Migrating User Mode Linux to Xen and Kernel Based Virtual Machines

Bachelor Thesis

Adrian Lambeck

August 2007

Supervisors:

Prof. Dr. Dieter Gollmann

Dipl.-Ing. Harald Sauff

Hamburg University of Technology

Security in Distributed Applications

`http://www.sva.tu-harburg.de/`

Harburger Schloßstraße 20

21079 Hamburg

Germany

Declaration

I, Adrian Lambeck, solemnly declare that I have written this bachelor thesis independently, and that I have not made use of any aid other than those acknowleged in this bachelor thesis. Neither this bachelor thesis, nor any other similar work, has been previously submitted to any examination board.

Hamburg, August 2, 2007

Adrian Lambeck

Contents

1 Introduction

Modern education in computer science involves teaching recent security topics. While cryptographic techniques like encryption are important fundamentals, students are eager to attack the latest technologies. To keep up with this fast changing world, universities provide labs for students. A lab allows students to use their theoretical knowledge and gain practical experience. In this case, the lab provides a teaching network so students can experiment for the classes ‘Network Security’ and ‘Application Security’ offered at the Hamburg University of Technology (TUHH). The limitations and necessary resources of a real lab are avoided by introducing the Virtual Teaching Network (VTN). The VTN is based on User Mode Linux (UML) and emulates a real network on a single server machine. A first implementation of the VTN is described in [Sau05] whereas [Sto06] provides extensions to use cases and network setup. A web-based interface for booking sessions is developed in [Zam07].

This bachelor thesis evaluates the migration of the VTN from UML to Xen and Kernel Based Virtual Machines (KVM). The term migration is often used to describe the process of moving a running virtual machine from one host system to another. In this text, migration refers to the process of implementing a virtual machine with a virtualization technology like Xen or KVM.

Xen and KVM belong to the group of Virtual Machine Monitors (VMM). VMMs provide a runtime environment for different, isolated processes (such as operating systems) by managing and scheduling hardware resources.

The migration of the VTN involves three main steps. The first one is the migration of the network infrastructure. Xen and KVM support different Linux network technologies. It is evaluated how the virtual machines (VMs) can be interconnected and which additional features are available. The second question concerns the modification of the existing UML VMs. Besides the migration, it is analyzed which new features can be implemented in the third step.

The first chapter provides a basic overview concerning virtualization technologies followed by Chapter Two which summarizes the impact of virtualization on security. The third chapter provides a more detailed comparison of Xen and KVM. Chapter Four is separated into three parts. The first part explains the steps to migrate the network infrastructure and the virtual machines in general. After making a decision in favor of KVM, the second part explains how the VTN is migrated and integrated into the host system. In the third part, the process of integrating the web application and the host system is described. Chapter Five provides information about extensions to the network. Chapter Six evaluates main network properties of the VTN which are compared to the UML based implementation. A final conclusion of this thesis is given in Chapter Seven.

2 Virtualization

This chapter provides an overview of virtualization technologies and their main properties. It explains why hardware support for virtualization is necessary and gives an overview of its benefits. In the end, a short summary of Xen and KVM is given. Chapter Three provides a detailed comparison.

2.1 Virtualization technology

Virtualization technology in general has been available since 1972 with respect to mainframe computers like IBM System/370 [Gum83]. The reasons for using this technology in the context of small and medium servers to desktops are numerous. Among them are workload consolidation, security aspects, fault tolerance, testing scenarios or cross development. There are several ways to achieve the goal of virtualization. Upon the most popular ones are Dynamic Binary Translation and paravirtualization.

As the name hints, Dynamic Binary Translation translates a program from one instruction set to another. In the case of virtualization, the target instruction set offers the possibility to run operating systems in parallel. Dynamic translation means that this process occurs while the system is running. The access to the instructions offers space for optimizations which are necessary to make up for performance penalties. A variety of modern operating systems can be run without modification [VI07a]. A popular Virtual Machine Monitor using Dynamic Binary Translation is VMware [VI07c] or Microsoft VirtualPC [Mic07].

