Network Performance Improvement for Cloud Computing using Jumbo Frames

ARJUN REDDY KANTHLA

KTH Information and
Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Network Performance Improvement for Cloud Computing using Jumbo Frames

Arjun Reddy Kanthla

Master of Science Thesis
March 28, 2014

Examiner and Academic Adviser
Professor Gerald Q. Maguire Jr.

Department of Communication Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden.

Abstract

The surge in the cloud computing is due to its cost effective benefits and the rapid scalability of computing resources, and the crux of this is virtualization. Virtualization technology enables a single physical machine to be shared by multiple operating systems. This increases the efficiency of the hardware, hence decreases the cost of cloud computing. However, as the load in the guest operating system increases, at some point the physical resources cannot support all the applications efficiently. Input and output services, especially network applications, must share the same total bandwidth and this sharing can be negatively affected by virtualization overheads. Network packets may undergo additional processing and have to wait until the virtual machine is scheduled by the underlying hypervisor before reaching the final service application, such as a web server.In a virtualized environment it is not the load (due to the processing of the user data) but the network overhead, that is the major problem. Modern network interface cards have enhanced network virtualization by handling IP packets more intelligently through TCP segmentation offload, interrupt coalescence, and other virtualization specific hardware.

Jumbo frames have long been proposed for their advantages in traditional environment. They increase network throughput and decrease CPU utilization. Jumbo frames can better exploit Gigabit Ethernet and offer great enhancements to the virtualized environment by utilizing the bandwidth more effectively while lowering processor overhead. This thesis shows a network performance improvement of 4.7% in a Xen virtualized environment by using jumbo frames. Additionally the thesis examines TCP's performance in Xen and compares Xen with the same operations running on a native Linux system.

Keywords: virtualization, cloud computing, jumbo frame, Xen, TCP

Sammanfattning

Den kraftiga ökningen i datormoln är på grund av dess kostnadseffektiva fördelar och den snabba skalbarhet av datorresurser, och kärnan i detta är virtualisering. Virtualiseringsteknik möjliggör att man kan köra flera operativsystem på en enda fysisk maskin. Detta ökar effektiviteten av hårdvaran, vilket gör att kostnaden minskar för datormoln. Men eftersom lasten i gästoperativsystemet ökar, gör att de fysiska resurserna inte kan stödja alla program på ett effektivt sätt. In-och utgångstjänster, speciellt nätverksapplikationer, måste dela samma totala bandbredd gör att denna delning kan påverkas negativt av virtualisering. Nätverkspaket kan genomgå ytterligare behandling och måste vänta tills den virtuella maskinen är planerad av den underliggande hypervisor innan den slutliga services applikation, till exempel en webbserver. I en virtuell miljö är det inte belastningen (på grund av behandlingen av användarens data) utan nätverket overhead, som är det största problemet. Moderna nätverkskort har förbättrat nätverk virtualisering genom att hantera IP-paket mer intelligent genom TCP- segmenterings avlastning, avbrotts sammansmältning och genom en annan hårdvara som är specifik för virtualisering.

Jumboramar har länge föreslagits för sina fördelar i traditionell miljö. De ökar nätverk genomströmning och minska CPU-användning. Genom att använda Jumbo frames kan Gigabit Ethernet användandet förbättras samt erbjuda stora förbättringar för virtualiserad miljö genom att utnyttja bandbredden mer effektivt samtidigt sänka processor overhead. Det här examensarbetet visar ett nätverk prestandaförbättring på 4,7% i en Xen virtualiserad miljö genom att använda jumbo frames. Dessutom undersöker det TCP prestanda i Xen och jämför Xen med samma funktion som körs på en Linux system.

Nyckelord: virtualisering, datormoln, jumboram, Xen, TCP

Acknowledgments

My sincere gratitude to my supervisor Professor Gerald Q. Maguire Jr. for giving me an opportunity to work under him and for his very best support throughout the thesis work. His enthusiasm and patience in clearing my doubts and providing suggestions is incalculable. Simply I found the epitome of a teacher.

Special thanks to my programme coordinator May-Britt Eklund-Larsson, for her gracious support throughout my graduate studies.

Finally, I would like to thank my family and my friend Pavan Kumar Areddy for their support during the thesis work.

Contents

Abstract i

Sammanfattning iii

Acknowledgments v

List of Figures ix

List of Tables xi

List of Acronyms and Abrreviations xiii

1 Introduction 1
- 1.1 Problem Statement . . . 2
- 1.2 Goals . . . 3
- 1.3 Structure of the Report . . . 4

2 Background 5
- 2.1 Cloud Computing . . . 5
  - 2.1.1 Infrastructure as a Service (IaaS) . . . 5
  - 2.1.2 Platform as a Service (PaaS) . . . 6
  - 2.1.3 Software as a Service (SaaS) . . . 6
- 2.2 Virtualization . . . 6
  - 2.2.1 Types of virtualization . . . 8
- 2.3 Virtualization Technologies . . . 9
  - 2.3.1 Xen . . . 9
  - 2.3.2 OpenVZ . . . 9
  - 2.3.3 Kernel-based Virtual Machine (KVM) . . . 9
- 2.4 Xen Hypervisor . . . 9
  - 2.4.1 Scheduling Mechanism in Xen . . . 11
- 2.5 MTU and Jumbo Frames . . . 12
- 2.6 Transmission Control Protocol . . . 14

2.7 Related Work 16
2.7.1 Work on schedulers 16
2.7.2 Work on MTU and network performance 17

3 Methodology **19**
3.1 Workloads and Tools 20
3.1.1 Iperf 20
3.1.2 TCPdump 21
3.1.3 httperf 22
3.1.4 Additional tools 23
3.2 Measurement Metrics 23
3.2.1 Network Throughput 24
3.2.2 Network Latency 24
3.2.3 CPU utilization 24
3.3 Experimental Setup 26
3.3.1 Bridging 27

4 Evaluation and Results **29**
4.1 Throughput 31
4.2 CPU Utilization 33
4.3 Throughput at the client 35
4.4 Additional Measurements 36
4.4.1 Xen Performance Comparison 36
4.4.2 TCP Behavior in Virtual Machine 37
4.5 Analysis and Discussion 39

5 Conclusions and Future Work **43**
5.1 Conclusions 43
5.2 Future Work 44
5.3 Required Reflections 44

A Configuration **55**

B Data Files **61**

C Issues **67**

List of Figures

1.1 Extra layers of processing 3

2.1 Types of hypervisors 8
2.2 Architecture of Xen hypervisor 10
2.3 Credit Scheduler 11
2.4 Standard and jumbo Ethernet Frames 12
2.5 TCP Header with data 14
2.6 TVP settings in a running Linux 16

3.1 Iperf server 20
3.2 Iperf client 21
3.3 Example of tcpdump output 22
3.4 Pictorial Representation of throughput and bandwidth for a physical link 24
3.5 Example of pidstat output 25
3.6 Experimental Setup 26
3.7 Screen-shot showing Dom0 and two running VMs 27
3.8 Linux Bridging 28
3.9 Linux bonding with bridge 28

4.1 NIC features enabled 30
4.2 Network protocol stack with Iperf and TCPdump 30
4.3 Virtual Machine and Dom0 Throughput 31
4.4 Throughput observed to decrease from 6000 bytes MTU 33
4.5 CPU usage of Netback service in Xen 34
4.6 Throughput seen at client 35
4.7 Xen Performance compared to native Linux system 37
4.8 Sequence of 1500 byte MTU packets in Dom0 38
4.9 Sequence of 1500 byte MTU packets in VM 38
4.10 Sequence of 5000 byte MTU packets in Dom0 38
4.11 Sequence of 5000 byte MTU packets in VM 38

4.12 Sequence of 9000 byte MTU packets in Dom0 39
4.13 Sequence of 9000 byte MTU packets in VM 39
4.14 Incongruency of abstraction layering concept 41

A.1 Processor Details 55
A.2 Xen Hypervisor Details 56
A.3 NIC Details 57
A.4 Other NIC Details 58

List of Tables

2.1 MTU size and Ethernet speeds 13
2.2 Overhead comparison of standard and jumbo frames 14

3.1 Bonding Modes 28

4.1 Performance gain of virtual machine 32
4.2 Performance gain in Dom0 32
4.3 Average Throughput over 10 seconds 32

List of Acronyms and Abrreviations

μs	Microseconds
ACK	acknowledgement
BDP	Bandwidth Delay Prodouct
BW	Bandwidth
CPU	Central Processing Unit
GbE	Gigabit Ethernet
I/O	Input and Output
IaaS	Infrastructure as a Service
IP	Internet Protocol
iSCSI	Internet Small Computer System Interface
IT	Information Technology
Mbps	Megabits per second
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NAS	Network-attached storage
NFS	Network File System
NIC	Network Interface Card
OS	Operating System
PaaS	Platfrom as a Service
PCI	Peripheral Component Interconnect

RFC	Request for Comments
RTT	Round Trip Time
SaaS	Software as a Service
SLA	Service Level Aggrement
SMP	symmetric multiprocessing
TCP	Transimssion Control Protocol
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
VM	Virtual Machine
VMM	Virtual Machine Monitor
VT	Virtualization Technology

Chapter 1

Introduction

Cloud Computing has become an essential part of the information technology infrastructure in the recent years. Cloud computing offers hardware resources and software services to users without requiring that the users actually own these resources. Some advantages for adopting cloud services are reduction in capital investments, hassle free maintenance, increased reliability, etc. However, the core advantages are flexibility, elasticity, and scalability as processing can be scaled up or down according to the user's needs. Cloud computing offers cost effective benefits in many fields, including but not limited to scientific processing, big data collection, rendering images for the entertainment industry, etc.

A prime reason for the proliferation of cloud computing is virtualization technology (VT), which enables the computer's owner to fully utilize the computer. Modern symmetric multiprocessing (SMP) processors are frequently idle and virtualization exploits this property to enable server and application consolidation by running multiple concurrent Operating Systems on a single physical processor. However, scaling the resources according to the user's needs and meeting a Service Level Aggrement (SLA) with this user is crucial in successfully exploiting VT.

Many studies have been done to understand the affects on the network of virtualization and many solutions have been proposed to reduce the network overhead on the Central Processing Unit (CPU), for example by performing part of the processing in the network interface itself. However, very few have studied the effects of jumbo frames in a virtual environment. The primary motivation to study jumbo frames is that, they are already available (i.e., already implemented by network interfaces) and no new software or hardware is necessary to make use of them. The open question is if using them can actually enhance the performance of VT.

1.1 Problem Statement

Ideally applications running in a virtual environment must run independent of each other, i.e., the application's performance should not be affected by other running applications. Unfortunately, this is not true as concurrently running applications do affect one another, inturn effecting both their individual and collective performance. Performance isolation is a major challenge when designing and implementing virtualization. In a SMP server, the negative performance impact on another application, running on a different core is called *cross-core interference* [1].

There are many factors affecting the performance of the applications running in a Virtual Machine (VM). Performance depends on the application type (whether it is Input and Output (I/O) intensive or Central Processing Unit (CPU) sensitive) and the scheduling mechanisms used within the hypervisor. Usually data-centers disallow latency sensitive and CPU sensitive applications being co-located. According to Gupta, et al. [2] achieving performance isolation requires good resource isolation policies. Email, web-search and web-shopping are I/O sensitive applications, while image rendering, data computations and file compression are processing intensive (require many CPU cycles). Paul, Yalamanchili, and John [3] showed that, suitable deployment of VMs reduces the interference between the actions of different VMs.

Although the latest generation of SMP processors are capable of large amounts of computing in a short period of time, today's high speed networks can quickly saturate these processors, thus the CPU's capacity is the bottleneck. As a result network resources can be underutilized. In a virtualized server with multiple network applications the load on the CPU is more compared to that of a traditional server. As Mahbub Hassan and Raj Jain [4] state:

> *"On the fastest networks, performance of applications using Transimssion Control Protocol is often limited by the capability of end systems to generate, transmit, receive, and process the data at network speeds."*

Virtualization of networking typically introduces additional layers of packet processing in the form virtual bridges, virtual network interfaces, etc. As shown in Figure 1.1 and according to Tripathi and Droux [5], fair sharing of the physical **network** resource among the virtual network interfaces is a primary requirement for network virtualization.

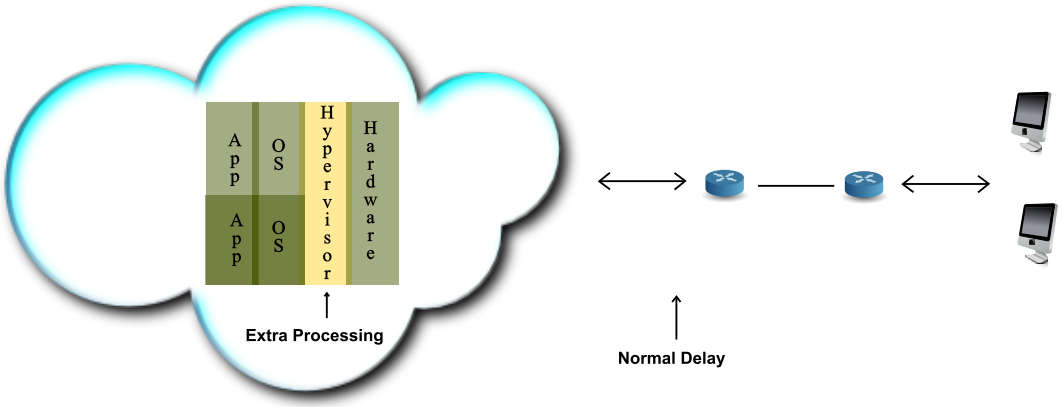

Figure 1.1: *Extra layers of processing*

Although many studies have been done to minimize the CPU load in a virtual environment, by measuring performance and optimizing the code, but very few researchers have examined what effects can occur when increasing the Maximum Transmission Unit (MTU). Jumbo frames, as will be discussed in section 2.5, are not currently being utilized in many virtual environments.

As the frame size increases, the same amount of user data can be carried in fewer frames. Utilizing larger frames requires in **less** CPU overhead, which is desirable. Furthermore, utilizing large frames is a great opportunity for VT to exploit the capacity offered by Gigabit Ethernet (GbE) and to decrease the network overhead and decrease the load on CPU, while at the same increasing the application's **effective** throughput. In the light of all these factors, the goal of this project is to study how much gain in effective throughput is possible when using jumbo frames rather than standard Ethernet frames in a virtualized environment. The second question is how much the load on the CPU can be reduced by utilizing large frames.

1.2 Goals

The initial idea was to test how two competing VMs affect one another with respect to their performance and bandwidth (BW). As the project progressed, the focus shifted to studying the effects jumbo frames, as inspired by [6, 7]

and others. It was clear that using jumbo frames has benefits in a standard physical environment. Thus, the goal was to study the affects and benefits of jumbo frames in a virtual machine environment.

