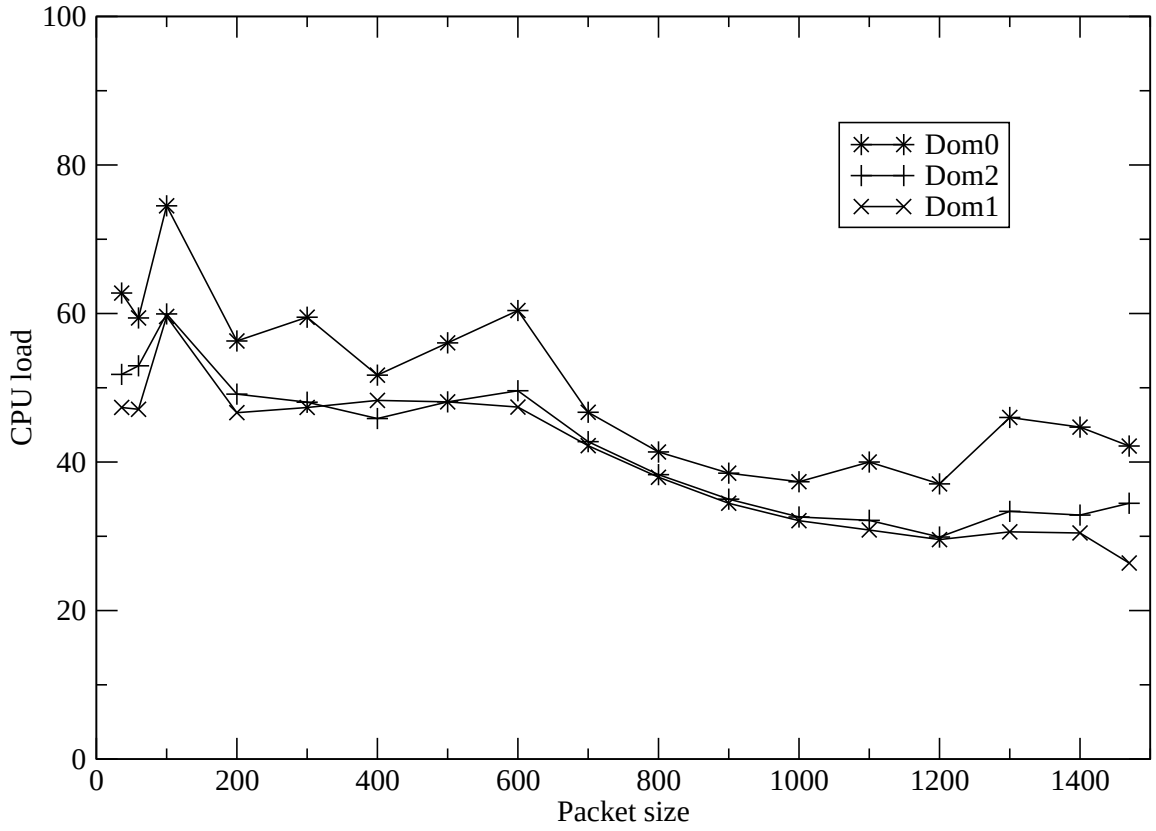

Figure 9. CPU Load for UDP in Domains for Double DomU in Xen—IPv4

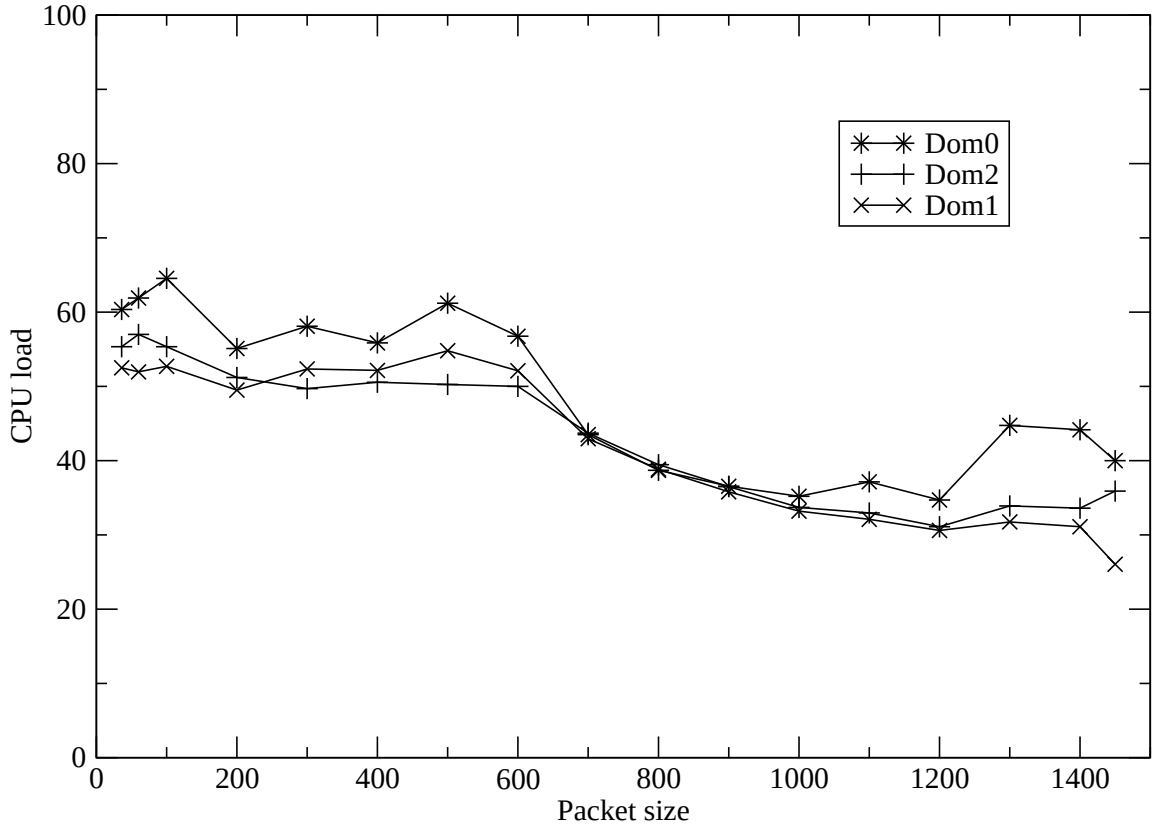

Figure 10. CPU Load for UDP in Domains for Double DomU in Xen—IPv6

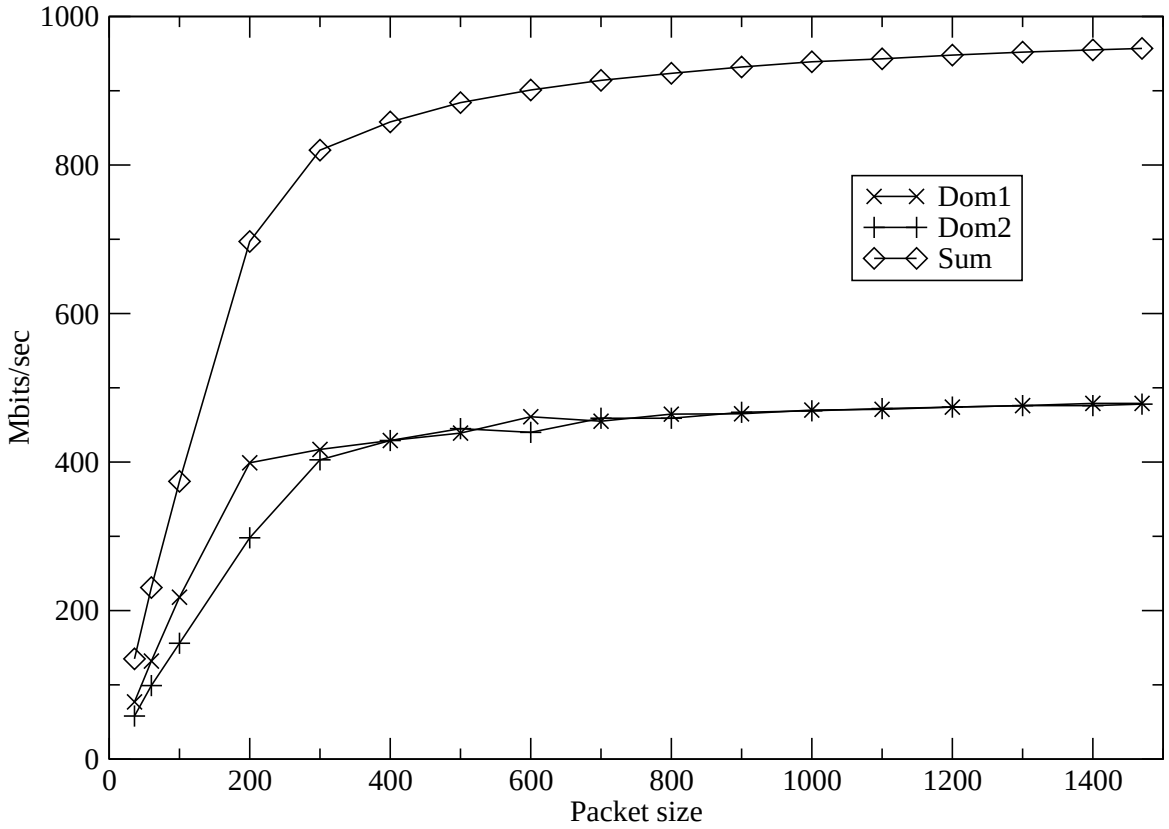

Figure 11. UDP Throughput for Double Domain VServer Interactions—IPv4

for IPv4. The values fluctuated heavily, so the medians give only a rough idea about TCP behaviour and the difference between IPv4 and IPv6 is not statistically important.

4.7 Effects of Heavy CPU Load in VServer

When a VM is heavily CPU loaded (using methods described in Section 4.6) in VServer, it has no effect on network throughput achievable by another virtual machine. This is caused by the fact that each machine uses its own processor. There is no need for a distinguished domain running drivers and network bridging as in Xen.

4.8 Round-Trip Time and Jitter

To the best of our knowledge, round-trip time and jitter results in Xen environments have not been published previously even for IPv4. Although *iperf* is capable of reporting jitter values, this functionality is not very well documented and it is definitely not able to measure round-trip time. We therefore used our own tool, despite its limitations—it is currently not IPv6 ready.

Generator7 was used to measure jitter and round-trip time (RTT) of UDP communication. As synchronisation of time in physical and virtual machines is not easy to manage, we decided to run both generator and receiver processes on machine 2. Machine 1 served as a simple forwarding unit based on the simplest single-thread