An operating system which supports a certain paravirtualization technology needs to be ported to this technology. To accomplish this task, the source code and good knowledge of the system is needed since the kernel and device drivers need to be ported. Opposed to Dynamic Binary Translation, the port leads to a static translation because the problematic instructions are replaced at compile time. The availability of the source code leads to a small bandwidth of supported operating systems. Paravirtualization is used in Xen [BDF$^+$03] which utilizes the native virtualization interface of the Linux kernel. It allows hypervisors to override instructions necessary for virtualization thus configuring the kernel for a specific hypervisor without changing the code base.

2.2 Hardware virtualization

Two x86 processor manufacturers, Intel and AMD, introduced hardware support for virtualization in 2006. Other than the x86 instruction set, the extensions are not standardized. Intel developed the Intel-VT 'Vanderpool' technology whereas AMD introduced the security and virtual machine (SVM) architecture 'Pacifica'. Both technologies provide an interface for a VMM, also called hypervisor, by introducing a guest mode for operating systems. If an operating system is run in guest mode, it is directly executed on the processor. A processor supporting hardware virtualization intercepts privileged instructions[1] and calls the VMM to handle these. Privileged instructions need a special treatment because they have the potential to break isolation or protection of the system.

The approach of hardware support solves the biggest disadvantages of Dynamic Binary Translation and paravirtualization at the same time. A guest operating system has no possibility to discover if it is running in a virtual machine and therefore does not need to be changed. Since the technology is supported natively by the processor, there is only a small performance penalty compared to a native installation of an operating system. A technical summary of 'Vanderpool' can be found at [UNR$^+$05] whereas [AMD07] informs about 'Pacifica'.

Depending on the type of workload, a software-based VMM without hardware support can generate better performance than a hardware-based VMM. The latter runs near native performance when computing is the task. It is slower performing I/O operations, process creation or rapid context switches. This is partly due to the memory management. The host system has to separate and isolate the memory of guest systems. When memory is addressed by a guest, the request is translated from virtual guest memory to physical guest memory. The bottleneck occurs when the physical guest memory is translated to the physical host address. The current hardware support for this translation is not sufficient when compared to a software-implemented Memory Management Unit (MMU) because the hardware MMU supports only the first level of translation. The lack of support for MMU is explained in greater detail in [AA06].

This summary of the current state in hardware virtualization shows that the first generation of x86 extensions provides well-working solutions to functional problems while performance issues need to be addressed in the next hardware generation.

2.3 Security aspects of virtualization

This section addresses different security scenarios concerning virtualization technologies. The first section describes aspects of software security concerning the Virtual Machine Monitor whereas the second part explores how existing threats are taken to the next level. The last part describes new security problems originated through hardware virtualization.

2.3.1 The Virtual Machine Monitor

The security of a VMM is gaining importance since it is targeted by different malware for different reasons. In security research, virtualization is used to analyze suspicious pieces of software in an isolated environment as described in [CWdO$^+$06]. In this case, a virtual environment is used to find out about the timetable of different malware so it can be predicted when certain events are triggered. In this context, it is crucial that a software program can not detect whether it is running in a virtualized environment or not. If the malware knows about the virtual environment, it might make analysis impossible. On the other hand, if the malware knows about the specific VMM and its security holes, it might break isolation or infect the host system. The detection of and attacks on virtual machine

[1] i.e. CPUID, MOV from/to CR3, MOV to CR0/CR4, RDMSR, WRMSR, HLT, INVLPG, MOV from CR8, MOV DR, and MWAIT

emulators is described in [Fer06]. The VMM is not the only source of security problems which is shown in the next example.

If Xen runs a guest using paravirtualization, it provides software interfaces to the guest systems so the developers of Xen are responsible to maintain security. When running unmodified guests, Xen and KVM use QEMU [Bel05] to emulate I/O devices. As it is shown in [Orm07], this opens new security issues which are not directly related to VMM but allow to compromise the host system. An advanced security architecture like the secure hypervisor [SJV$^+$05] (sHype), implemented in Xen, might be able to solve some of these threats. As a result, a VM should always be treated as a real computer meaning that common security practices like deactivating unnecessary services, i.e. USB device drivers, should be followed in order to reduce the attack surface.