The main goal of the thesis is to study the benefits of using jumbo frames in a virtual machine environment. Thus the subgoals were to quantify how much performance improvement can be achieved using jumbo frames and how much CPU load (associated with networking protocol stack processing) can be reduced by sending large frames instead of standard sized Ethernet frames (which for the purpose of this report are assumed to be limited to a MTU of 1500 bytes).

1.3 Structure of the Report

The rest of the thesis is structured as follows:

Chapter 2 introduces the basic concepts of cloud computing, virtualization, and summarizes some open source technologies that are relevant. The chapter also describes the Xen hypervisor, one is of the popular open source hypervisors used to realize VT. This is followed by a description of jumbo frames and their benefits. The Transimssion Control Protocol is a complex protocol and some of the most important parts of this protocol are described in the section 2.6. The chapter concludes with a summary of related work .

Chapter 3 begins with a description of the methodology that was applied, then explains the tools and workloads that were utilized for this research. This is followed by an explanation of the metrics used for the evaluation. The chapter finishes by describing the experimental set up used for all the measurements.

Chapter 4 presents all the measurements in the form of visual representations (graphs), rather than as numeric data (detailed numeric data is included in an appendix). The last section of this chapter discusses the benefits of jumbo frames from a holistic viewpoint.

Chapter 5 concludes the thesis with a conclusion, then suggests some future work and finally ends with some the reflections on the project in a broader context.

Chapter 2

Background

This chapter lays the foundation for the rest of the thesis. It begins by introducing cloud computing, then briefly explains the different types of cloud computing services. This is followed by a detailed description of virtualization, followed by a discussion of the Xen hypervisor. Next the TCP protocol and MTU concepts are explained. The final section in this chapter discusses related work.

2.1 Cloud Computing

Cloud computing according to Armbrust, et al. includes both the hardware resources and software services offered by a data center [8]. If these services can be accessed by public (who can be charged based upon their usage), then it is called a *Public Cloud*. Some of the companies offering these services are Amazon, Google, Microsoft, and Heroku. In contrast, a *Private Cloud* can only be accessed by one organization. Additionally, there are *Hybrid Clouds* that mix both public and private clouds. A number of sources (such as [8, 9, 10]) explain cloud computing in detail and also from an economic perspective. Depending on the type of service and the level of the administrative access, cloud computing is classified into many sub-classes. The following subsections describe three of the most common sub-classes.

2.1.1 Infrastructure as a Service (IaaS)

IaaS cloud providers offer physical or virtual hardware. Amazon Elastic Compute Cloud (EC2) is one such commercial public IaaS cloud. Users can buy computing capacity according to their needs and scale the amount of resources that they utilize as required. Customers can build any kind of software and have full control over this software, but they do not know exactly what the

underlying hardware over which this software runs. EC2 uses Xen virtualization (see section 2.4).

2.1.2 Platform as a Service (PaaS)

PaaS cloud providers offer a particular platform as a service. Customers have to use the specific software provided, to build their applications and cannot use any software which is inconsistent with this platform. As a result customers have to selectively choose their cloud service provider based upon the application they are going to write and use. Google App Engine [11] is an example of PaaS, currently it offers only a few programming languages such as, Java, Php. Heroku [12] is another PaaS provider, but they support Ruby, Python, and several other languages.

2.1.3 Software as a Service (SaaS)

SaaS directly provides the applications which users can use. These applications have already been built (for specific needs). Typically users access these applications using a browser and their data is stored in the provider's servers. Examples of SaaS are email (such as Google's gmail), salesforce.com and Dropbox.

2.2 Virtualization

The genesis of mainstream virtualization technology [13, 14, 15, 16] dates back to 1972, when IBM first introduced in its System 370 server commercially. Virtualization is a framework in which one or more Operating Systems (OSs) share same physical hardware. Modern computers (such as servers) are frequently idle and are powerful enough to run multiple OSs on a single physical machine. Virtualization leverages the use of hardware, while reducing costs and carbon footprint. Running multiple OSs on single physical machine also helps a data center to achieve server consolidation. Since certain applications are compatible only with certain OSs, there are some limitations in combinations of applications and OS. For example, Microsofts's Windows Server can only be installed on top of a Microsoft OS. As a result by running both Windows and Linux OSs on same physical machine, modern multicore processors can be utilized efficiently and applications can still run at near native speed, without the need to run two or more separate computers each running only a single OS.

A *Hypervisor* or Virtual Machine Monitor (VMM) is a software layer, which presents an abstraction of the physical resources to an OS installed on top of this VMM. Traditional OS perform context switching between applications, without the applications being aware of this context switching, whereas a *VMM performs context switches between two or more VMs without the applications in these VMs being aware of this VM context switching.* The OS running in a VM is called a **virtual OS** or a **guest** or simply an VM instance. The hypervisor's main functions are scheduling, memory allocation to each guest, and virtualization of the underlying hardware. The hypervisor runs in a privileged mode and guest the OSs run in the user mode (or another unprivileged mode). Guest OSs do not have direct access to the hardware, instead the hypervisor schedules all jobs and assigns the necessary physical resources by some scheduling mechanism. This is achieved by trapping the guest OS's instructions and processing these instructions in the hypervisor. After the hypervisor executes the instruction the result is return back to the guest OS [17] [18].

In a traditional environment once an OS is installed, the drivers for the hardware devices are also installed into this OS. Since an OS installed inside a VM never directly accesses the underlying physical hardware it is possible to move the guest OS to an other physical machine. A VM makes use of virtual resources provided by the underlying physical machine. These virtual resources include one or more virtual CPUs (vCPUs), virtual network interface cards (vNICs), etc. Just as the device drivers loaded into an OS can be optimized based upon the physical hard device that is present, the device drivers loaded into a guest OS can be optimized for the virtual resource that the hypervisor presents to the guest instance.

A hypervisor that is installed directly above the hardware is called a **Type 1** hypervisor, bare-metal, or native hypervisor. Such a hypervisor has complete control over the hardware. A **Type 2** hypervisor is installed as a regular application inside a host OS, while all the hardware control is controlled by the host OS. This is illustrated in Figure 2.1. It is also possible in some VMMs to give a gust OS direct access to the underlying hardware, for example by mapping part of a devices Peripheral Component Interconnect (PCI) address space into the guest OS.

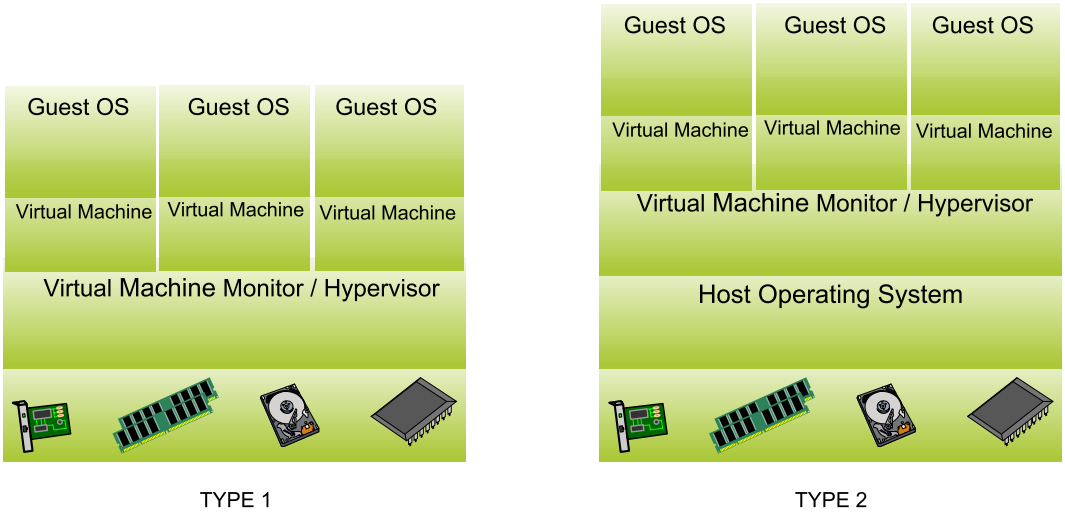

Figure 2.1: *Types of hypervisors*

2.2.1 Types of virtualization

Depending on how the hardware is virtualized, virtualization can be broadly classified into three types:

Full Virtualization In full virtualization, the hypervisor virtualizes the entire set of hardware. As a result the guest OS is unaware of being virtualized and believes that it controls all of hardware. In this case, the guest OS can be installed without any modification. However, the performance of a fully virtualized system is low compared to other types of virtualization, since all of the hardware is virtualized.

Paravirtualization In paravirtualization, the guest OS is cognizant that it is residing on virtualized hardware. In this approach the guest OS must be modified (ported) in order to be installed on the hypervisor. Some hardware is exposed to the guest and this results in increased performance. For example, in this approach all but a small set of instructions can be directly executed by the guest OS and its application, while some instructions cause a trap which invokes the VMM.

Hardware-assisted Virtualization The surge in VT has prompted vendors to manufacture hardware specifically designed to support virtualization. Hardware components such as processors and Network Interface Card (NIC) are being manufactured to assist or compliment VT. Virtualization at the hardware level gives a great boost to the performance. Technologies such as Intel's VT [19, 20] and Single Root Input and Output virtualization

(SR-IOV) is a new feature added to PCI devices, by which the I/O virtualization overhead is significantly reduced [21].

2.3 Virtualization Technologies

There are many hypervisors available, some of them are open source and some are proprietary. A few of them are described below. The following subsections describe three of the most common open source hypervisors.

2.3.1 Xen

Xen is a open source VMM for the x86 architecture, first developed at the University of Cambridge Computer Laboratory [22]. Xen has wide industry support and many cloud providers use Xen to virtualize their servers, such as Amazon EC2, Citrix Systems, and Rackspace. Xen can be implemented in both either in paravirtualization or full virtualization modes.

2.3.2 OpenVZ

OpenVZ is a hypervisor which is built into the Linux kernel. As it is built into the kernel, only a Linux based OS can be installed. VMs are called Linux Containers. It has less overhead than Xen, as there no separate (hypervisor) layer. This type of virtualization is called OS-level virtualization. Oracle's Solaris zones is another example of OS-level virtualization.

2.3.3 Kernel-based Virtual Machine (KVM)

KVM requires hardware assisted virtualization [23] i.e, to install KVM the underlying processor must have virtualization capability. In addition to the open source hypervisors using hardware assisted virtualization, a number of commercial products, such as VMware, Inc.'s VMware ESX server and Microsoft's Hyper-V. VirutalBox supports hardware assisted virtualization and paravirtualization and exists as both open source and as a proprietary product from Oracle.

2.4 Xen Hypervisor

Xen is popular for paravirtualization and provides good isolation among guest OSs. The current stable release is Xen 4.3. As stated above in order to implement paravirtualization, a guest OS must be modified. This gives

performance benefits, but limits the choice of guest OSs. Whereas, full virtualization allows us to choose any OS, but at the cost of increased overhead. Figure 2.2 shows the paravirtualized architecture of the Xen hypervisor, showing the main tasks of the hypervisor as scheduling and memory management. In paravirtualization, I/O virtualization is handled by a privileged domain called driver domain the (Dom0). Dom0 has direct access to the I/O devices and I/O traffic must flow through Dom0. Dom0 is also a guest, but has privileged access to the underlying hardware.

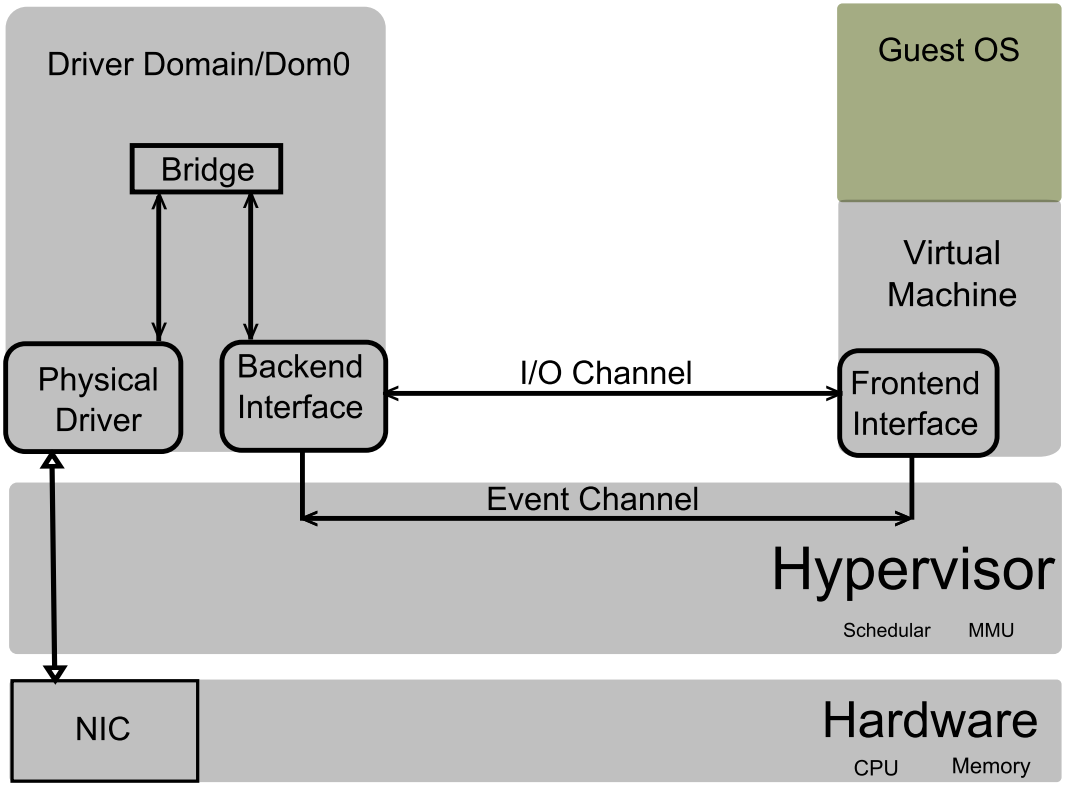

Figure 2.2: *Architecture of Xen hypervisor*

Each guest OS (DomU in Xen terminology) has one or more virtual frontend network interface and Dom0 has corresponding virtual backend network interfaces. Each backend network interface is connected to a physical network interface through a logical bridge. *Hypercalls* are software traps from a domain to the hypervisor. These hypercalls are analogous to system calls used by applications to invoke OS operations. *Event Channels* are used to communicate event notifications to/from guest domains. These events are analogous to hardware interrupts.

2.4.1 Scheduling Mechanism in Xen

Scheduling is a mechanism by which a process or job is assigned system resources. The actual scheduling is handled by a scheduler. Usually there are many processes to be executed on a computer, a scheduler decides which process to run next (from those processing in the run queue) and assigns a CPU for some time for this process. Xen allows us to choose the appropriate scheduling mechanism depending on our needs.