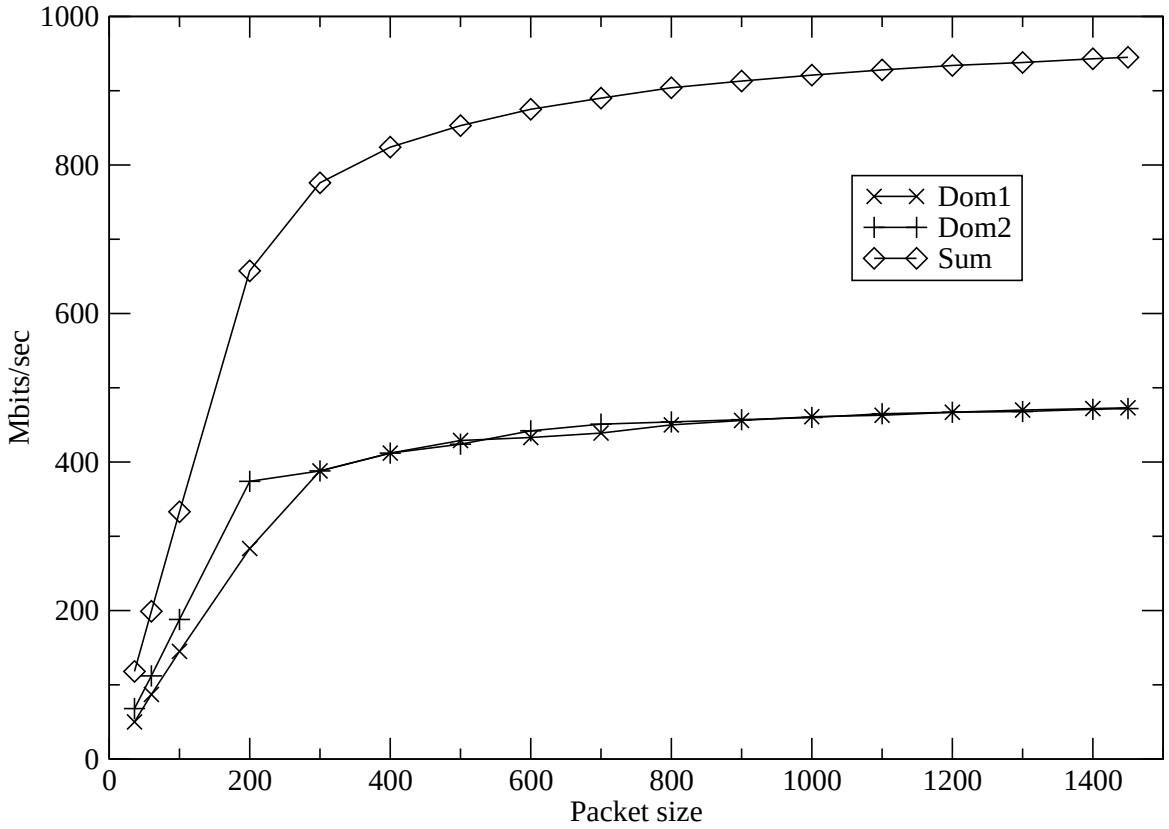

Figure 12. UDP Throughput for Double Domain VServer Interactions—IPv6

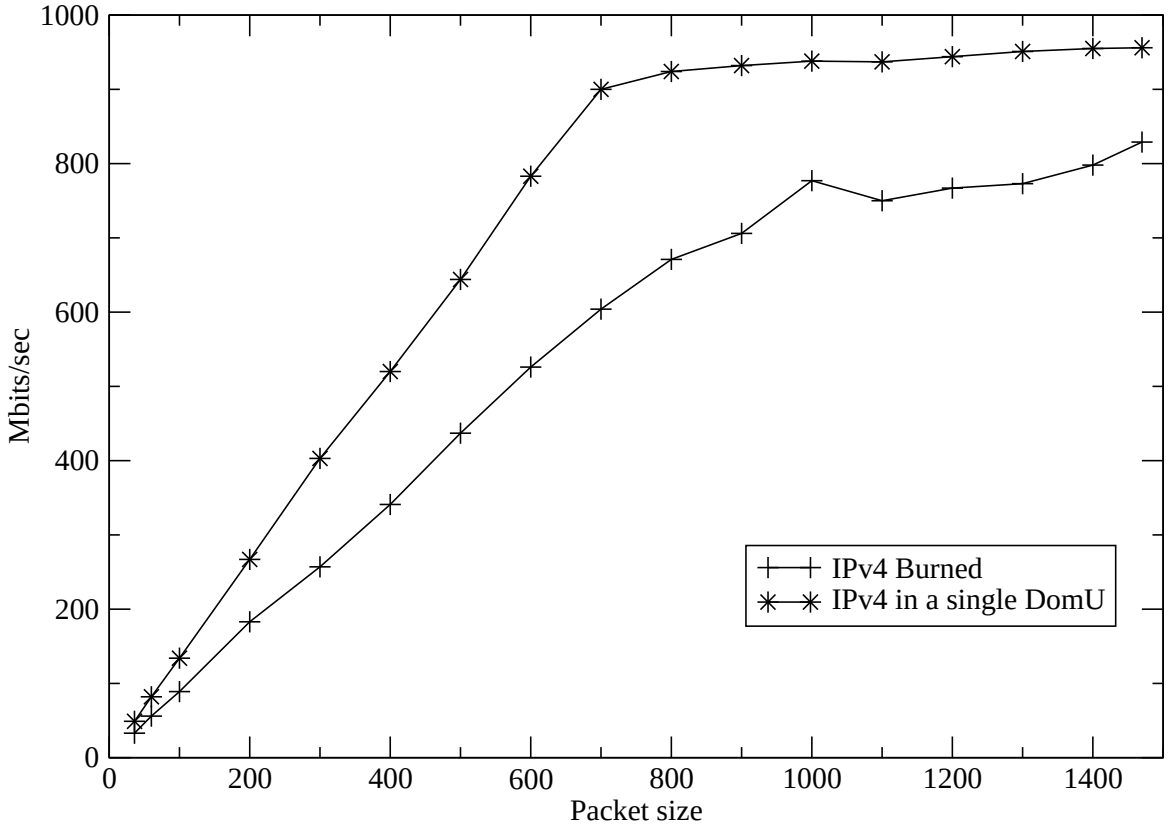

Figure 13. UDP Throughput with Another Domain Burned in Xen—IPv4

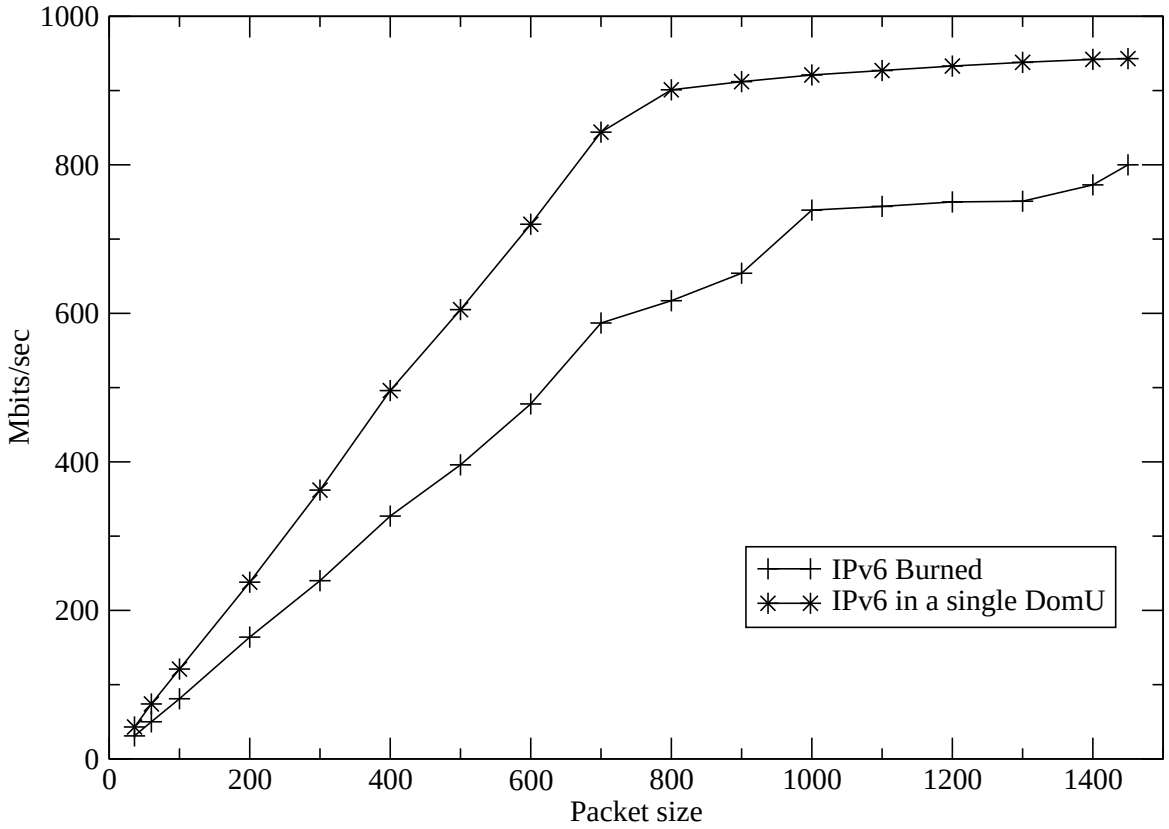

Figure 14. UDP Throughput with Another Domain Burned in Xen—IPv6

packet reflector [4]. The packet reflector is a user-space process that just receives UDP packets in a single loop and sends them back. This setup allows us to measure user space to user space round-trip time.

RTT and jitter was measured on physical machine (Section 4.1), in the single DomU (Section 4.2) configuration, and with another heavily CPU loaded domain (Section 4.6).

Table 2. RTT and Jitter

Configuration	**RTT [ms]**	**st. dev.**	**Jitter [µs]**	**st. dev.**
Native Machine	0.2288	0.0067	13.7420	6.0488
Single DomU	0.2643	0.0019	21.5879	0.5033
CPU-Burned DomU	0.6793	0.1742	21.9621	35.0039

We measured UDP stream of 500 Mbit/s. The results are shown in Table 2. RTT and jitter are median values of 140 measurements. While values for single user domain configuration are decent, values for the heavily CPU loaded domain fluctuate rapidly, as can be seen from its deviation. This is caused by scheduling domains in Xen, which adds significant irregularities in speed of packet processing.

We left obtaining the results for IPv6 as a future work. It requires modifying both *Generator7* and the packet reflector.