2.3.2 Existing threats at a new level

Developments in software and hardware technology often increase productivity but might also open new security holes. This is especially the case in a scenario where users make frequent usage of different VMs, exchange special purpose VMs with each other or save VMs to portable storage devices. Garfinkel and Rosenblum [GR05] describe and analyse the security problems that arise with this scenario. They point out that these threats are not totally unknown since mobile computers, Virtual Private networks and Windows Restore have been available. In most organizations these are not widely deployed or used infrequently whereas VMs might be widely adopted. A short overview of the main points by Garfinkel and Rosenblum is given. More detailed examples and also recommendations how to deal with these can be found in the paper.

VMs can be shut down, suspended, resumed or restarted. This adds a new dimension to the network of an organization since machines are added or disappear all the time. A typical security case is a machine which got infected with some kind of worm. By the time the IT staff has found out about the worm, the machine might already be shut down. It is not possible to locate where the machine was run and therefore difficult to clean the machine.

The latest VMMs provide a snapshot feature, also called roll-back, to recover a previous state of a VM. This can be useful when testing software. When using the roll-back functionality, all the changes up to a specific point are lost including virus scanner updates or operating system patches. The roll-back functionality is not anticipated by software products or operating systems nor do users record all the changes they made with respect to security. In this context, one needs to think about the validity of encryption keys, disabled accounts or the reintroduction of worms that were deleted before the roll-back.

The mobility of VMs is a big advantage in general since the VM consists of normal files. The files can be stored on different media like CD-ROM, DVD, tape and hard disk. Storing the VM files on a file server allows users share the VMs within an organization. The disadvantage is that if a file server is compromised, one does not know what might have happened to personal or sensitive data stored in the virtual machines.

2.3.3 A new scenario

A new scenario is presented using hardware virtualization technology. With this attack, the operating system is virtualized on the fly without interruption. The attacker gains total control of the machine thus circumventing any security mechanisms of the operating system. Proof on concept rootkits for this kind of attack work using MacOS X and Intel VT [Zov06] or Windows Vista and AMD SVM [Rot06]. Besides compromising the whole machine, it is also a problem that the rootkit can not be detected by the operating system. The hypervisor does not change the system and does not operate within its scope. The attack can only be successful if the system is not already virtualized by a hypervisor. In this case, it is the only possibility to exploit a vulnerability in the already running hypervisor which is the topic of section 2.3.1.

3 Comparing Xen and KVM

This chapter provides an overview of Xen and KVM by giving some background information in the first part. In the following section the hypervisors are compared. A hypervisor can benefit directly from the hardware virtualization features which are compared in the third part. In the fourth section, the supported I/O devices are compared. They are followed by a comparison of the networking capabilities. In the end, tools supporting the creation, usage and monitoring of Xen and KVM VMs are introduced.

3.1 Xen

Xen is developed at the University of Cambridge as part of the XenoServer project [FHH+03]. A first version was published in 2003. Paravirtualization technology is used but x86 hardware virtualization technologies are also supported. As shown in Figure 3.1, Xen is a layer directly above the hardware. It provides basic control operations and allows access to them through an interface. The interface is accessed by the domain0 (dom0) which is created at boot time. Xen grants access privileges to dom0 to allow control over the hardware.

The guest domains are called domU and run in an unprivileged mode. A split driver architecture is implemented by offering a back-end device driver in dom0 and a front-end driver in domU. Requests concerning I/O in domU are directly sent to dom0 which checks the input and executes the necessary operations. The result of an operation takes the same path backwards to domU.

3.2 KVM

KVM has been developed by Qumranet since early 2006. It supports hardware virtualization, but unofficial paravirtualization patches are available [Mol]. KVM does not include a hypervisor. Instead, it makes the Linux kernel a hypervisor. For this reason, KVM benefits directly from any improvements made to the Linux kernel while keeping the hypervisor small. In this text, KVM plus the Linux kernel is referred to as the hypervisor.