The **Credit** scheduler is a pre-emptive* and proportional share scheduler and is currently Xen's default scheduler. Ongaro, Cox, and Rixner [24] give details of the Xen credit scheduler. Each domain is given a Weight and a Cap. A domain with higher weight gets more CPU time. By default a domain is given a weight of 256 [25], as shown in Figure 2.3. The Cap function limits the maximum CPU capacity a domain can consume, even if the system has idle CPU cycles. The Schedule Rate Limiting is a feature added to the credit scheduler, by which a VM is given a minimum amount of CPU time without the possibility of being preempted. The minimum time by default is one millisecond (ms). Another VM with higher priority is denied the CPU until the currently running process has had its one ms of CPU time. We can change the *ratelimitus* value to suit the type of applications running on a particular VM. If latency sensitive applications are running, then VMs can be assigned a μs factor, so that VMs are scheduled frequently.

`xm sched-credit`

```
Name                                ID Weight  Cap
Domain-0                             0    256    0
vm1                                       256    0
vm2                                  4    256    0
vm3                                  1    256    0
```

Figure 2.3: *Credit Scheduler*

The **Simple Earliest Deadline First (SEDF)** scheduler uses real-time algorithms to assign the CPU. Each domain is given a Period and a Slice. A period is a guaranteed time unit of CPU time to be given to a domain and a

*In a preemptive scheduler the running process is stopped, if there is any other process with a higher priority in the run queue.

slice is a time per period that a domain is guaranteed to be able to run a job (without preemption). Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat have compared various schedulers in [26].

2.5 MTU and Jumbo Frames

The Maximum Transmission Unit (MTU) is the maximum amount of payload a data-link frame can carry. A **jumbo frame** [27] can be defined as an Ethernet frame carrying more than 1500 bytes of payload. This includes all of the upper-layer headers and application data. In contrast, a standard Ethernet frame is restricted to carrying only a payload of 1500 bytes. Gigabit Ethernet (GbE), as standardized in IEEE 802.3ab standard [28], is capable of carrying more than 9000 bytes of payload. However, jumbo frames have never been standardized, because of compatibility issues and because vendors potentially need to change their equipment. Today GbEs are becoming a common network interface even for personal computers and laptops. Figure 2.4 shows both standard and jumbo frames with TCP and IP headers (in both cases assuming no options are being used).

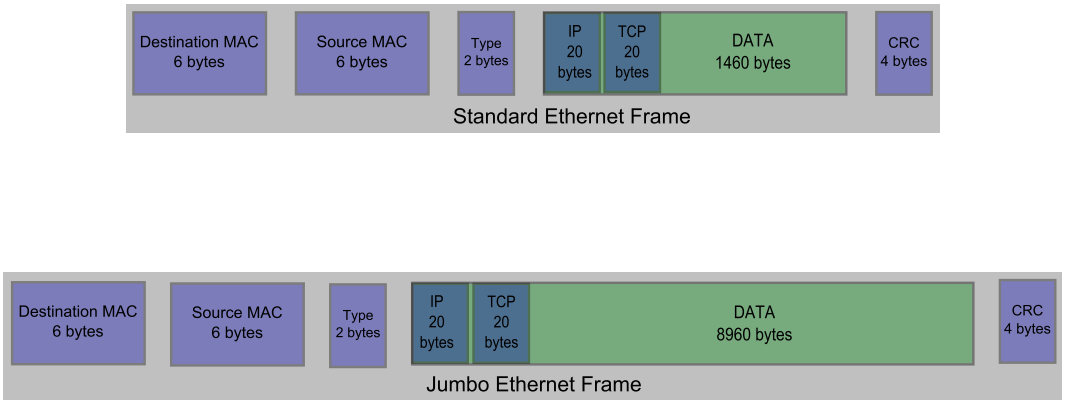

Figure 2.4: *Standard and jumbo Ethernet Frames*

Table 2.1 summarizes the amount of time required to send a frame, as seen wire-time has decreased for different versions of Ethernet, but MTU has been relatively constant.

Table 2.1: *MTU size and Ethernet speeds*

Ethernet Technology	Rate	Year	Wire Time	MTU
Ethernet	10 Mbps	1982	1200 μs	1500
Fast Ethernet	100 Mbps	1995	120 μs	1500
Gigabit Ethernet	1 Gbps	1998	12 μs	1500
10 Gigabit Ethernet	10 Gbps	2002	1.2 μs	1500
100 Gigabit Ethernet	100 Gbps	2010	0.12μs	1500

(Adapted from [29])

Jumbo frames offer a substantial improvement in throughput over standard Ethernet frames, as they carry six times more data in a single frame. Hence the same amount data can be carried more effectively as a large IP packet might not needed to be fragmented or if it does need to be fragmented it results in fewer IP packets. Assuming TCP and IP headers of 20 bytes each and excluding any options, we can see from the computation below that data to header utilization is 2.55% higher. Hence jumbo frames can more *effectively* utilize the available Bandwidth (BW).

$$\text{Data to header ratio for 1500 MTU} = \frac{1500-40}{1500} = 97.00\%$$

$$\text{Data to header ratio for 9000 MTU} = \frac{9000-40}{9000} = 99.55\%$$

Now, consider an application has to sent a data of 18000 bytes via TCP, then the Maximum Segment Size (MSS) is equal to 8960 bytes, based upon:

$$MSS = MTU - 20(IP_{\text{header}}) - 20(TCP_{\text{header}}) \tag{2.1}$$

As Table 2.2 shows, by using jumbo frames the same amount of data can be carried in fewer packets. Sending fewer packets decreases the CPU overhead as the network stack has fewer packets to handle and only has to perform Transimssion Control Protocol (TCP) related operations three times rather than 13 times (as a minimum for both - assuming no packets are lost). A natural question is to ask, why the MTU is limited to 9000 bytes. This limit is because for frames larger than 12000 bytes, the bit error rate increases and it is difficult to detect the errors in physical link layer (due to the choice of checksum algorithm that is used). Hence, from the above discussion it is clear that using large packets reduces CPU utilization for protocol processing and increases the effective throughput.

Table 2.2: *Overhead comparison of standard and jumbo frames*

MTU size	MSS = MTU - 40	Total Packets Generated	Overhead bytes
1500	1460	13	$13 \times 40 = 559$
9000	8960	3	$3 \times 40 = 120$

2.6 Transmission Control Protocol

The Transimssion Control Protocol (TCP) is a reliable, connection-oriented, byte stream protocol. Figure 2.5 shows the TCP header. In order for two entities to communicate using the TCP protocol, they must first establish a connection using a three-way handshake. Following this each entity can stream data bytes to the other entity, who when they successfully receive these bytes will acknowledge the received bytes. If an ACK is not received within a certain period of time, then a transmission timer goes off and the sender retransmits the unacknowledged bytes. Retransmission is the foundation of TCP's reliable service.

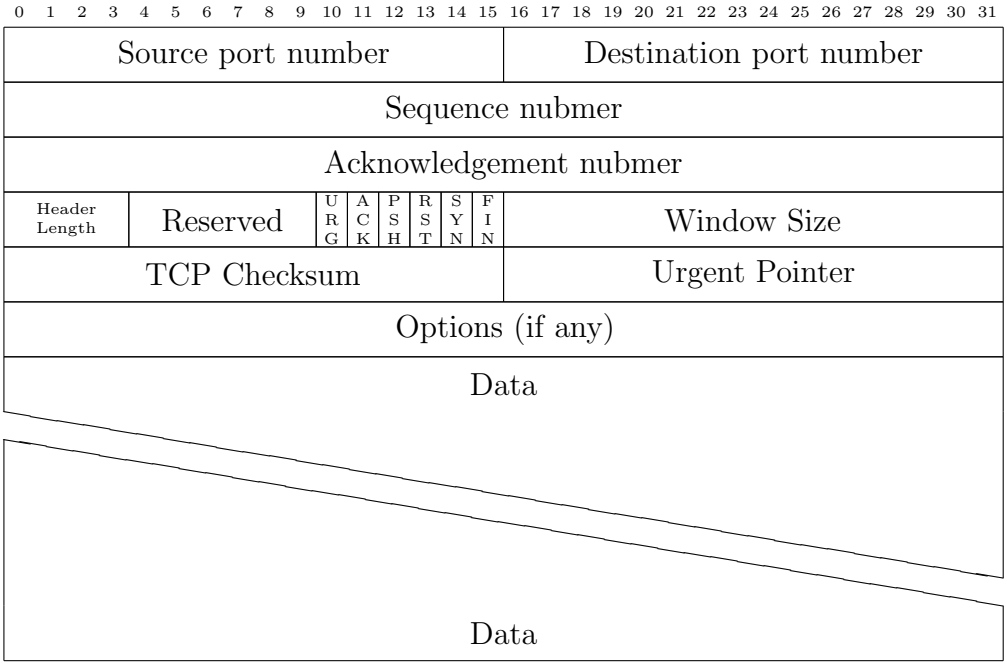

Figure 2.5: *TCP Header with data*
(Adapted from [30])

Other vital functions of TCP are **flow control** and **congestion control**.

Flow control is used by the *receiver* to control the transmission rate of the sender. Flow control prevents buffer overflow at the receiver. The receiver advertises a window size with every ACK, thus the sender can only send the number of bytes specified in this advertised window. Note that flow control **only** prevents the receiver's buffer from overflowing, it does not consider the buffering at any of the intermediate routers. To prevent the sender from exceeding the buffering capacity of the intermediate routers (and the network links) another window called the *congestion window* is used by the *sender* to avoid causing congestion in the network. In TCP the congestion window is governed by three algorithms: *slow start, congestion avoidance,* and *multiplicative decrease* [31].

During the initial connection establishment, the receiver advertises its *window size* (i.e., amount of data the sender can send) without awaiting for an acknowledgement. Once the connection is established, the congestion window size is additively increased until a loss is detected or a timer goes off. If either of these events occur the sending rate is decreased by a multiplicative factor. Dukkipati, et al. [32, 33], have shown the affects of window size and congestion window on throughput and have recently proposed in Request for Comments (RFC) 6928 to increase the initial window size to ten segments[†] [34]. Additionally, new protocols such as the "Fast and secure protocol" (fasp) [35], are being developed to overcome TCP's weaknesses.

In Linux, the TCP socket buffer space can altered in the kernel via the virtual file `proc/sys/net/ipv4/tcp_rmem`. Figure 2.6 shows that `tcp_rmem` has minimum, default, and maximum values. In this kernel, when an application creates a TCP socket it receives by default a buffer of 87380 bytes. However, the amount of buffer space can be altered by an application using the socket's Application Programming Interface (API) as long as the requested buffer space lies between the minimum and maximum values (6MB in this case). For an ideal receive window size, the buffer size should be greater than or equal to the Bandwidth Delay Product (BDP) in order to be able to fully utilize a physical link. BDP is given by equation 2.2. Linux also auto-tunes the default buffer size (within the minimum and maximum limit) for the given connection. Auto tuning is enabled in this kernel as `tcp_moderate_rcvbuf` is set to 1 and the TCP congestion algorithm used is cubic. For proper tuning of the system refer to [36].

[†]A segment is simply MSS bytes.

cat /proc/sys/net/ipv4/

```
tcp_rmem
4096        87380        6291456

tcp_moderate_rcvbuf
1

tcp_congestion_control
cubic
```

Figure 2.6: *TVP settings in a running Linux*

$$BDP = BW \times Dealy \tag{2.2}$$

2.7 Related Work

Virtualization has improved the overall utilization of computing resources, especially because it can exploit the processing power of multicore CPUs. However, virtualization also poses new challenges with regard to the networking as network packets undergo additional processing before reaching the guest OS. Although the available network bandwidth is typically high within a datacenter, the application performance of an application running in a guest OS is degraded because of the extra layers of processing. For this reason it is very important to measure and characterize the overheads and optimize the parameters of the network stack based upon understanding how these parameters and overheads affect the network's performance. A number of studies have been done on I/O virtualization. The following subsections summarize this research in two ares: scheduling and MTU size.

2.7.1 Work on schedulers

Performance of applications can be affected by the scheduler used in the VMM as explained in section 2.4. Different scheduling mechanisms have different effects on performance [37, 38]. An I/O intensive application's performance highly depends upon which scheduler is used. In the case of Xen, performance depends upon how the Dom0 is scheduled and Dom0 is generally scheduled more frequently than the guest domains [37]. Xen schedulers perform well on CPU intensive applications, but for I/O sensitive applications they achieve

varied results[24]. Mei, et al. [39] showed a performance gain of upto 40% simply by co-locating two communicating I/O sensitive applications. However, not all applications can be co-located as this would greatly limit scalability and the types of applications that can be run.

Apparao, Makinei, and Newell [40] showed that **TCP**'s performance decreased by 50% in a Xen virtualized system compared to a native Linux environment. This was due to the increase in the path length (the extra path length was due to the extra layers of processing). Benevenuto, et al. in order to assess the virtual overhead on applications, present a performance evaluation of a number of applications when these migrated from native execution to a Xen virtual environment [41]. Whiteaker, Schneider, and Teixeira [42] showed that network usage from competing VMs can introduce delays as high as 100 ms and that virtualization adds more to delay to the sending of packets than to receiving packets.

2.7.2 Work on MTU and network performance

There are very few studies done on how jumbo frames affect performance in a virtual environment, but there are extensive studies done on performance that can be achieved by exploiting other NIC capabilities in both traditional and virtual environments. Oi and Nakajima [43] showed that when using Large Receive Offload (LRO) in a vNIC, throughput increased by 14% and that large MTUs considerably improved throughput. Menon, Cox, and Zwaenepoel [44] proposed a new virtual interface architecture, by adding a software offload driver to the driver domain of the VMM. Y. Dong and colleagues[45, 46] showed the advantages of interrupt coalescence[‡]. The advantages of jumbo frames have been extensively studied and debated [6, 7, 47, 48, 29], but all of these studies were confined to a traditional physical environment.

[‡]In interrupt coalescence the CPU is interrupted once for a collection of multiple packets instead of generating interrupts for every single packet.

Chapter 3

Methodology

This chapter describes the methodology adapted for testing the affects of jumbo frames in a virtualized environment. The first section discusses the general considerations taken into account this evaluation. The first section explains the criterion considered when choosing a suitable workload. The second section explains what tools are required to measure these workloads. The final section describes the experimental setup.

As stated in section 1.1, the goal of this project is to study the affects of jumbo frames in a virtualized environment. There are many choices available for building a virtual environment and choosing the appropriate virtualization platform depends on many different factors. Hwang Jinho, et al. [49] compared four different hypervisors and concluded that no single virtualization platform was suitable for all types of applications. Different platforms are best suited for different applications and a heterogeneous environment is a desirable strategy for cloud data centers.

In order to build a virtualized environment for use in this thesis project, Xen was selected as described in section 2.4. Xen makes efficient use of multicore processors by scheduling the vCPUs appropriately. Since extensive studies have been done using Xen and a lot of existing research results are available, Xen was a good choice to build our virtualized environment. As more and more hardware assisted virtualization computers are being manufactured, it is appropriate to measure performance in a fully virtualized environment rather than a paravirtulized environment. However, in order to build a fully virtualized Xen environment, the underlying processor has to support virtualization.