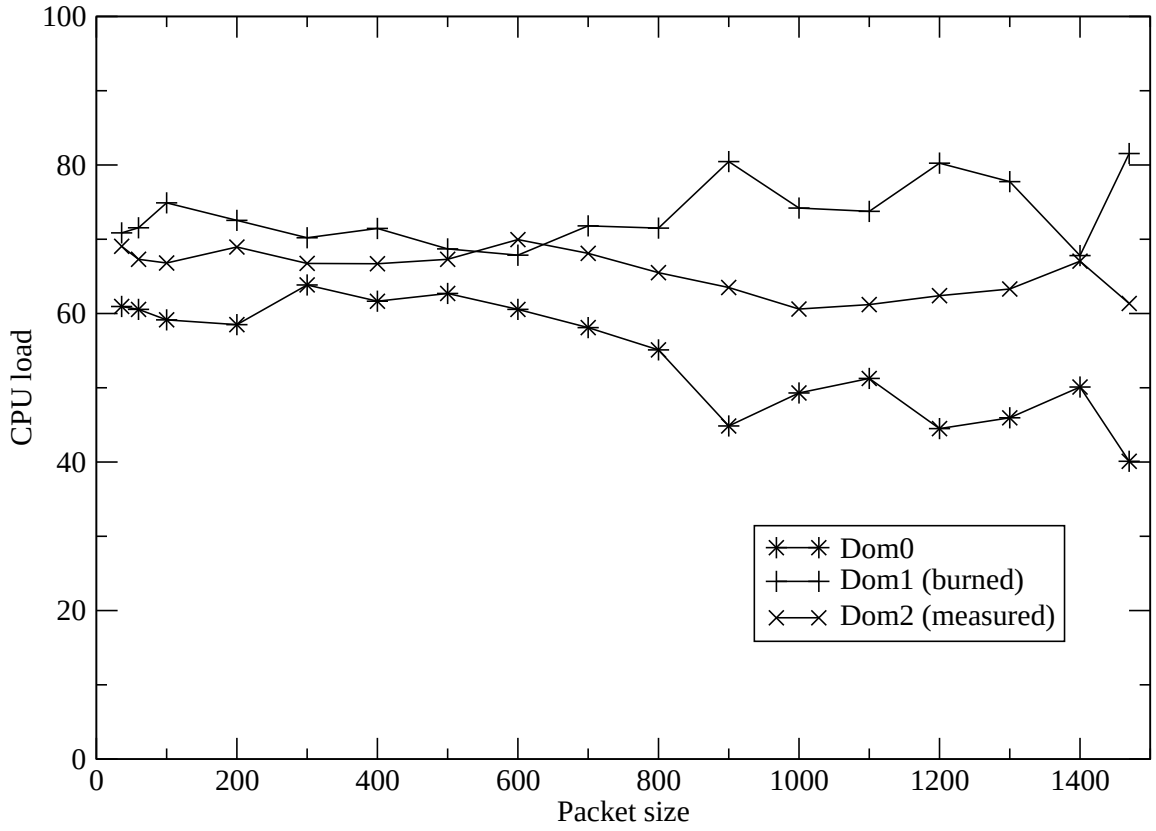

Figure 15. CPU Load—Burned Domain in Xen—IPv4

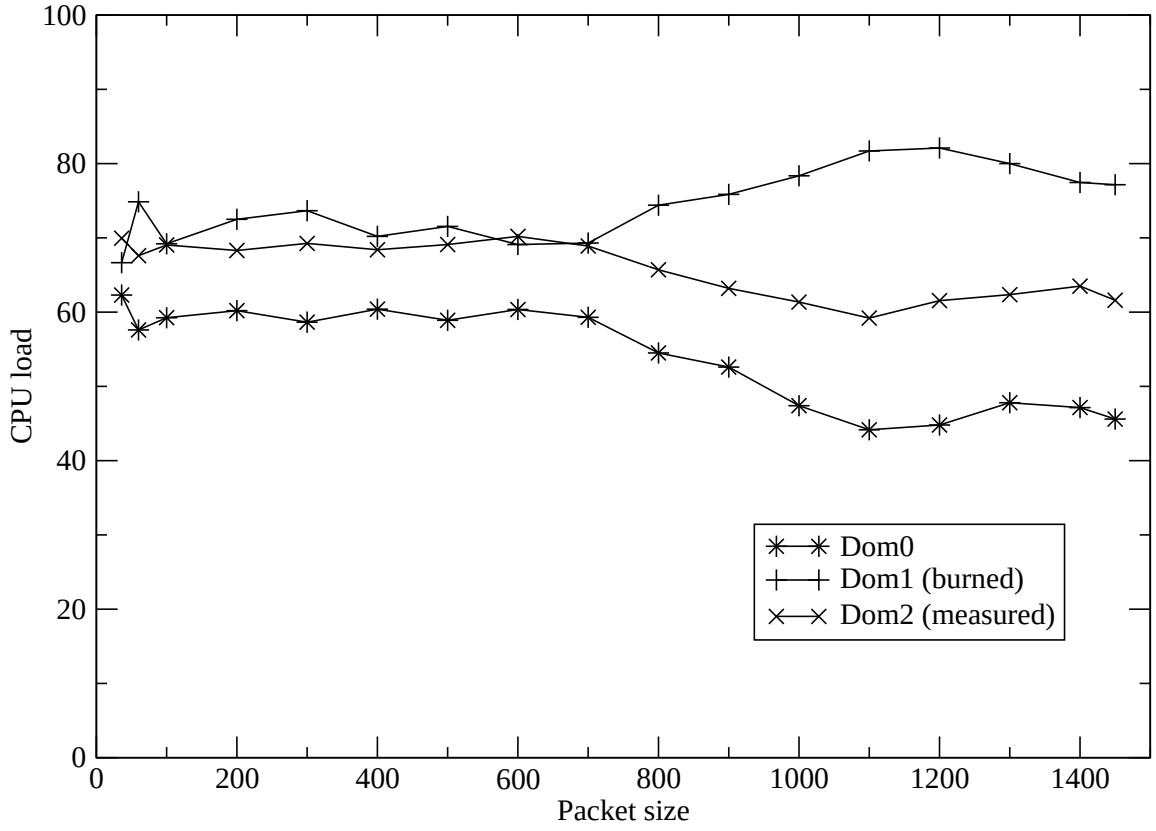

Figure 16. CPU Load—Burned Domain in Xen—IPv6

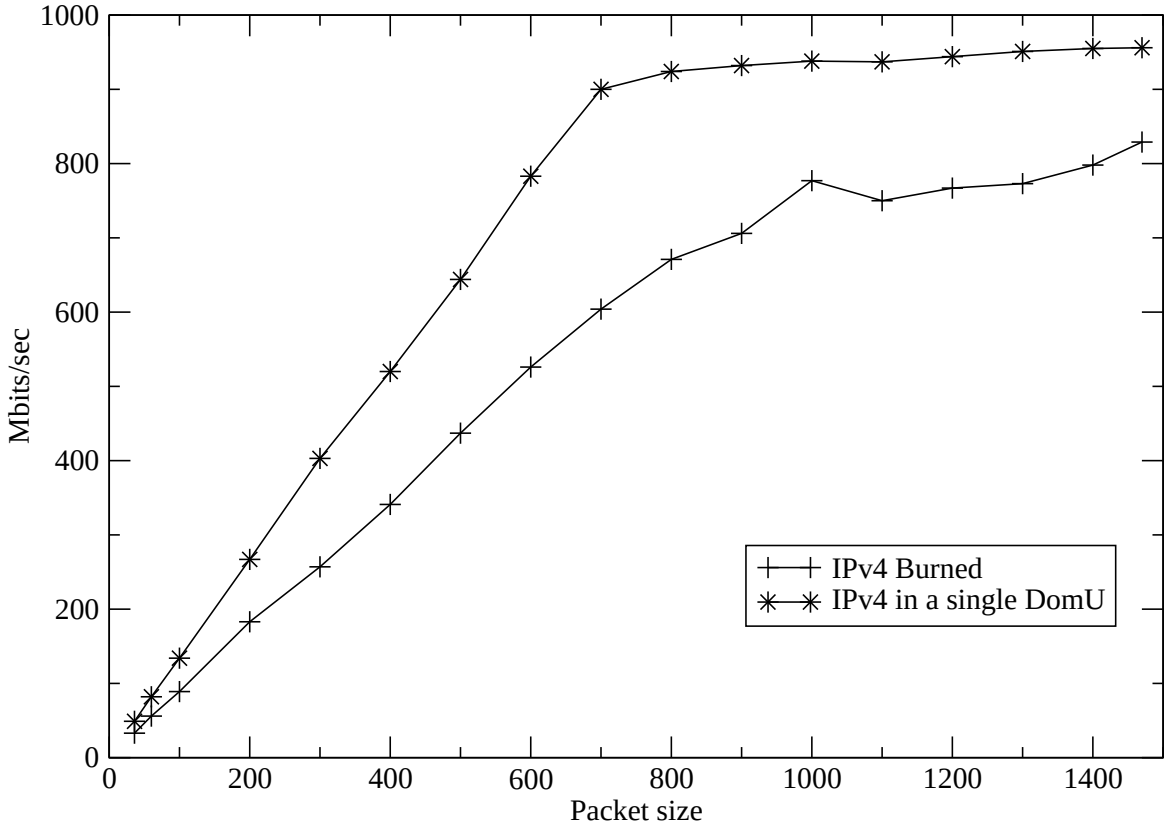

Figure 17. UDP Throughput with Another VM Burned in VServer—IPv4

5 Conclusion

The report shows the effect of the use of IPv6 native addressing mode on the network behaviour in the virtual environment of Xen and VServer and effects of the virtualisation itself.

The measurements from Xen user domains under different workloads show that the IPv6 processing in Xen is mature enough and does not bring any unexpected overhead. The slight difference in IPv6 and IPv4 behaviour in Xen is fully comparable with the difference observed on physical machines. Moreover, the price of the virtual network in Xen itself (i.e., the bridging in Dom0) is completely affordable. Due to character of VServer virtualisation, the performance of VServer is practically identical with a native machine.

The results are encouraging for the deployment of the pure IPv6 based grid infrastructures based on virtualisation approach using virtual machine environment.