3.3 The hypervisor

Xen and KVM differ in their strategy to implement a hypervisor. For Xen the Linux kernel and device drivers must be modified and compiled. Depending on the hardware of the host system, patches of certain device drivers might be necessary. As of the time of writing, Xen supports Linux 2.6.16 (March 2006) while 2.6.22 (July 2007) is the most recent version. New features, hardware support and bug fixes introduced during this period are not available to Xen. Besides paravirtualization in Xen, a so called Xen Hardware Virtual Machine (HVM) is available. Using the HVM mode allows Xen to run operating systems not ported to Xen. Even though Microsoft Windows was ported to Xen, the results

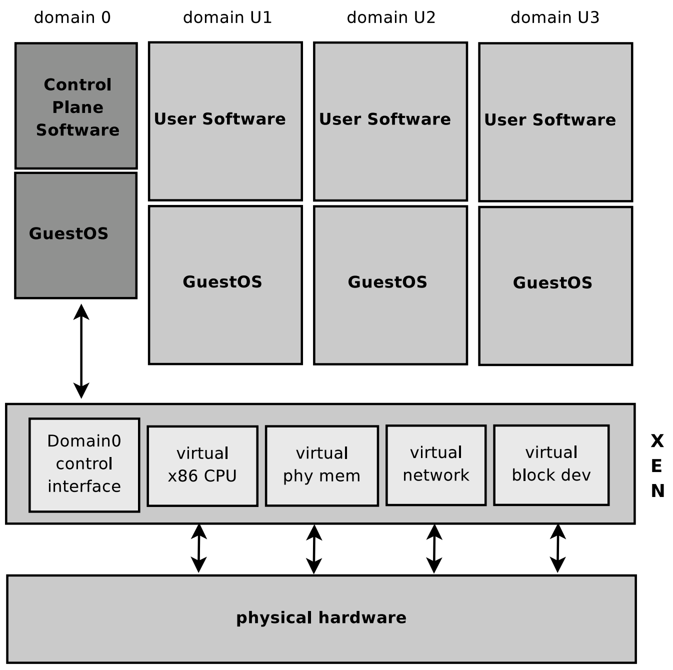


Figure 3.1: Xen based architecture, reproduced from [BDF+03]

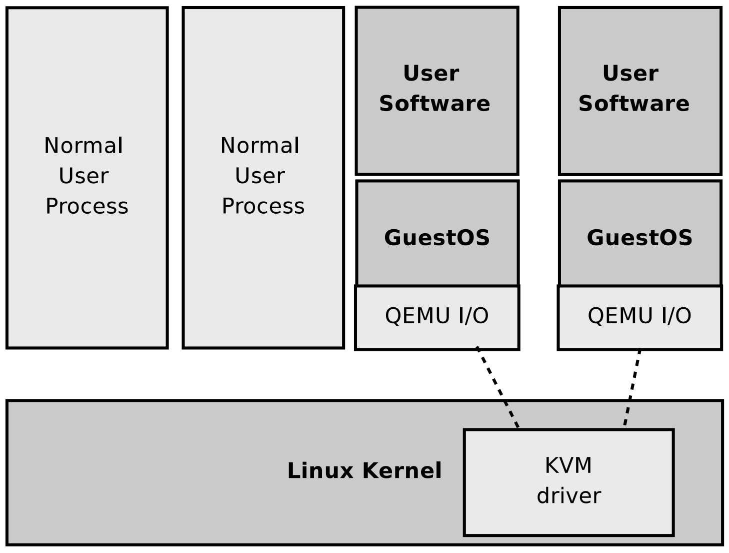


Figure 3.2: KVM based architecture, reproduced from [Qum06]

were not made public due to licensing restrictions. Running unported Windows on Xen is possible by using the HVM mode. In this case, some components [DLM+06] of Xen are exchanged by the QEMU emulator. It is important to understand that Xen provides virtual devices when using paravirtualization and emulated devices when guests are run in HVM mode. The virtual device is an abstract one which can only be used with Xen while the emulated device works by using the device driver from the Linux kernel. In this text, paravirtualized guests are used.

KVM is implemented as Linux kernel modules loaded during runtime thus limiting KVM to Linux host systems. It is split up into an architecture independent KVM module and architecture dependent modules kvm_intel and kvm_amd. No changes to the kernel are necessary in order to support KVM thus all device drivers are available for the host and guest system. Compilation into the kernel is an option. The hypervisor handles guest mode events from the processor only. Other tasks like scheduling and memory management are done by the Linux kernel which officially includes KVM since version 2.6.20. The hardware for guest systems is emulated using QEMU.