3.1 Workloads and Tools

There are innumerable tools available to generate traffic for different needs when testing network performance. Many tools have been tested and considered during the course of this thesis project, nonetheless for the testing of network performance only a few tools suffice (in the context of this thesis). These tools provide similar statistical output. The following subsections describe some of the tools that were investigated and used for testing network performance in the experimental setup described in section 3.3.

3.1.1 Iperf

Iperf [50] is a powerful and simple tool used for measuring throughput and other network parameters. Iperf works on a client-server model and measures the throughput between two end systems. It can generate both TCP and User Datagram Protocol (UDP) traffic. Iperf allows us to test the network by setting various protocol parameters, such as MSS size, TCP window size, buffer length, multiple parallel connections, etc. After the program runs it provides a report on the throughput, jitter (packet delay variation), and packet loss. The main purpose of using Iperf is to fine tune a system by varying different parameters (for the given network conditions). The default port number of Iperf is 5001, which should be allowed through the firewall in order for client and server to connect. Iptables (i.e., the firewall in Linux) can be turned off entirely (this should only be done in an isolated testing environment). Figures 3.1 and 3.2 show an example of the output regarding MSS and bandwidth seen at server and client respectively.

`iperf -s -m`

```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte default
------------------------------------------------------------
[  4] local 192.168.1.102 port 5001 connected with 192.168.1.135 port 43931
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 5.0 sec    560 MBytes    935 Mbits/sec
[  4] MSS size 1448 bytes MTU 1500 bytes, ethernet
```

Figure 3.1: *Iperf server*

```
iperf -c 192.168.1.102 -i 1 -t 5
```

```
------------------------------------------------------------
Client connecting to 192.168.1.102, TCP port 5001
TCP window size: 87 KByte default
------------------------------------------------------------
[  3] local 192.168.1.135 port 43931 connected with 192.168.1.102 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec    114 MBytes    960 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  1.0- 2.0 sec    111 MBytes    934 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  2.0- 3.0 sec    111 MBytes    934 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  3.0- 4.0 sec    111 MBytes    934 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  4.0- 5.0 sec    112 MBytes    936 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 5.0 sec    560 MBytes    939 Mbits/sec
[  3] MSS size 1448 bytes MTU 1500 bytes, ethernet
```

Figure 3.2: *Iperf client*

3.1.2 TCPdump

TCPdump [51] was used to capture the network traffic. Ingress or egress traffic can be captured on a selected interface or of a network. TCPdump outputs the contents of the packets that match a boolean expression, to the user's desired level of detail. This program reports details such as what transport or application protocol is being used, hostnames, IP addresses, sequence numbers, and so on. We can also configure the program to capture a desired number of packets. For example 100 packets can captured using the command: `tcpdump -i eth0 -c 100`. Figure 3.3 shows a capture of the first five packets, as seen in a three-way TCP handshake indicated by `Flag [s]` - for the Syn flag and sender starts sending data beginning with the fourth packet.

tcpdump -i eth0 tcp -c 5

```
04:14:55.877324 IP 192.168.1.135.43928 > 192.168.1.102.commplex
-link: Flags [S], seq 753210484, win 14600, options [mss 1460,
sackOK,TS val 3049047 ecr 0,nop,wscale 7], length 0

04:14:55.877400 IP 192.168.1.102.commplex-link >
192.168.1.135.43928: Flags [S.], seq 3188553051, ack 753210485,
 win 14480, options [mss 1460,sackOK,TS val 141285276 ecr
3049047,nop,wscale 7], length 0

04:14:55.877516 IP 192.168.1.135.43928 > 192.168.1.102.commplex
-link: Flags [.], ack 1, win 115, options [nop,nop,TS val
3049047 ecr 141285276], length 0

04:14:55.877547 IP 192.168.1.135.43928 > 192.168.1.102.commplex
-link: Flags [P.], seq 1:25, ack 1, win 115, options [nop,nop,
TS val 3049047 ecr 141285276], length 24


04:14:55.877564 IP 192.168.1.102.commplex-link >
192.168.1.135.43928: Flags [.], ack 25, win 114, options [nop,
nop,TS val 141285276 ecr 3049047], length 0
```

Figure 3.3: *Example of tcpdump output*

3.1.3 httperf

Httperf was developed by David Mosberger and others at Hewlett-Packard (HP) Research Laboratories [52, 53]. It is a tool to measure a webserver's performance. Following is an example output for a webserver running in the VM, with httperf sending 2500 requests per second for a total of 10000 requests.

```
httperf --client=0/1 --server=192.168.1.104 --port=80 --uri=/ --rate=2500 --
    send-buffer=4096 --recv-buffer=16384 --num-conns=10000 --num-calls=1
Maximum connect burst length: 4

Total: connections 10000 requests 10000 replies 10000 test-duration 6.560 s

Connection rate: 1524.5 conn/s (0.7 ms/conn, <=639 concurrent connections)
Connection time [ms]: min 0.6 avg 115.2 max 4014.2 median 52.5 stddev 332.1
Connection time [ms]: connect 31.0
Connection length [replies/conn]: 1.000

Request rate: 1524.5 req/s (0.7 ms/req)
Request size [B]: 66.0

Reply rate [replies/s]: min 1991.0 avg 1991.0 max 1991.0 stddev 0.0 (1 samples
    )
Reply time [ms]: response 84.2 transfer 0.0
Reply size [B]: header 198.0 content 5039.0 footer 0.0 (total 5237.0)
Reply status: 1xx=0 2xx=0 3xx=0 4xx=10000 5xx=0

CPU time [s]: user 0.35 system 6.21 (user 5.3% system 94.6% total 99.9%)
Net I/O: 7894.8 KB/s (64.7*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
```

3.1.4 Additional tools

These additional tools were tested during the project. Netperf [54] was also developed at HP and is similar to iperf. Tcptrace [55] outputs statistics from a capture file (a sample output is given in appendix), ostinato [56] to generate traffic, and wireshark [57] (similar to tcpdump) can capture live traffic. Some other useful tools are tcpspray [58] and tcpreplay [59].

3.2 Measurement Metrics

The performance metrics we measured were network throughput and CPU utilization. By measuring these network related metrics, we can understand whether BW utilization has increased and whether the user (client) is getting increased performance by using jumbo frames. In particular, measurements were done only on TCP, as explained in the chapter 2. TCP was selected rather than User Datagram Protocol (UDP), because TCP is used by a lot of applications and TCP was thought to be more susceptible to the behavior of the scheduling mechanism of the hypervisor, thereby scheduling would have a larger effect on the throughput. These two network metrics will be analyzed and the results of the experiments will be present Chapter 4.

3.2.1 Network Throughput

Network throughput can be defined as the number of user data bytes transferred per unit amount of time. As Hassan and Jain, state *"An inefficient TCP algorithm or implementation can significantly reduce the effective throughput even if the underlying network provides a very high speed communication channel"* [31]. Equation 3.1 was proposed by Mathis, et al. in the paper "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm" [60]. Throughput is the primary metric used and it is expressed in Megabit per second (Mbps). Throughput in bits per second is calculated using the following equation:

$$Throughput \leq 0.7 \times \frac{MSS}{RTT\sqrt{PLoss}} \tag{3.1}$$

In this equation PLoss is the probability of packet loss, MSS is Maximum Segment Size in bits, and RTT is the Round Trip Time in seconds. We can see that throughput will always be less than the available network link bandwidth (as shown in Figure 3.4)

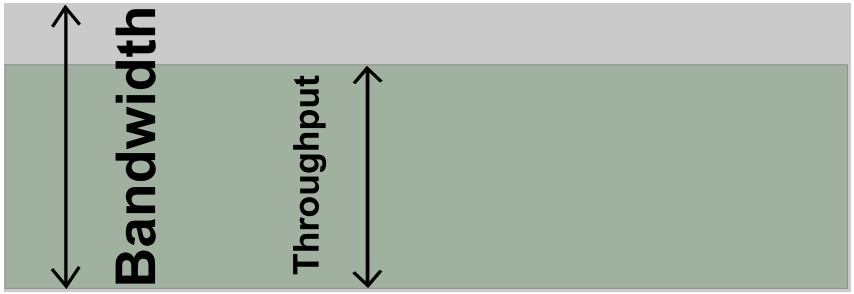

Figure 3.4: *Pictorial Representation of throughput and bandwidth for a physical link*

3.2.2 Network Latency

Latency or delay is the time taken for a IP packet from source to destination. The *Round Trip Time* is the time taken for an IP packet to travel from a sender to the receiver and the time it takes for the original sender to receive an ACK. RTT can be measured using the *ping* utility [61]. Ping is also a primary tool to check the network connectivity.

3.2.3 CPU utilization

Measuring CPU utilization of a process is necessary because this metric shows, how much of the CPU's processing power a particular process is consuming.

Utilization of the CPU is important because the CPU (or set of CPUs) is being shared by all the other applications. CPU utilization is usually measured as the percentage of the CPU that is being utilized. To collect this data we monitor the **netback** process, which is the backend in the Xen hypervisor as explained 2.4. This backend actually sends the traffic between virtual machines and the hypervisor. CPU utilization can be obtained in many different ways. In the following paragraphs we consider three alternative methods to obtain this data.

/proc/*pid*/stat

Every process running under a Linux or any Unix based OS has a Process Identification Number (pid). In Linux, statistics of a process can be found via the file `/proc/pid/stat`. These stats are calculated from the boot time, hence the values represent the cumulative resource usage of a particular process from boot time, rather than instantaneous usage.

pidstat

Pidstat [62] is a monitoring tool used for currently (or recently) executing processes in Linux. It provides CPU utilization and other resource statistics. A specific process can be monitored by entering its pid. Figure 3.5 shows the output for monitoring of the netback service once every second.

```
pidstat -p 790 1
```

```
10:05:29 PM       PID    %usr %system  %guest    %CPU   CPU  Command
10:05:30 PM       790    0.00   22.00    0.00   13.33     0  netback/0
10:05:31 PM       790    0.00   21.00    0.00   12.65     0  netback/0
10:05:32 PM       790    0.00   21.00    0.00   12.35     0  netback/0
10:05:33 PM       790    0.00   21.00    0.00   12.14     0  netback/0
10:05:34 PM       790    0.00   22.00    0.00   13.10     0  netback/0
10:05:35 PM       790    0.00   21.00    0.00   12.28     0  netback/0
10:05:36 PM       790    0.00   21.00    0.00   12.35     0  netback/0
10:05:37 PM       790    0.00   21.00    0.00   12.50     0  netback/0
10:05:38 PM       790    0.00   22.00    0.00   13.02     0  netback/0
```

Figure 3.5: *Example of pidstat output*

top

The top utility shows the currently running processes in Unix-like OSs. It periodically displays CPU usage, memory usage, and other statistics. The default ordering is from highest to lowest CPU usage of processes. Only the top CPU consuming processes can be seen, with the number limited by the display size.

3.3 Experimental Setup

Figure 3.6 shows a schematic view of the experimental setup. Dom0 is CentOS 6.5 running on Xen 4.3.2 hypervisor. Almost all major Linux distributions [63, 64, 65] were tested including Xenserver [66], which was recently released as an opensource server by Citrix [67]. Out of those OSs tested, CentOS [68] was chosen because it was easy to implement and has good support for Xen. VMs (DomUs) and clients are also running CentOS. Clients are connected directly to the server using cross-over Ethernet cables. A similar setup was used in the paper "Large MTUs and Internet Performance" [29] by Murray, et al.

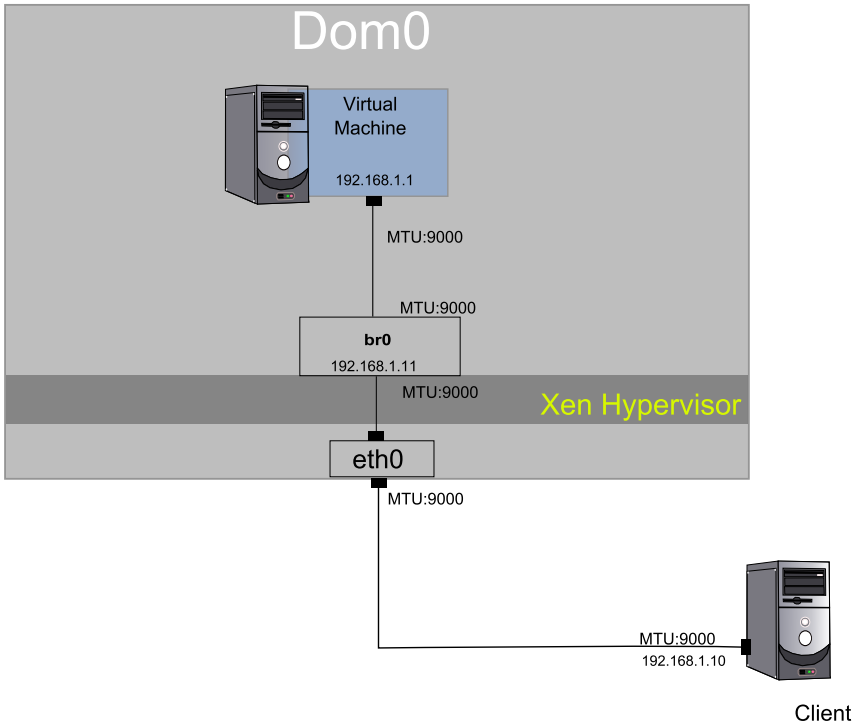

Figure 3.6: *Experimental Setup*

The testing* environment consisting of a server with an Intel E6550, 2.33GHz dual core processor, which has virtualization support, 3GB RAM, one on-board Intel 1GbE network interface and one 1GbE Peripheral Component Interconnect (PCI)-NIC. See appendix A for a complete configuration. Figure 3.7 shows a screen shot of the virtual machine manager (this application is used to manage VMs) with Dom0 and three VMs running on Xen hypervisor.

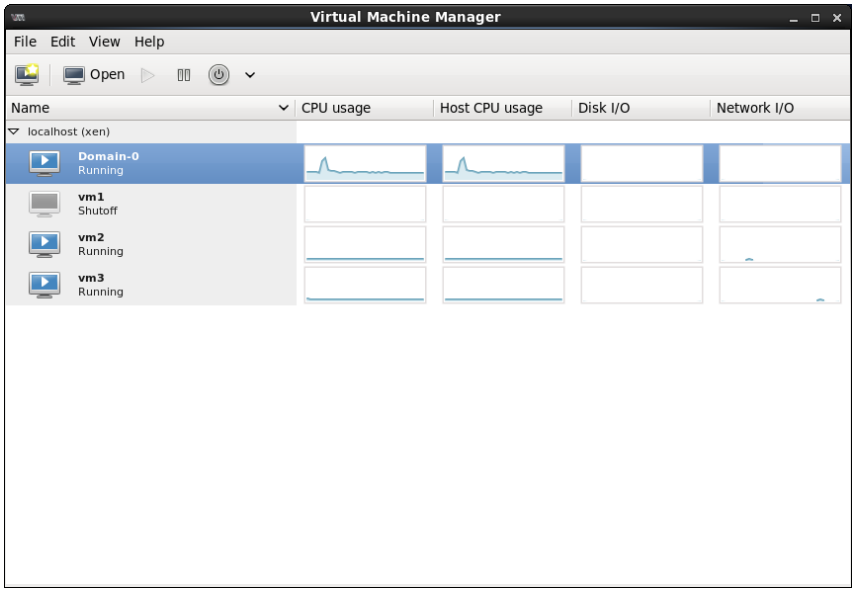

Figure 3.7: *Screen-shot showing Dom0 and two running VMs*

3.3.1 Bridging

Bridging inside Linux (Dom0) is an emulated physical bridge [69], which works just like a physical bridge (see Figure 3.8). This bridge forwards Ethernet frames to designated bridge ports based on Media Access Control (MAC) addresses. A virtual interface of a VM generated by the hypervisor is assigned a MAC address by the hypervisor and is attached to the bridge. Additionally a `bond` can be configured to use multiple NICs (see Figure 3.9) and different bonding modes (as shown in Table 3.1) can be used to suit network requirements, for example to load balance.