As the sets of virtual machines are used to work synchronously, e.g., when processing an MPI job or when used as part of active network elements processing real-time data, we also measured the jitter in two workloads: with and without additional CPU intensive task. Adding CPU load causes the RTT to increase and moreover fluctuate significantly. These results must be taken into account when planning shared simultaneous use of the same physical machine by several virtual machines. The IPv6 behaviour will be studied as part of our future work, when an appropriate measurement environment for IPv6 are available.

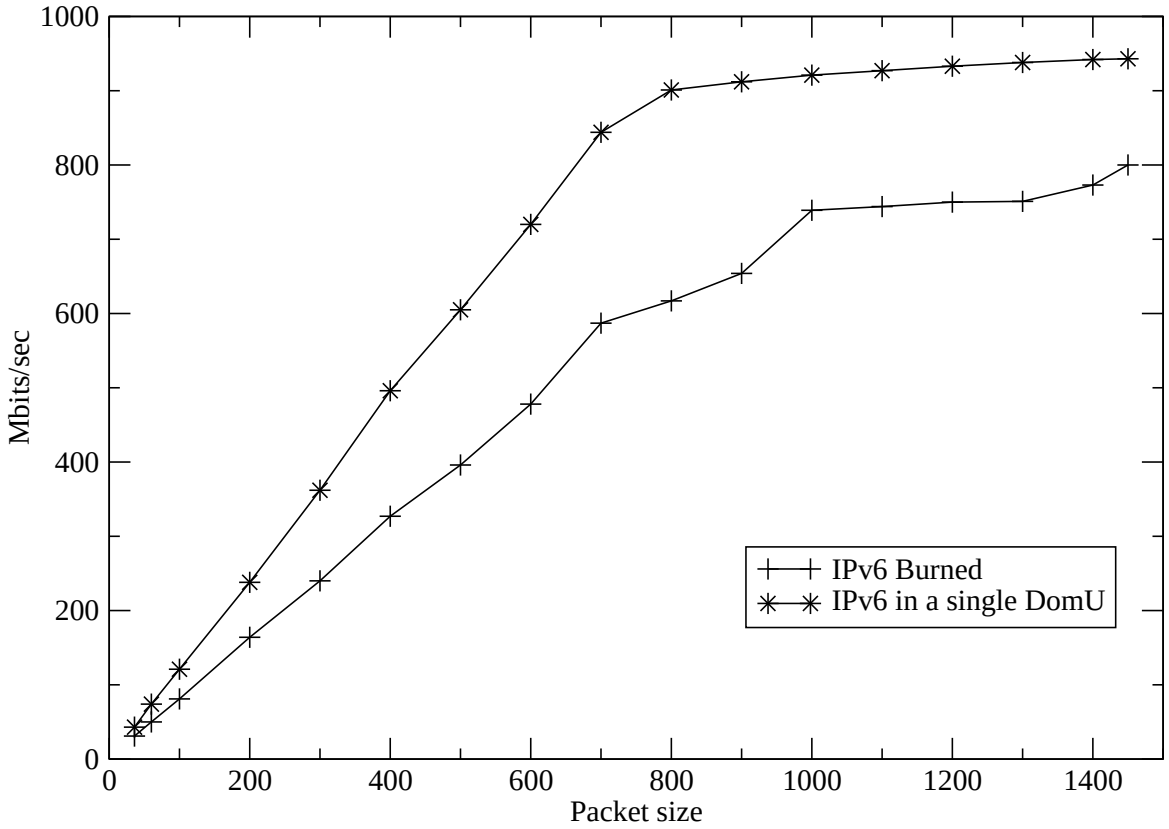

Figure 18. UDP Throughput with Another VM Burned in VServer—IPv6

We also spotted unexpected behaviour in some cases, esp. with large packet sizes, and we will investigate this in our future work.

References

[1] BARHAM, P. et al. Xen and the Art of Virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, p. 164–177, New York: ACM Press, 2003. ISBN 1-58113-757-5.

[2] FISCHBACH, J.; HENDRICKS, D.; TRIPLETT, J. *xentop(1) manual page.* Available online[6].

[3] GUPTA, D.; GARDNER, R.; CHERKASOVA, L. *XenMon: QoS Monitoring and Performance Profiling Tool.* Technical Report HPL-2005-187, Palo Alto: HP Labs, 2005.

[4] HLADKÁ, E. *User Empovered Collaborative Environment: Active Network Support*. PhD thesis, Brno: Masaryk University, 2004.

[5] SCHULZRINNE, H. et al. *RTP: A Transport Protocol for Real-Time Applications.*. RFC 3550[7], IETF, July 2003.

[6] `http://www.die.net/doc/linux/man/man1/xentop.1.html`

[7] `http://www.ietf.org/rfc/rfc3550.txt`