The CPU scheduling algorithm is a good example to illustrate the advantages and disadvantages of the two approaches. The Xen hypervisor implements the Borrowed Virtual Time (BVT) algorithm which is chosen by the Xen team because of short latency dispatching. KVM relies on the Linux kernel algorithm. While the Xen team needs to maintain the algorithms, it allows easy extension or changes. KVM does not need to bother with maintenance since it is done by the Linux kernel maintainers. This also means that optimizations with respect to a virtual environment might be hard or impossible to be enforced by the KVM developers.

Modern processors often provide more than one core. Xen takes advantage of these by supporting Symmetric Multiprocessing (SMP) for guest systems. SMP support for KVM hosts is available through the Linux kernel while SMP support for guest systems is scheduled for Linux 2.6.23.

Fault tolerance is more and more evolving as being one of the big advantages of using virtualization. Both hypervisors support migration in the classical sense. This means that a VM is running on a host machine and can be transfered to another without being shut down. Tests using Xen show downtimes during this process between 60 ms and 3.5 s [CFH+05].

3.4 Hardware support for virtualization

Supporting the latest hardware virtualization technology is important for the success of a VMM. Xen supports both VT-x (IA-32) and VT-i (Itanium) whereas KVM lacks support for VT-i. Both implement AMD SVM as summarized in Table 3.1. The Intel-VT technology in Xen was implemented by Intel's Core Software Division described in [DLM+06].

Intel adds several optimizations concerning I/O, i.e. IDE DMA, NIC and VGA, in QEMU. Also the Bochs BIOS [But] is extended. KVM users profit from this effort since they make use of these emulators (see 3.5). VT-i support for KVM might be easier to be implemented since Intel has already extended available open-source projects and published the results.

	SVM	VT-x	VT-i
Xen	x	x	x
KVM	x	x	-

Table 3.1: supported hardware virtualization technologies by Xen and KVM

3.5 Input/Output

The I/O features of Xen depend on whether a paravirtualized or fully virtualized guest (HVM) is run. In case of paravirtualization, Xen provides device abstractions called virtual devices drivers. Data is transfered using shared-memory and asynchronous buffer rings which allow data to move vertically through the system. Access to the buffer ring is implemented using consumer-producer pointers. The buffer ring contains references to shared memory locations instead of the data itself which causes a speed advantage. KVM does not implement such a functionality which is another performance problem besides MMU issues explained in section 2.2. The upcoming Intel Virtualization Technology for Directed I/O [AJM$^+$06] addresses these issues and promises better performance.

3.6 Network

The network is another kind of Input/Output. It is important to notice that neither Xen nor KVM provide network technology itself. Strictly speaking, they support certain network interfaces either provided by the Linux kernel or QEMU. The virtual networking devices found in Xen use a split driver architecture mentioned in 3.1. This architecture is used internally by Xen and does not connect to the host system. One possibility for KVM and Xen guests is the connection to the host by using ethernet bridges. This is a common solution also used in [Sau05]. It performs well and is used in many cases to connect several guest systems to one host.

Features available only to KVM guests are TUN/TAP devices, used in [Sto06], to exchange packets directly with the host kernel. The TAP device emulates a virtual ethernet device while the TUN device simulates a network tunnel to the host system. The combination of TUN/TAP devices allows user-space programs to communicate with the ethernet devices.

A lot analysis and research of Xen and for example the network capabilities has been done. Menon et al. [MST$^+$05] analysed the network performance overhead and published several improvements [MCZ06]. The result is a dramatic improvement of throughput. It could be shown that software implementations of features only found in feature rich network cards can improve the performance of virtual network devices.

The process to establish a unified virtual device driver for the most common virtualization technologies is started in [MSO07]. The result could be a common virtual ethernet device in the Linux kernel tree. Differences as described above could still exist but not on the interface of the guest machine. This allows better connectivity through all virtualization technologies.

3.7 Tools

The following tools support the operation of virtual machines. The libvirt project is mentioned for completeness but is not used in this thesis.

3.7.1 Xen

The Xen distribution includes a command called *xm* to start, stop and manage guest domains. Besides these necessary commands, physical memory allocations, control of scheduling parameters and the access to the host's devices can be controlled. Using this mechanism, it is possible to resize the memory of a guest machine without rebooting it. With the release of Xen 3.1, this feature is available for fully virtualized guests.