*The resources and space used for these experiments were at the Communication Systems department at KTH Royal Institute of Technology.

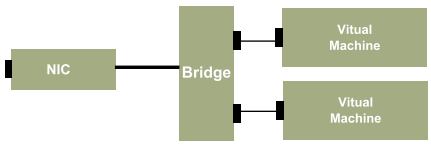

Figure 3.8: *Linux Bridging*

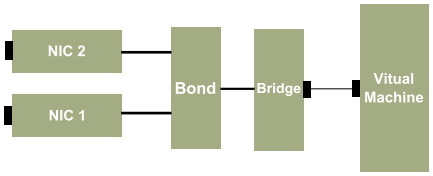

Figure 3.9: *Linux bonding with bridge*

Table 3.1: *Bonding Modes*

Mode 1	Active-backup
Mode 2	Balance-xor
Mode 3	Broadcast
Mode 4	Link aggregation
Mode 5	Adaptive transmit load balancing
Mode 6	Adaptive transmit and receive load balancing

Chapter 4

Evaluation and Results

This chapter begins with an explanation of the various test cases designed to evaluate the impact of the use of jumbo frames on network performance in keeping with the objective of the project. Extensive measurements were taken to analyze the experimental setup under various conditions. This chapter discusses the results based on the parameters chosen in section 3.2.

Lowering CPU overhead by using jumbo frames is desirable in a virtualized environment as it reduces the overhead in comparison with the use of standard Ethernet frames. Because TCP window size is sensitive to the time outs and packet drops, if the physical CPU cannot schedule the VM at sufficiently frequent time intervals due to increased load, then the congestion window in the client machine will perceive congestion in the network and TCP will enter into either slow start or the congestion avoidance phase. This will occur despite there **not** actually being any evidence of congestion in the network.

Given the experimental setup in Section 3.3 we changed the MTU, then see how this affects network performance. The interface's MTU can be set to 9000 bytes on an interface using the simple command: `ifconfig eth0 mtu 9000`. However, this MTU has to be enabled along the entire path between the end systems, otherwise fragmentation* will need to occur along the way. In this case, the same MTU size has been configured on all clients, the Dom0 physical NIC, virtual bridges, virtual interfaces, and in each guest OS. If the desired MTU is **not** correctly enabled on the bridge, then the bridge will silently discard the frames **without** any error notification. *Path MTU discovery* has been used to check the test link. Path MTU discovery detects the largest possible MTU that can be sent over the path without fragmentation. Additionally, there might be

*Fragmentation splits IP packets into smaller IP packets, so each can pass through the next hop link.

additional performance issues if different MTU sizes are set on a connecting link.

Different TCP buffer sizes and other parameters have been via measurements, for example iperf can be used with the option `-w` to specify different buffer sizes. The NIC used in the test was an `Intel 82566DM-2 Gigabit Network Connection (rev 02)` and this NIC has a number of capabilities, which offload the CPU load. Figure 4.1 shows the NIC features enabled throughout the tests, unless otherwise specified (for complete NIC features reference to Appendix A). For example, with this configuration TCP segmentation and checksum are done in the NIC rather than in network stack using the host's CPU. Hence enabling these NIC features boosts the network performance. Figure 4.2 shows the network stack with Iperf running in the user space and TCPdump capture at the network driver.

`NIC functions enabled`

```
tcp-segmentation-offload: on
        tx-tcp-segmentation: on
        tx-tcp-ecn-segmentation: off [fixed]
        tx-tcp6-segmentation: on
udp-fragmentation-offload: off [fixed]
generic-segmentation-offload: on
generic-receive-offload: on
large-receive-offload: off [fixed]
```

Figure 4.1: *NIC features enabled*

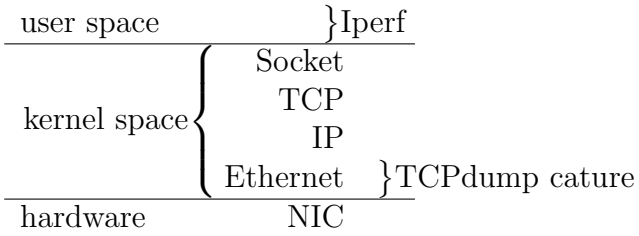

Figure 4.2: *Network protocol stack with Iperf and TCPdump*

4.1 Throughput

The first step is to measure the throughput between the client and Dom0, this measurement gives the throughput between the physical machines, *without* involving any other VMs. In this measurement we measured a performance improvement of 4.4 % in network throughput when using a jumbo frame of size 5000 bytes rather than the standard Ethernet frame size. The improvement in the virtual machine's network throughput is approximately 4.7 %, as shown in Figure 4.3.

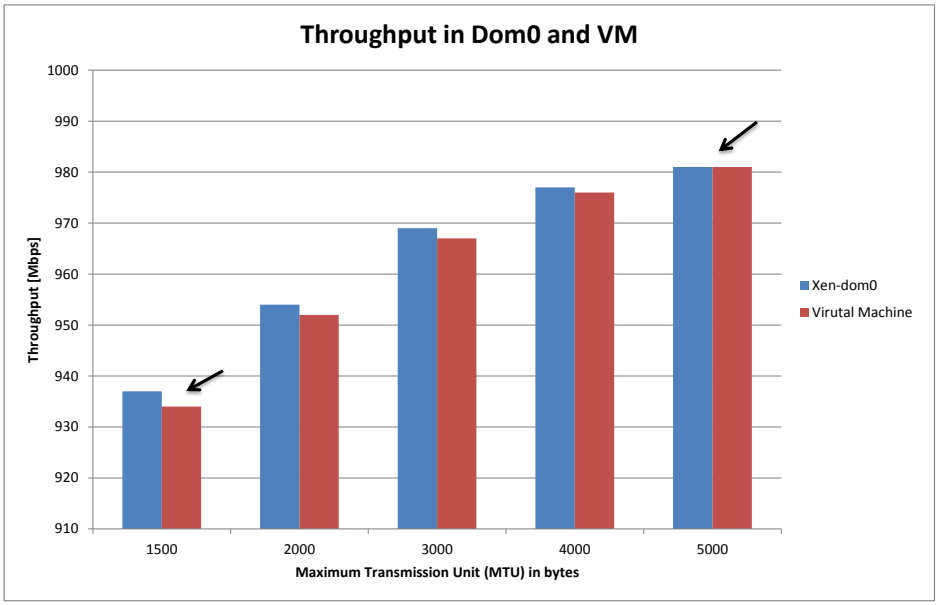

Figure 4.3: *Virtual Machine and Dom0 Throughput*

Table 4.1 and 4.2 shows the percent gain in throughput in virtual machine and Dom0 respectively. Throughput increase is initially greater (for 2000 bytes MTU), but as MTU increases, the increase in the throughput decreases. Beyond an MTU of 5000 bytes, there was no substantial gain in network throughput. This increase is approximately equal to 50 Mbps from 934 Mbps for standard Ethernet frame to 981 Mbps for 5000 bytes MTU, as shown in Table 4.3. Starting at 6000 bytes of MTU throughput suddenly drops as seen in Figure 4.4 (but only in the virtualized environment). Possible reasons for this behavior are discussed the Section 4.5.

Table 4.1: *Performance gain of virtual machine*

MTU	BW Utilization	Gain
1500	93.4 %	4.7 %
5000	98.1 %	

Table 4.2: *Performance gain in Dom0*

MTU	BW Utilization	Gain
1500	93.7 %	4.4 %
5000	98.1 %	

Table 4.3: *Average Throughput over 10 seconds*

MTU	Dom0 [Mbps]	VM [Mbps]
1500	937	934
2000	954	952
3000	969	967
4000	977	976
5000	981	981

For complete measurement data refer to Appendix B.

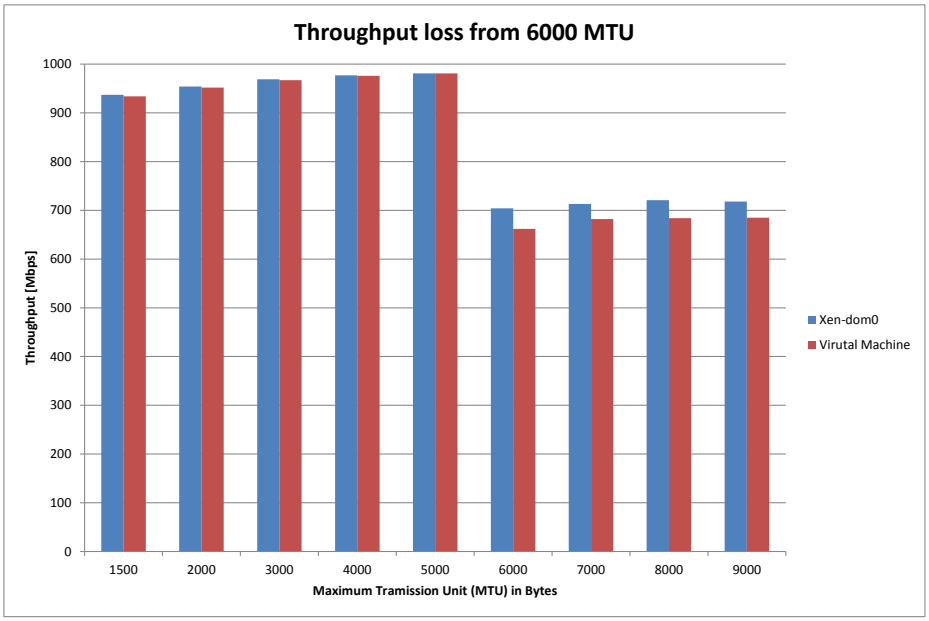

Figure 4.4: *Throughput observed to decrease from 6000 bytes MTU*

4.2 CPU Utilization

Figure 4.5 shows CPU utilization of the *netback* service of Xen, as it forwards packets to the VM. As can be seen in this figure the CPU utilization of this service decreases, as the MTU increases. The netback service consumed 24.70% of CPU when 1500 byte packets were being forwarded to the VM, while when 4000 bytes MTU packets were being forwarded netback was consuming only about 17.10% of CPU. However, once the MTU reached 5000 bytes the CPU utilization increased to 19.20%, however - the CPU utilization was still lower that when the MTU was 1500 bytes. Finally the CPU utilization was 20.88% for 9000 bytes packets. Overall, the CPU consumption of the service decreased for larger sized frames than for standard sized Ethernet frames. One point to remember is, these values are percentage values rather than absolute values (i.e., number of CPU cycles).

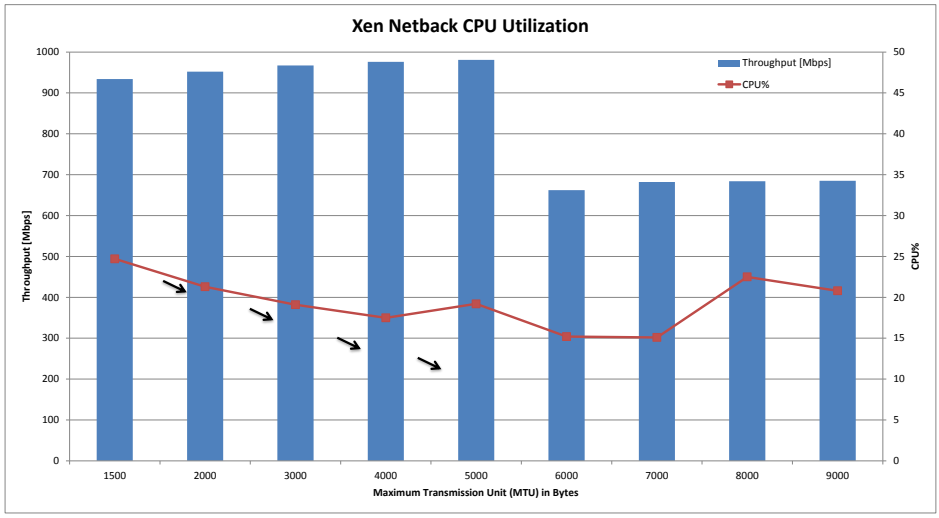

Figure 4.5: *CPU usage of Netback service in Xen*

4.3 Throughput at the client

Figure 4.6 shows the throughput as seen at the clients for 1500 and 5000 bytes of MTU over an interval of 60 seconds.

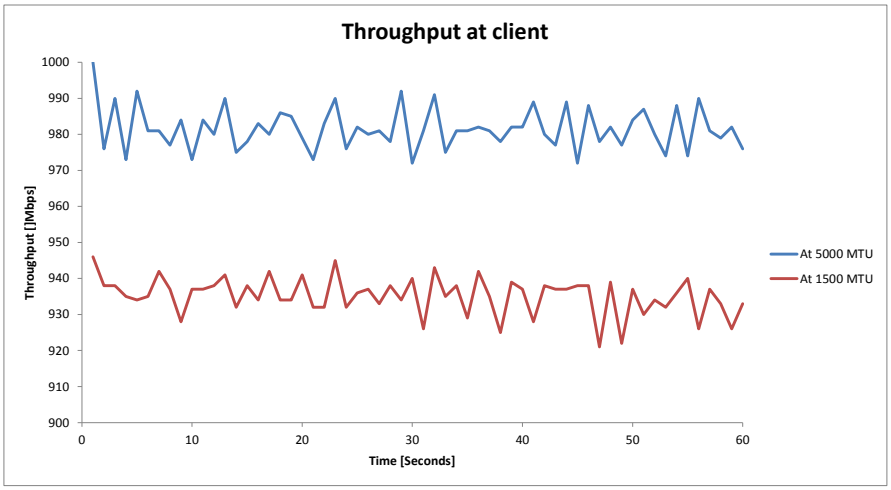

Figure 4.6: *Throughput seen at client*

The confidence interval is given by equation 4.1, where 1.96 is the constant for a Normal distribution for a confidence of 95%. In this equation n is the sample size, σ is the Standard Deviation, and $\bar{x}$ is mean of the sample. The confidence interval tells us that 95% of the time or with 95% certainty the result will be in between the upper and lower bounds, with the specified margin of error.

$$ConfidenceInterval = \bar{x} \pm \frac{1.96\sigma}{\sqrt{n}} \tag{4.1}$$

Confidence Interval for 1500 MTU

```
Confidence Level95,0%          1.359049674
Lower bound                    933.8242837
Upper bound                    936.542383
```

Confidence Interval for 5000 MTU

```
Confidence Level95,0%          1.521455208
Lower bound                    980.0618781
Upper bound                    983.1047885
```

4.4 Additional Measurements

The following subsections presents additional measurements. The Xen installed machine is compared with the native Linux machine in the first section. Following this TCP's performance in the virtual machine is analyzed.