3.7.2 KVM

The KVM distribution does not include tools to manage VM instances. Since every VM is a user process, the standard Linux process management tools such as

- `kill` - to suspend or terminate
- `top` or `ps` - to query process information
- `taskset` - to run a VM on a specific CPU

can be applied. Features such as dynamic memory allocation are not available. To gain information about the state of a virtual machine, QEMU monitor can be used. It is possible to change removable media like CD-ROM drives or USB devices. A unique feature in this context is the possibility to print information about CPU registers and attach a debugger to the virtual machine.

3.7.3 Virtual Distributed Ethernet

VDE is part of the Virtual Square project [Dav07] by Renzo Davoli from the University of Bologna. Components used in this thesis are VDE switches, cables and the wirefilter. Vde_switch is a software switch to interconnect VMs. It is partly based on the code of uml_switch used in the UML based network. Compared to uml_switch, vde_switch is extended and supports UML, QEMU, BOCHS and native Linux. It can not be used with Xen. The switch can also act as a hub which is crucial for the VTN since it eases the traceing of network traffic.

Cables are used to connect switches with each other. They can be compared to uplink cables used for stacked switches. A wirefilter can be attached to every cable in order to change the properties such as packet loss, delay, noise factor or bandwidth.

Every virtual machine connects the network interface card directly to the corresponding vde_switch. It means that no VDE cable is used and therefore the wirefilter can not be applied to the VM-vde_switch connection. Note that the direction of packets flowing through the filter can be distinguished between left-to-right (LR) and right-to-left (RL). The distinction of the direction allows the introduction of new network elements. A simple example is a satellite link consisting of a slow link with low latency in one direction (c.f. ISDN) and a high bandwidth high latency connection in the other.

A possible application for the wirefilter is DNS cache poisoning [Ste07]. This form of attack is not possible in the UML based network because the authoritative nameserver is able to answer very fast. The probability of a successful attack is higher if the delay between the attacked and authoritative nameserver is long enough.

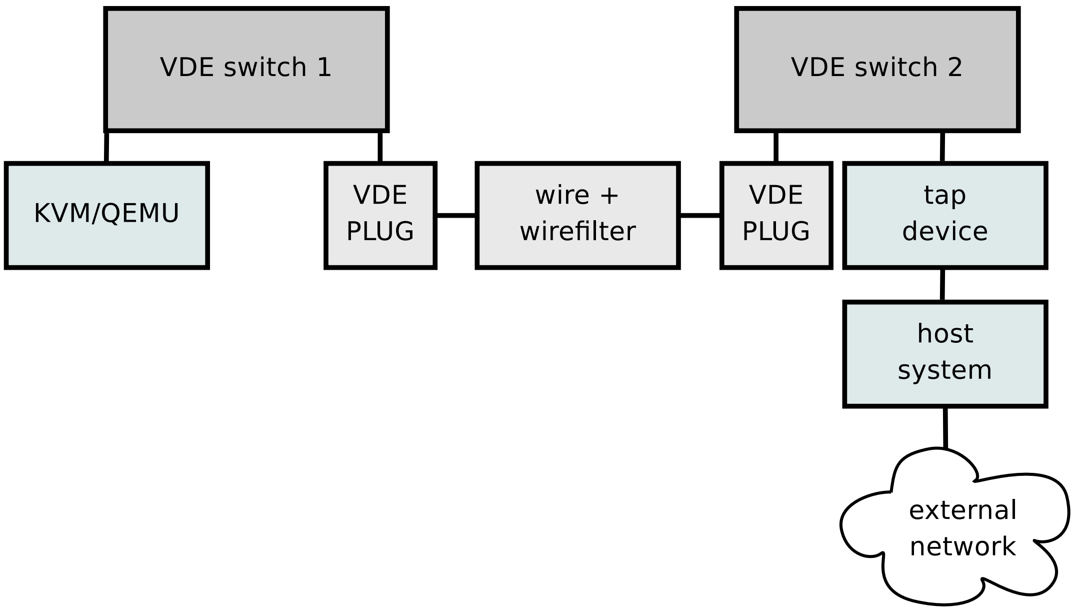


Figure 3.3: setup of two VDE switches with a wirefilter

3.7.4 libvirt

The libvirt [Vei] project aims to provide operations to manage virtualized domains on a host. Different technologies like Xen and KVM/QEMU are accounted for. There exists a graphical interface called virt-manger [DB] to visualize the management of virtual instances. Virtual machines are configured by using XML files. The main advantage is the common interface for different technologies. In this way, the user can always use the same command to start or stop any virtual machine without knowing the details of the VM. In the context of this thesis, libvirt is not used due to the current status of the libvirt project. It is worth mentioning because of the potential benefits mentioned above.

4 Migrating the Virtual Teaching Network

This chapter starts with a short description of the hardware environment. In the second part, a summary of [Sto06] is given. The third section describes which changes are necessary to migrate the network and the virtual machines. It concludes with a decision for the KVM technology used for the integration of the network described in the fourth section. The installation of the web-based interface developed in [Zam07] is explained in the fifth section. The last part explains the setup of the firewall on the host machine.

4.1 The environment

This project is realized on a Dell Power Edge 860 server with an Intel Xeon 3060 processor featuring Intel-VT-x, 4GB ECC DDR2 RAM, two 160GB SATA hard drives and two 1 Gbit/s Broadcom Tigon3 network cards.

4.2 The existing network

The UML based network is running on a Intel Pentium 4 processor with 1700 MHz and 256KB Cache, 512 MB RDRAM and one 34GB SCSI hard disk drive.

The current topology is shown in Figure 4.1. Guest systems run Debian stable 3.1 with kernel version 2.6.15. Dynamic routing is based on the Routing-Information-Protokoll (RIP) and Open-Shortest-Path-First (OSPF) routing protocols implemented by the QUAGGA Routing suite [qua] version 0.99. The current implementation of the virtual teaching network uses several GUI based tools like etherreal. The DISPLAY environment variable of the X-Windows system is exported to make GUI tools available to students via ssh. The network connection to the host is implemented using TUN/TAP devices. Machines are interconnected using uml_switches running in hub mode.

4.3 The migration of the network

This section describes the necessary steps to run a UML VM using Xen and KVM. The decision for a specific technology is made upon these experiences. The integration of all parts into a working VTN is described in section 4.4.

4.3.1 The host system

The host system runs Gentoo Linux 2006.1 [FIa]. It is installed as described in the installation guide [FIb] dated April 13th 2007. Gentoo Linux distributes its software using source packages instead of compiled binary archives. It uses a powerful package system with the latest software packages available. This is an advantage when working with fast evolving software. This is especially the case if

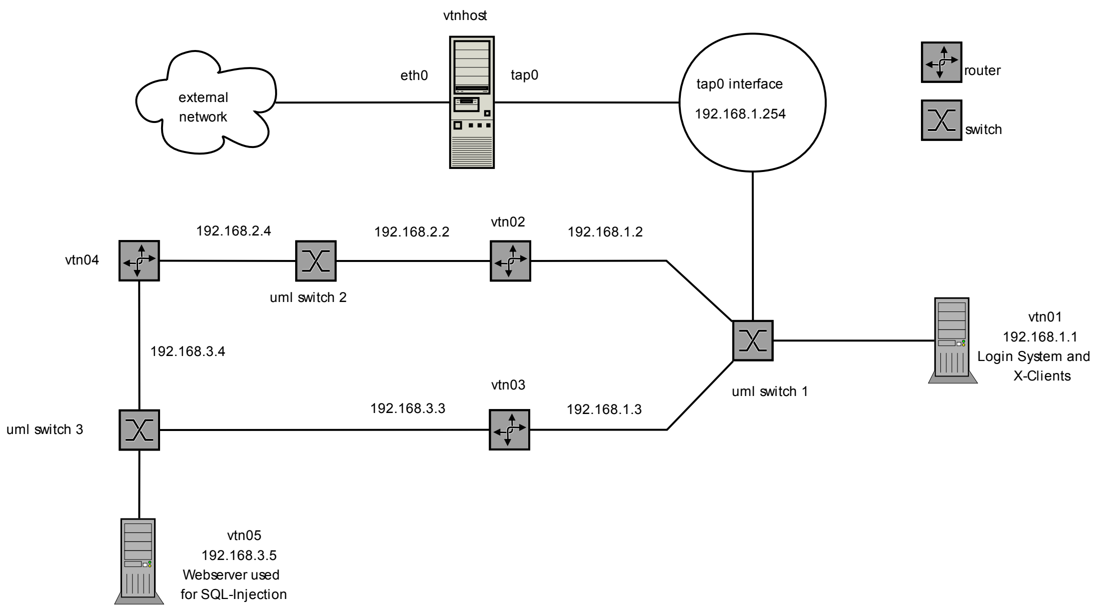


Figure 4.1: topology of the virtual teaching network, reproduced from [Sto06]