4.4.1 Xen Performance Comparison

Figure 4.7 compares native Linux to a Xen installed Linux. Xen performs remarkably well and there was no performance loss compared to the native Linux machine. We can see this as we vary the MTU size from the standard frame of 1500 bytes to a 5900 bytes MTU, there was no loss in throughput. However, with an MTU of 6000 bytes, throughput suddenly falls for both Xen Dom0 and the VM - but **not** for the native Linux.

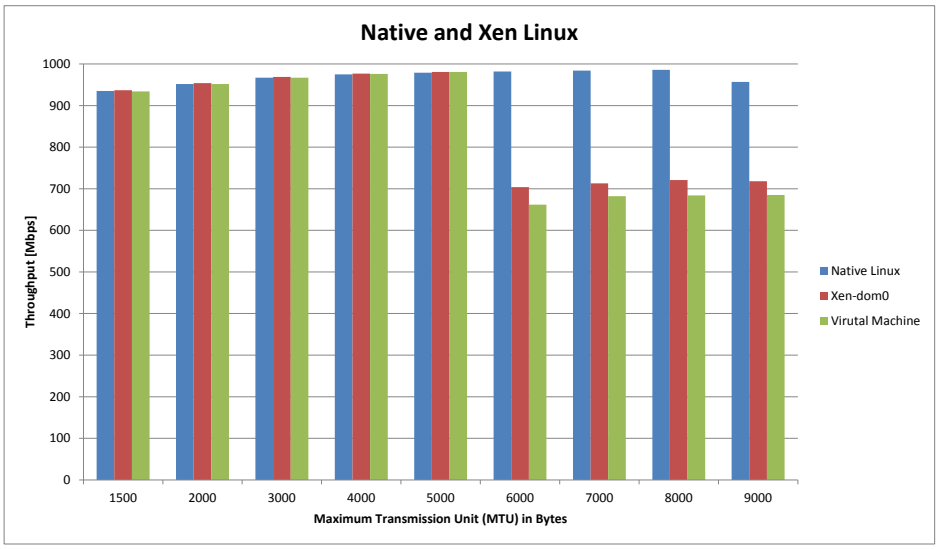

Figure 4.7: *Xen Performance compared to native Linux system*

4.4.2 TCP Behavior in Virtual Machine

In the following measurements the offloading capabilities of the NIC were turned **off**. This offloading was turned off in order to have accurate measurements of TCP's performance in the VM. If reassembling is done in the NIC, then packets are assembled before reaching the hypervisor and VM and TCPdump captures these large packet sizes which are reassembled by the NIC. Hence NIC offloading features were turned off to capture the correct packet sizes.

Figures 4.8 to 4.13, show packet captures with three different packet sizes (1500, 5000, and 9000 bytes), ACK packets have not been included. The client is sending these packets to VM, while these packets were captured as they enter Dom0 and VM. These measurements have μs granularity. As seen, the TCP's flow (sequence) in Dom0 is normal, but the flow in the VM is regular this is because of the VM scheduling. The inter-arrival time in the VM is regular, rather than bursty and this is good for applications running in VM. However, as the number of VMs increases the gap between the sequences of packets also increases.

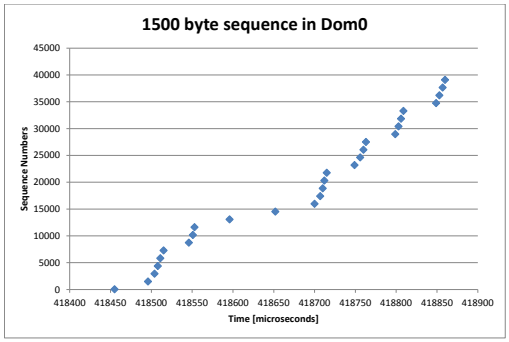

Figure 4.8: *Sequence of 1500 byte MTU packets in Dom0*

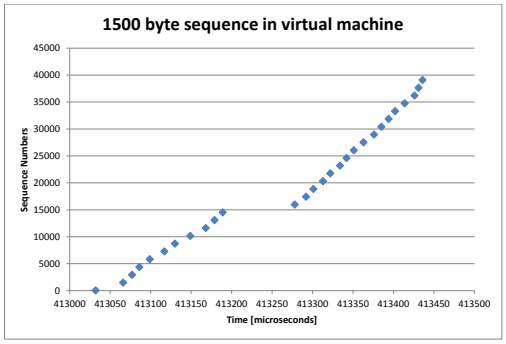

Figure 4.9: *Sequence of 1500 byte MTU packets in VM*

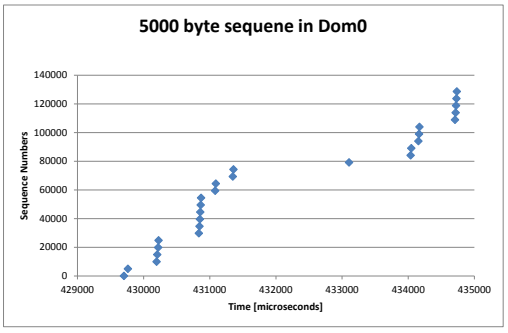

Figure 4.10: *Sequence of 5000 byte MTU packets in Dom0*

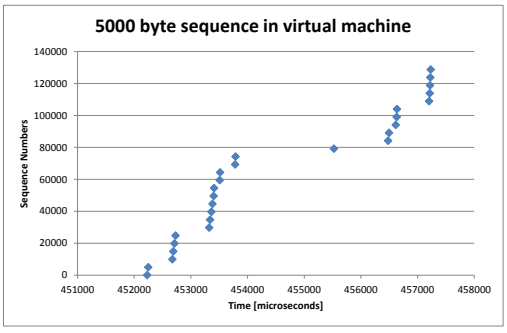

Figure 4.11: *Sequence of 5000 byte MTU packets in VM*

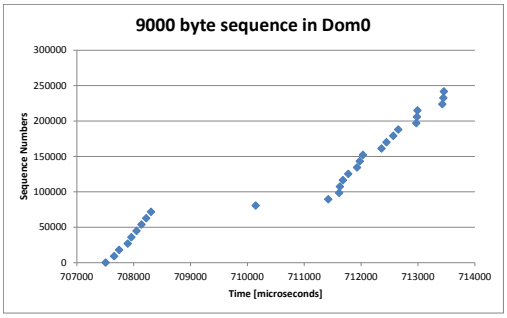

Figure 4.12: *Sequence of 9000 byte MTU packets in Dom0*

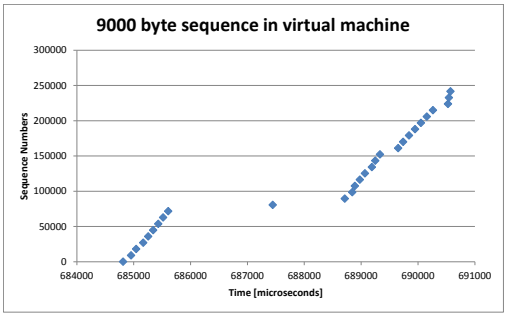

Figure 4.13: *Sequence of 9000 byte MTU packets in VM*

4.5 Analysis and Discussion

Although the test environment might represent a real virtual production environment, these measurements do not correspond to real network measurements as these measurements were made in a loss-less laboratory environment. In a real production environment other factors may come into play. Hence these measurements might not predict what would happen in a wide area network setting or in a real production environment. However, *within* a data center we expect a very low packet loss rate, i.e., comparable to what we had in our lab environment.

Equation 3.1 can be interpreted as, throughput is directly proportional to MSS, which in-turn is limited by MTU, as given by equation 2.1. Hence the greater the MTU size the higher the network performance. Transmitting larger packets by increasing MTU size reduces the CPU overhead. The overhead reduction on the CPU is due to a decrease in the aggregate of overhead, while the *per-packet* overhead is constant. This decrease in overhead is highly beneficial for bulk transfers (which is essentially what Iperf is measuring). While per-packet overhead remains the same the use of larger packets reduces the load on the end-systems and the load on the intermediate routers.

There are certain points to consider when implementing jumbo frames. The first point to consider is the buffer space required in the end systems and in each of the intermediate routers. If a large amount of buffer space is required in the routers, then this buffer space could be filled quickly and buffer overflow

might be occur if packets are not forwarded swiftly. Additionally, if an error occurs then re-transmission will be more costly. In the end system the TCP window size shrinks quickly, if the application does not process the data bytes quickly enough.

As seen in Section 4.4, there is no performance variation between standard sized packets and jumbo frames in a virtual environment. Xen performed well in scheduling the VM and TCP's performance (at μs granularity) is good when viewed from the viewpoint of the VM.

Recall from Section 2.2 that virtualization itself is nothing more than abstraction beneath the OS, as the hypervisor simply schedules the execution of the VMs. The hypervisor's scheduling algorithm does not consider the functioning of the OS running above it. Hence the entire protocol stack, from application to physical layer, should be taken into account in order to achieve greater performance. The *abstraction* concept of protocols has brought independence and facilitated the design and implementation of protocols. However, the performance of the entire system depends upon the complete design, implementation, and operation of all parts of the protocols and additionally is affected by their interaction (as shown in Figure 4.14).

Even if the physical layer is a GbE and the data link layer's MTU has the capacity to transfer data at a higher rate if the transport layer (in this case due to TCP's slow start and congestion avoidance behavior is not congruent with the hardware's capabilities, then the performance will suffer. Therefore, the focus to achieve network performance improvements should be on core protocols behavior inside the OS. One of the principle motivations of this thesis project is that the hardware capabilities are not completely exploited by software protocols. The limit of 1500 bytes MTU was designed for old hardware, thus today's GbE will be underutilized when using this size MTU.

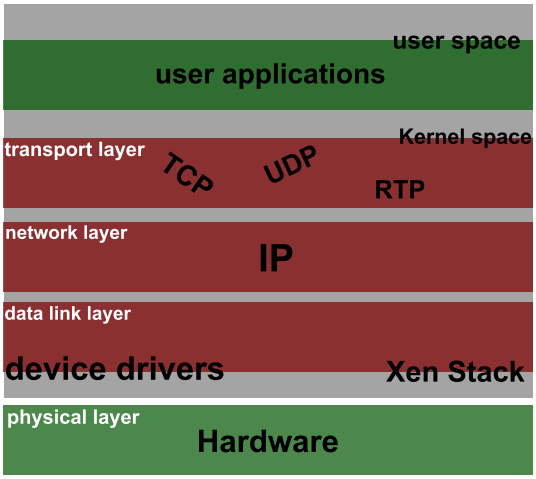

Figure 4.14: *Incongruency of abstraction layering concept*

Chapter 5

Conclusions and Future Work

This chapter concludes the thesis report with some future work and reflections on the project.

5.1 Conclusions

The results of the tests performed and evaluation of these results has clearly demonstrated that network performance can be improved in both the virtual and physical environments by using larger MTUs. Approximately 5% performance gain was achieved in the virtual environment by increasing the MTU from 1500 bytes to 5000 bytes. Additionally the CPU overhead can be reduced by transmitting larger frames. Hence the use of jumbo frames is clearly advantageous for virtual environments. Enabling jumbo frames will have a tremendous advantage for cloud computing, for example live migration and other data storage and transfer protocols (such as Network File System (NFS), Network-attached storage (NAS), Internet Small Computer System Interface (iSCSI), etc.).

One key point cloud customers have to remember is that although cloud providers provide hardware resources dynamically, the performance seen by the end user depends upon the choice of scheduling mechanism, scheduling and kernel parameters, and upon data prioritization.

In a virtual environment, the complete utilization of the network's resources and the observed performance of an application depends entirely on the how the network protocols in the OS are implemented and how applications are prioritized. The flow of information from end-to-end is very dependent upon

the system's data path (kernel) rather than on the long fast pipes that connect the endpoints.

5.2 Future Work

Throughput suddenly decreased starting at an MTU size of 6000 bytes MTU, while upto that point throughput increased as MTU size increased, especially in the virtual environment.This is might to attributed to limits in the Xen stack, page size, or some other limit may be causing this behavior. Due to the time constraints for this thesis project, further exploration of the reasons why this behavior occurs has been left for future work. This future work should consider making changes to the Xen code in order to explore the causes for the observed behavior as well as looking for solutions to maintain the increased performance with larger MTU sizes for Dom0 and the VM.

It would be interesting to see if there will be additional performance increases using an MTU of 9000 or greater. Additionally, throughput can be measured between two VM's, which is not limited by MTU and performance might be faster since it is just memory a mapping. This project specifically focused on TCP's performance while excluding UDP, therefore an investigation of UDP's performance is clearly another area of future work.

Jitter in the network due to VM scheduling of Xen is an interesting area to explore. The affect on jumbo frames in comparison with standard Ethernet frame sizes can be studied further. How do two or more VMs using different MSS sizes co-exist in a single physical machine, since the total amount of buffer space available to the network stack is limited. As Eniko Fey pointed in the her thesis "The Effect of Combining Network and Server QoS Parameters on End-to-End Performance" VLANs can be used to prioritize traffic [70].

5.3 Required Reflections

Cloud computing itself is a cost effective solution for the IT industry and other IT enabled industries. Yet cloud providers have to minimize their costs and maximize their profits. Jumbo frames further reduces the costs of cloud providers by efficiently utilizing their in-house network and reducing power consumption by reducing the number of CPU cycles required to transfer a given amount of data. This project effectively addressed both economic and environmental aspects of cloud providers and clients.

Cloud computing has leveraged the IT infrastructure by utilizing the unused computing capacity. Usually, only an average of 30% or even lower processing power is utilized in a data center [71], which is incredibly low utilization. But using virtualization these underutilized servers can used more efficiently. With proper planning, monitoring and personal, data centers and companies can virtualize their own available infrastructure without moving to any third party cloud providers, which results in huge power savings.

Recent research [72] concluded that there will a substantial energy savings by moving the office work to cloud. However this might not be the precise interpretation, as according to Scott M. Fulton III [73] the *virtualization* used by the cloud providers is the primary reason for cost savings and reduced power consumption, rather than the use of the cloud model itself. In order for clouds to be greener, more work has to be done on virtualization specifically on network virtualization which is responsible for interconnecting rapidly changing clouds.

Bibliography

[1] J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "Contention aware execution: online contention detection and response," in *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, CGO '10, (New York, NY, USA), pp. 257–265, ACM, 2010.

[2] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing Performance Isolation Across Virtual Machines in Xen," in *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, Middleware '06, (New York, NY, USA), pp. 342–362, Springer-Verlag New York, Inc., 2006.

[3] I. Paul, S. Yalamanchili, and L. John, "Performance impact of virtual machine placement in a datacenter," in *Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International*, pp. 424–431, Dec 2012.

[4] M. Hassan and R. Jain, *High Performance TCP/IP Networking*, ch. 13, TCP Implemenation, p. 308. Pearson Prentice Hall, 2004.

[5] S. Tripathi, N. Droux, T. Srinivasan, and K. Belgaied, "Crossbow: From Hardware Virtualized NICs to Virtualized Networks," in *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '09, (New York, NY, USA), pp. 53–62, ACM, 2009.

[6] M. Mathis, "Raising the Internet MTU." http://staff.psc.edu/mathis/MTU/index.html. [Online; Last accessed 19-March-2014; Last modified: not available].

[7] P. Dykstra, "Gigabit Ethernet Jumbo Frames." http://sd.wareonearth.com/~phil/jumbo.html, December, 1999. [Online; Last accessed 2-February-2014].

[8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[9] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 5th ed., 2011.

[10] J. Hamilton, "Perspectives." http://perspectives.mvdirona.com/. [Online; Last accessed 12-March-2014; Last modifited 14-February-2014].

[11] "Google App Engine." https://developers.google.com/appengine/. [Online; accessed 19-March-2014; Last modified 7-February-2014].

[12] "Heroku." https://www.heroku.com/. [Online; Last accessed 19-March-2014; Last modified: not available].

[13] R. A. Meyer and L. H. Seawright, "A Virtual Machine Time-sharing System," *IBM Syst. J.*, vol. 9, pp. 199–218, Sept. 1970.

[14] R. P. Goldberg, "Architecture of Virtual Machines," in *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*, AFIPS '73, (New York, NY, USA), pp. 309–318, ACM, 1973.

[15] R. P. Goldberg, "Survey of virtual machine research," *Computer*, vol. 7, pp. 34–45, June 1974.

[16] P. H. Gum, "System/370 Extended Architecture: Facilities for Virtual Machines," *IBM Journal of Research and Development*, vol. 27, pp. 530–544, Nov. 1983.

[17] K. Adams and O. Agesen, "A Comparison of Software and Hardware Techniques for x86 Virtualization," *SIGARCH Comput. Archit. News*, vol. 34, pp. 2–13, Oct. 2006.

[18] R. Dittner and D. Rule, *The Best Damn Server Virtualization Book Period: Including Vmware, Xen, and Microsoft Virtual Server*. Burlington, MA: Elsevier, 2007.

[19] "Hardware-Assisted VT." http://www.intel.com/content/www/us/en/virtualization/virtualization-technology/hardware-assist-virtualization-technology.html. [Online; Last accessed 2-March-2014; Last modified: not available].

[20] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith, "Intel Virtualization Technology," *Computer*, vol. 38, pp. 48–56, May 2005.

[21] Y. Luo, "Network I/O Virtualization for Cloud Computing," *IT Professional*, vol. 12, pp. 36–41, Sept 2010.

[22] 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," *SIGOPS Oper. Syst. Rev.*, vol. 37, pp. 164–177, Oct. 2003.

[23] "Kernel-based Virtual Machine." http://www.linux-kvm.org/page/Main_Page. [Online; Last accessed 19-March-2014; Last modified: not available].

[24] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in Virtual Machine Monitors," in *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '08, (New York, NY, USA), pp. 1–10, ACM, 2008.

[25] "Xen Credit Scheduler." http://wiki.xen.org/wiki/Credit_Scheduler. [Online; Last accessed 19-March-2014; Last modified 15-December-2013].

[26] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the Three CPU Schedulers in Xen," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, pp. 42–51, Sept. 2007.

[27] M. Levy, "Jumbo Frame Deployment at Internet Exchange Points." http://tools.ietf.org/html/draft-mlevy-ixp-jumboframes-00, November 14, 2011. [Online; Last accessed 20-February-2014].

[28] IEEE Standards Association. http://standards.ieee.org/about/get/802/802.3.html. [Online; Last accessed 19-March-2014; Last modified: not available].

[29] D. Murray, T. Koziniec, K. Lee, and M. Dixon, "Large MTUs and internet performance," in *High Performance Switching and Routing (HPSR), 2012 IEEE 13th International Conference on*, pp. 82–87, June 2012.

[30] W. R. Stevens, *TCP/IP Illustrated (Vol. 1): The Protocols*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1993.

[31] M. Hassan and R. Jain, *High Performance TCP/IP Networking*. Pearson Prentice Hall, 2004.

[32] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin, "An Argument for Increasing TCP's Initial Congestion Window," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 26–33, June 2010.

[33] N. Dukkipati, M. Mathis, Y. Cheng, and M. Ghobadi, "Proportional Rate Reduction for TCP," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, (New York, NY, USA), pp. 155–170, ACM, 2011.

[34] J. Chu, N. Dukkipati, Y. Cheng, M. Mathis, "RFC6928 - Increasing TCP's Initial Window." https://tools.ietf.org/html/rfc6928, April, 2013. [Online; Last accessed 20-February-2014].

[35] Z. Conghua and C. Meiling, "Analysis of fast and secure protocol based on continuous-time Markov chain," *Communications, China*, vol. 10, pp. 137–149, Aug 2013.

[36] J. Mahdavi, "Enabling High Performance Data Transfers." http://www.psc.edu/index.php/networking/641-tcp-tune. [Online; Last Updated on Wednesday, 24 October 2012 15:50; Last accessed 20-February-2014].

[37] L. Cherkasova, D. Gupta, and A. Vahdat, "When Virtual is Harder than Real: Resource Allocation Challenges in Virtual Machine Based IT Environments." http://www.hpl.hp.com/techreports/2007/HPL-2007-25.pdf, February 2007.

[38] Zhuang, Hao, "Performance Evaluation of Virtualization in Cloud Data Center," Master's thesis, School of Science, Aalto University, 2012. Available at: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-104206.

[39] Y. Mei, L. Liu, X. Pu, S. Sivathanu, and X. Dong, "Performance Analysis of Network I/O Workloads in Virtualized Data Centers," *IEEE Trans. Serv. Comput.*, vol. 6, pp. 48–63, Jan. 2013.

[40] P. Apparao, S. Makineni, and D. Newell, "Characterization of network processing overheads in Xen," *First International Workshop on Virtualization Technology in Distributed Computing (VTDC 2006)*, pp. 2–2, Nov. 2006.

[41] F. Benevenuto, C. Fernandes, M. Santos, V. Almeida, and J. Almeida, "A Quantitative Analysis of the Xen Virtualization Overhead." http://homepages.dcc.ufmg.br/~fabricio/download/ccc07.pdf, November 2007.

[42] Whiteaker, Jon and Schneider, Fabian and Teixeira, Renata, "Explaining Packet Delays Under Virtualization," *SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. 38–44, Jan. 2011.

[43] H. Oi and F. Nakajima, "Performance Analysis of Large Receive Offload in a Xen Virtualized System," *2009 International Conference on Computer Engineering and Technology*, pp. 475–480, Jan. 2009.

[44] A. Menon, A. L. Cox, and W. Zwaenepoel, "Optimizing Network Virtualization in Xen," in *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, (Berkeley, CA, USA), pp. 2–2, USENIX Association, 2006.

[45] H. Guan, Y. Dong, R. Ma, D. Xu, Y. Zhang, and J. Li, "Performance Enhancement for Network I/O Virtualization with Efficient Interrupt Coalescing and Virtual Receive-Side Scaling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1118–1128, 2013.

[46] Y. Dong, D. Xu, Y. Zhang, and G. Liao, "Optimizing Network I/O Virtualization with Efficient Interrupt Coalescing and Virtual Receive Side Scaling," in *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, (Washington, DC, USA), pp. 26–34, IEEE Computer Society, 2011.

[47] Alteon Networks, white paper, "Extended Frame Sizes for Next Generation Ethernets." http://staff.psc.edu/mathis/MTU/AlteonExtendedFrames_W0601.pdf. [Online; Last accessed 20-February-2014].

[48] W. Rutherford, L. Jorgenson, M. Siegert, P. Van Epp, and L. Liu, "16000-64000 B pMTU Experiments with Simulation: The Case for Super Jumbo Frames at Supercomputing '05," *Opt. Switch. Netw.*, vol. 4, pp. 121–130, June 2007.

[49] J. Hwang, S. Zeng, F. Wu, and T. Wood, "A component-based performance comparison of four hypervisors," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pp. 269–276, May 2013.

[50] "Iperf." http://sourceforge.net/projects/iperf/. [Online; accessed 4-February-2014; Last modified 12-June-2013].

[51] "TCPdump." http://www.tcpdump.org/. [Online; Last accessed 19-March-2014; Last modified: not available].

[52] D. Mosberger and T. Jin, "httperf - A Tool for Measuring Web Server Performance," in *In First Workshop on Internet Server Performance*, pp. 59–67, ACM, 1998.

[53] "Httperf." http://sourceforge.net/projects/httperf/. [Online; Last accessed 19-March-2014; Last modified 08-April-2013].

[54] "Netperf." http://www.netperf.org/netperf/. [Online; Last accessed 19-March-2014; Last modified: not available].

[55] "Tcptrace." http://www.tcptrace.org/. [Online; Last accessed 18-March-2014; Last modified: not available].

[56] "ostinato." https://code.google.com/p/ostinato/. [Online; Last accessed 18-March-2014; Last modified: not available].

[57] "Wireshark." http://www.wireshark.org/. [Online; Last accessed 18-March-2014; Last modified: not available].

[58] "Tcpspray." http://linux.die.net/man/1/tcpspray. [Online; Last accessed 18-March-2014; Last modified 31-October-2010].

[59] "Tcpreplay." http://tcpreplay.synfin.net/. [Online; Last accessed 8-March-2014; Last modified 14-December-2013].

[60] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 27, pp. 67–82, July 1997.

[61] "Ping Utility." http://www.freebsd.org/cgi/man.cgi?query=ping&sektion=8. [Online; Last accessed 12-March-2014; Last modified: not available].

[62] "pidstat man page." http://linux.die.net/man/1/pidstat andhttp://www.mjmwired.net/kernel/Documentation/filesystems/proc.txt. [Online; Last accessed 20-February-2014].

[63] "Debian." https://wiki.debian.org/Xen. [Online; Last accessed 12-March-2014; Last modified 05-March-2014].

[64] "Fedora Xen Archive." https://fedoraproject.org/wiki/Archive:Tools/Xen?rd=FedoraXenQuickstart. [Online; Last accessed 12-March-2014; Last modified 04-July-2013].

[65] "Ubuntu Xen Wiki." https://help.ubuntu.com/community/Xen. [Online; Last accessed 12-March-2014; Last modified 22-December-2012].

[66] "XenServer Open Source Virtualization." http://www.xenserver.org/. [Online; Last accessed 12-March-2014; Last modified: not available].

[67] "Citrix." http://www.citrix.com/. [Online; Last accessed 12-March-2014; Last modified: not available].

[68] "Xen4 CentOS6 QuickStart." http://wiki.centos.org/HowTos/Xen/Xen4QuickStart. [Online; Last accessed 12-March-2014; Last modified 10-January-2014].

[69] "Linux and Xen Bridging." http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge and http://wiki.xen.org/wiki/Network_Throughput_and_Performance_Guide. [Online; Last modified on 29 August 2012, at 14:18; Last accessed 20-February-2014].

[70] E. Fey, "The Effect of Combining Network and Server QoS Parameters on End-to-End Performance," Master's thesis, KTH Royal Institute of Technology, Teleinformatics, 2000. Available at: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-93516.

[71] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, ch. 6, p. 440. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 5th ed., 2011.

[72] E. Masanet, A. Shehabi, L. Ramakrishnan, J. Liang, X. Ma,B. Walker, V. Hendrix, and P. Mantha, "The Energy Efficiency Potential of Cloud-Based Software: A U.S Case Study." http://crd.lbl.gov/assets/pubs_presos/ACS/cloud_efficiency_study.pdf, June, 2013. [Online; Last accessed 17-March-2014;].

[73] Scott M. Fulton III, "Cloud Data Centers: Power Savings or Power Drain?." http://www.networkcomputing.com/data-center/cloud-data-centers-power-savings-or-powe/240166022. [Online; Last accessed 17-March-2014; Last modified 07-February-2014].

Appendix A

Configuration

/proc/cpuinfo

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 15
model name      : IntelRIntel CoreTMCore2 Duo CPU     E6550  @
   2.33GHz
stepping        : 11
microcode       : 0xb3
cpu MHz : 2327.542
cache size      : 4096 KB
fpu             : yes
fpu_exception: yes
cpuid level: 10
wp              : yes
flags           : fpu de tsc msr pae mce cx8 apic sep mca cmov
   pat clflush acpi mmx fxsr sse sse2 ss ht syscall nx lm
   constant_tsc rep_good nopl pni monitor est ssse3 cx16
   hypervisor lahf_lm dtherm
bogomips        : 4655.08
clflush size    : 64
cache_alignment: 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

Figure A.1: *Processor Details*

xl info

```
host                   : localhost.localdomain
release                : 3.10.25-11.el6.centos.alt.x86_64
version                : #1 SMP Fri Dec 27 21:44:15 UTC 2013
machine                : x86_64
nr_cpus                : 2
max_cpu_id             : 7
nr_nodes               : 1
cores_per_socket       : 2
threads_per_core       : 1
cpu_mhz                : 2327
hw_caps                : bfebfbff
   :20100800:00000000:00000940:0000e3fd
   :00000000:00000001:00000000
virt_caps              : hvm hvm_directio
total_memory           : 3043
free_memory            : 8
sharing_freed_memory   : 0
sharing_used_memory    : 0
free_cpus              : 0
xen_major              : 4
xen_minor              : 2
xen_extra              : .3-26.el6
xen_caps               : xen-3.0-x86_64 xen-3.0-x86_32p hvm
   -3.0-x86_32 hvm-3.0-x86_32p hvm-3.0-x86_64
xen_scheduler          : credit
xen_pagesize           : 4096
platform_params        : virt_start=0xffff800000000000
xen_changeset          : unavailable
xen_commandline        : dom0_mem=1024M,max:1024M loglvl=all
   guest_loglvl=all
cc_compiler            : gcc GCC 4.4.7 20120313 Red Hat 4.4.7-3
cc_compile_by          : mockbuild
cc_compile_domain      : centos.org
cc_compile_date        : Tue Dec 10 20:32:58 UTC 2013
xend_config_format     : 4
```

Figure A.2: *Xen Hypervisor Details*

ethtool -k eth0

```
Features for eth0:
rx-checksumming: on
tx-checksumming: on
        tx-checksum-ipv4: off [fixed]
        tx-checksum-ip-generic: on
        tx-checksum-ipv6: off [fixed]
        tx-checksum-fcoe-crc: off [fixed]
        tx-checksum-sctp: off [fixed]
scatter-gather: on
        tx-scatter-gather: on
        tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: on
        tx-tcp-segmentation: on
        tx-tcp-ecn-segmentation: off [fixed]
        tx-tcp6-segmentation: on
udp-fragmentation-offload: off [fixed]
generic-segmentation-offload: on
generic-receive-offload: on
large-receive-offload: off [fixed]
rx-vlan-offload: on
tx-vlan-offload: on
ntuple-filters: off [fixed]
receive-hashing: on
highdma: on [fixed]
rx-vlan-filter: off [fixed]
vlan-challenged: off [fixed]
tx-lockless: off [fixed]
netns-local: off [fixed]
tx-gso-robust: off [fixed]
tx-fcoe-segmentation: off [fixed]
tx-gre-segmentation: off [fixed]
tx-udp_tnl-segmentation: off [fixed]
fcoe-mtu: off [fixed]
tx-nocache-copy: on
loopback: off [fixed]
rx-fcs: off
rx-all: off
tx-vlan-stag-hw-insert: off [fixed]
rx-vlan-stag-hw-parse: off [fixed]
rx-vlan-stag-filter: off [fixed]
```

Figure A.3: *NIC Details*

ethtool -l / -g eth0

```
Settings for eth0:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Speed: 1000Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 1
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: off
        Supports Wake-on: pumbg
        Wake-on: g
        Current message level: 0x00000007 7
                               drv probe link
        Link detected: yes


Ring parameters for eth0:
Pre-set maximums:
RX:                4096
RX Mini:        0
RX Jumbo:        0
TX:                4096
Current hardware settings:
RX:                256
RX Mini:        0
RX Jumbo:        0
TX:                256
```

Figure A.4: *Other NIC Details*

tcptrace -l filename

```
1 arg remaining, starting with 'test.dmp'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004

36000 packets seen, 36000 TCP packets traced
elapsed wallclock time: 0:00:08.998177, 4000 pkts/sec analyzed
trace file elapsed time: 0:00:06.991139
TCP connection info:
1 TCP connection traced:
TCP connection 1:
        host a:        192.168.1.135:41734
        host b:        192.168.1.104:5001
        complete conn: no        SYNs: 1  FINs: 0
        first packet:  Fri Feb 14 17:15:14.851892 2014
        last packet:   Fri Feb 14 17:15:21.843031 2014
        elapsed time:  0:00:06.991139
        total packets: 36000
        filename:      test.dmp
   a->b:                                b->a:
     total packets:         36000           total packets:             0
     ack pkts sent:         35999           ack pkts sent:             0
     pure acks sent:            1           pure acks sent:            0
     sack pkts sent:            0           sack pkts sent:            0
     dsack pkts sent:           0           dsack pkts sent:           0
     max sack blks/ack:         0           max sack blks/ack:         0
     unique bytes sent: 596896032           unique bytes sent:         0
     actual data pkts:      35998           actual data pkts:          0
     actual data bytes: 596969880           actual data bytes:         0
     rexmt data pkts:           5           rexmt data pkts:           0
     rexmt data bytes:      73848           rexmt data bytes:          0
     zwnd probe pkts:           0           zwnd probe pkts:           0
     zwnd probe bytes:          0           zwnd probe bytes:          0
     outoforder pkts:           0           outoforder pkts:           0
     pushed data pkts:        698           pushed data pkts:          0
     SYN/FIN pkts sent:       1/0           SYN/FIN pkts sent:       0/0
     req 1323 ws/ts:          Y/Y           req 1323 ws/ts:          N/N
     adv wind scale:            0           adv wind scale:            0
     req sack:                  Y           req sack:                  N
     sacks sent:                0           sacks sent:                0
     urgent data pkts:          0 pkts      urgent data pkts:          0 pkts
     urgent data bytes:         0 bytes     urgent data bytes:         0 bytes
     mss requested:          1460 bytes     mss requested:             0 bytes
     max segm size:         26064 bytes     max segm size:             0 bytes
     min segm size:            24 bytes     min segm size:             0 bytes
     avg segm size:         16583 bytes     avg segm size:             0 bytes
     max win adv:           14600 bytes     max win adv:               0 bytes
     min win adv:             115 bytes     min win adv:               0 bytes
```

zero win adv:	0 times	zero win adv:	0 times
avg win adv:	115 bytes	avg win adv:	0 bytes
initial window:	596896032 bytes	initial window:	0 bytes
initial window:	35993 pkts	initial window:	0 pkts
ttl stream length:	NA	ttl stream length:	NA
missed data:	NA	missed data:	NA
truncated data:	0 bytes	truncated data:	0 bytes
truncated packets:	0 pkts	truncated packets:	0 pkts
data xmit time:	6.991 secs	data xmit time:	0.000 secs
idletime max:	142.4 ms	idletime max:	NA ms
throughput:	85378939 Bps	throughput:	0 Bps

Appendix B

Data Files

Throughput Native Linux

```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte default
------------------------------------------------------------
[  4] local 192.168.1.135 port 5001 connected with 192.168.1.101 port 47989
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.09 GBytes    935 Mbits/sec
[  4] MSS size 1448 bytes MTU 1500 bytes, ethernet
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte default
------------------------------------------------------------
[  4] local 192.168.1.135 port 5001 connected with 192.168.1.101 port 47990
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.11 GBytes    952 Mbits/sec
[  4] MSS size 1948 bytes MTU 1988 bytes, unknown interface
[  5] local 192.168.1.135 port 5001 connected with 192.168.1.101 port 47992
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec  1.13 GBytes    967 Mbits/sec
[  5] MSS size 2948 bytes MTU 2988 bytes, unknown interface
[  4] local 192.168.1.135 port 5001 connected with 192.168.1.101 port 47993
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.14 GBytes    975 Mbits/sec
[  4] MSS size 3948 bytes MTU 3988 bytes, unknown interface
[  5] local 192.168.1.135 port 5001 connected with 192.168.1.101 port 47994
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec  1.14 GBytes    979 Mbits/sec
[  5] MSS size 4948 bytes MTU 4988 bytes, unknown interface
[  4] local 192.168.1.135 port 5001 connected with 192.168.1.101 port 47995
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.15 GBytes    982 Mbits/sec
[  4] MSS size 5948 bytes MTU 5988 bytes, unknown interface
[  5] local 192.168.1.135 port 5001 connected with 192.168.1.101 port 47996
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec  1.15 GBytes    984 Mbits/sec
[  5] MSS size 6948 bytes MTU 6988 bytes, unknown interface
[  4] local 192.168.1.135 port 5001 connected with 192.168.1.101 port 47997
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.15 GBytes    986 Mbits/sec
[  4] MSS size 7948 bytes MTU 7988 bytes, unknown interface
[  5] local 192.168.1.135 port 5001 connected with 192.168.1.101 port 47998
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec  1.12 GBytes    957 Mbits/sec
[  5] MSS size 8948 bytes MTU 8988 bytes, unknown interface
```

Throughput dom0

```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte default
------------------------------------------------------------
[  4] local 192.168.1.102 port 5001 connected with 192.168.1.135 port 44063
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.09 GBytes    937 Mbits/sec
[  4] MSS size 1448 bytes MTU 1500 bytes, ethernet
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte default
------------------------------------------------------------
[  4] local 192.168.1.102 port 5001 connected with 192.168.1.135 port 44064
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.11 GBytes    954 Mbits/sec
[  4] MSS size 1948 bytes MTU 1988 bytes, unknown interface
[  5] local 192.168.1.102 port 5001 connected with 192.168.1.135 port 44065
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec  1.13 GBytes    969 Mbits/sec
[  5] MSS size 2948 bytes MTU 2988 bytes, unknown interface
[  4] local 192.168.1.102 port 5001 connected with 192.168.1.135 port 44066
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.14 GBytes    977 Mbits/sec
[  4] MSS size 3948 bytes MTU 3988 bytes, unknown interface
[  5] local 192.168.1.102 port 5001 connected with 192.168.1.135 port 44067
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec  1.15 GBytes    981 Mbits/sec
[  5] MSS size 4948 bytes MTU 4988 bytes, unknown interface
[  4] local 192.168.1.102 port 5001 connected with 192.168.1.135 port 44068
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec    843 MBytes    704 Mbits/sec
[  4] MSS size 5948 bytes MTU 5988 bytes, unknown interface
[  5] local 192.168.1.102 port 5001 connected with 192.168.1.135 port 44069
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec    853 MBytes    713 Mbits/sec
[  5] MSS size 6948 bytes MTU 6988 bytes, unknown interface
[  4] local 192.168.1.102 port 5001 connected with 192.168.1.135 port 44070
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec    863 MBytes    721 Mbits/sec
[  4] MSS size 7948 bytes MTU 7988 bytes, unknown interface
[  5] local 192.168.1.102 port 5001 connected with 192.168.1.135 port 44071
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec    859 MBytes    718 Mbits/sec
[  5] MSS size 8948 bytes MTU 8988 bytes, unknown interface
```

Throughput VM

```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte default
------------------------------------------------------------
[  4] local 192.168.1.104 port 5001 connected with 192.168.1.135 port 47648
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.09 GBytes    934 Mbits/sec
[  4] MSS size 1448 bytes MTU 1500 bytes, ethernet
[  5] local 192.168.1.104 port 5001 connected with 192.168.1.135 port 47652
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec  1.11 GBytes    952 Mbits/sec
[  5] MSS size 1948 bytes MTU 1988 bytes, unknown interface
[  4] local 192.168.1.104 port 5001 connected with 192.168.1.135 port 47653
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.13 GBytes    967 Mbits/sec
[  4] MSS size 2948 bytes MTU 2988 bytes, unknown interface
[  5] local 192.168.1.104 port 5001 connected with 192.168.1.135 port 47654
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec  1.14 GBytes    976 Mbits/sec
[  5] MSS size 3948 bytes MTU 3988 bytes, unknown interface
[  4] local 192.168.1.104 port 5001 connected with 192.168.1.135 port 47655
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1.14 GBytes    981 Mbits/sec
[  4] MSS size 4948 bytes MTU 4988 bytes, unknown interface
[  5] local 192.168.1.104 port 5001 connected with 192.168.1.135 port 47656
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec    792 MBytes    662 Mbits/sec
[  5] MSS size 5948 bytes MTU 5988 bytes, unknown interface
[  4] local 192.168.1.104 port 5001 connected with 192.168.1.135 port 47657
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec    817 MBytes    682 Mbits/sec
[  4] MSS size 6948 bytes MTU 6988 bytes, unknown interface
[  5] local 192.168.1.104 port 5001 connected with 192.168.1.135 port 47658
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec    819 MBytes    684 Mbits/sec
[  5] MSS size 7948 bytes MTU 7988 bytes, unknown interface
[  4] local 192.168.1.104 port 5001 connected with 192.168.1.135 port 47659
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec    820 MBytes    685 Mbits/sec
[  4] MSS size 8948 bytes MTU 8988 bytes, unknown interface
```

CPU usage for 1500 bytes frame

```
Linux 3.10.25-11.el6.centos.alt.x86_64 02/27/2014 _x86_64_        2 CPU

06:41:15 PM       PID    %usr %system  %guest    %CPU   CPU  Command
06:41:16 PM       791    0.00   25.00    0.00   25.00     0  netback/0
06:41:17 PM       791    0.00   24.00    0.00   24.00     0  netback/0
06:41:18 PM       791    0.00   25.00    0.00   25.00     0  netback/0
06:41:19 PM       791    0.00   25.00    0.00   25.00     0  netback/0
06:41:20 PM       791    0.00   25.00    0.00   25.00     0  netback/0
06:41:21 PM       791    0.00   24.00    0.00   24.00     0  netback/0
06:41:22 PM       791    0.00   25.00    0.00   25.00     0  netback/0
06:41:23 PM       791    0.00   25.00    0.00   25.00     0  netback/0
06:41:24 PM       791    0.00   24.00    0.00   24.00     0  netback/0
06:41:25 PM       791    0.00   25.00    0.00   25.00     0  netback/0
Average:          791    0.00   24.70    0.00   24.70     -  netback/0
```

CPU usage for 5000 bytes frame

```
Linux 3.10.25-11.el6.centos.alt.x86_64 02/27/2014 _x86_64_        2 CPU

06:39:20 PM       PID    %usr %system  %guest    %CPU   CPU  Command
06:39:21 PM       791    0.00   19.00    0.00   19.00     0  netback/0
06:39:22 PM       791    0.00   19.00    0.00   19.00     0  netback/0
06:39:23 PM       791    0.00   20.00    0.00   20.00     0  netback/0
06:39:24 PM       791    0.00   19.00    0.00   19.00     0  netback/0
06:39:25 PM       791    0.00   19.00    0.00   19.00     0  netback/0
06:39:26 PM       791    0.00   20.00    0.00   20.00     0  netback/0
06:39:27 PM       791    0.00   19.00    0.00   19.00     0  netback/0
06:39:28 PM       791    0.00   19.00    0.00   19.00     0  netback/0
06:39:29 PM       791    0.00   19.00    0.00   19.00     0  netback/0
06:39:30 PM       791    0.00   19.00    0.00   19.00     0  netback/0
Average:          791    0.00   19.20    0.00   19.20     -  netback/0
```

CPU usage for 9000 bytes frame

```
Linux 3.10.25-11.el6.centos.alt.x86_64 02/27/2014 _x86_64_        2 CPU

06:35:54 PM       PID    %usr %system  %guest    %CPU   CPU  Command
06:35:55 PM       791    0.00   20.79    0.00   20.79     0  netback/0
06:35:56 PM       791    0.00   21.00    0.00   21.00     0  netback/0
06:35:57 PM       791    0.00   21.00    0.00   21.00     0  netback/0
06:35:58 PM       791    0.00   21.00    0.00   21.00     0  netback/0
06:35:59 PM       791    0.00   21.00    0.00   21.00     0  netback/0
06:36:00 PM       791    0.00   21.00    0.00   21.00     0  netback/0
06:36:01 PM       791    0.00   21.00    0.00   21.00     0  netback/0
06:36:02 PM       791    0.00   21.00    0.00   21.00     0  netback/0
06:36:03 PM       791    0.00   21.00    0.00   21.00     0  netback/0
06:36:04 PM       791    0.00   20.00    0.00   20.00     0  netback/0
Average:          791    0.00   20.88    0.00   20.88     -  netback/0
```

Appendix C

Issues

- Best practices for Xen suggests, to pin (CPU affinity) one core to Dom0, i.e. to dedicate a separate core to Dom0, which increases the performance (no other VMs can use that core if it is idle). However, performance decreased when one core is dedicated to Dom0. This might be because only one core was available other VMs (insufficient physical resources).
- Tried using the HP procruve (2626-PWR J8164A) switch, which was available in the laboratory. Though the switch has two GbE ports, it do not support jumbo frames. Another NIC tested, `D-link System Inc DGE-528T Gigabit Ethernet Adapter` supports a maximum 7152 bytes of MTU.

TRITA-ICT-EX-2014:27

www.kth.se