patches need to be applied or compilation options have to be changed. Debian Linux on the other hand is very useful in virtual machines. It does not need a compiler and the tools and libraries associated with it. The installation of binary packages saves space and compile time.

In order to provide the same setup for Xen and KVM, the same installation is used for both. For this reason, they are equally affected by any changes made to the core system. The main difference lies within the kernel.

At first, a minimal kernel is created which is also used as backup kernel. Xen works with kernel version 2.6.16.49 while KVM uses 2.6.20. When booting the machine, one can choose between the minimal, Xen and KVM kernel. Unless it is mentioned in the text, compilation is done using the compiler flags `-O2 -march=nocona -pipe -mno-tls-direct-seg-refs`. To take advantage of the two CPU cores when compiling sources, two compiler processes are started in parallel using `make -j2`. Please note that USB drivers are disabled since they are known to cause problems with Dell Power Edge servers.

4.3.2 Installation of Xen

The information given in [LW07] is used to install Xen. The following changes and configurations are made.

Overlay

To compile unofficial ebuilds from various authors, Gentoo uses so called overlays. This is necessary since Xen 3.0.4 is not available for Gentoo unless overlays are used. In this case, the overlay of Michael Marineau [Mar] provides these files. It is added by using the following commands:

```
~$ emerge layman
~$ layman -a marineam-xen
```

dom0 kernel

The dom0 kernel features the device drivers to access the hardware. It is located in /boot/vmlinuz-2.6.16.49-xen. Since the installation utilizes grub as bootloader, the following entry is used:

```
title Xen 3.0.4 / Linux 2.6.16.49
root (hd0,0)
kernel /xen.gz dom0_mem=256M
module /vmlinuz-2.6.16.49-xen root=/dev/sda3 netloop.nloopbacks=16 max_loop=16
```

Listing 4.1: a dom0 entry in /boot/grub/grub.conf

The last line contains two important kernel commands which change the number of loopback devices. By default the kernel provides eight such devices which is not enough for the VTN. The command `netloop.nloopbacks=16 max_loop=16` changes the number of available devices to 16. The VTN with the current topology utilizes nine devices and leaves room for additional hosts.

domU kernel

The domU kernel is used by all Linux VMs to communicate with the dom0 kernel and runs the guest system. For this reason it includes other device drivers than dom0. The domU kernel needs to match the previous UML kernel regarding features like network options. It is pointed out in [Sto06], page 30, that the .config file, holding the kernel configuration, needs to be stored in case the system is upgraded later on. This is the case when migrating the VMs, but the .config file can not be found. The proc filesystem of the UML machines reveals basic information about the kernel which is then used to build a kernel with similar features. The domU kernel is located in /home/vtn/xen_kernel/xen-2.6.16.49-xen-domU. The latest .config file is now stored in /proc/config.gz.

4.3.3 Migrating the network to Xen

This section describes the setup of the network infrastructure used by the Xen VMs. Migration of the network is done in two steps.

Defining the network infrastructure

In the default configuration, Xen uses one bridge to connect VMs to a physical ethernet device. This standard setup is enough for most network configurations where several VMs are connected to the same device. This setup is not satisfying for the Virtual Teaching Network. The existing UML network can not be used since it uses other technologies not supported by Xen. To set up the default network, Xen uses the following scripts specified in /etc/xen/xend-config.sxp:

1. network-bridge - used to create the default bridge
2. vif-bridge - called when a guest VM is booted to connect the virtual interfaces (vif) to a bridge

In order to build the virtual network, xend-config.sxp is changed to call vtn.sh instead of network-bridge:

```
1 #!/bin/sh
2 dir=$(dirname "$0")
3 "$dir/network-bridge" "$@" vifnum=0
4 "$dir/vtn_lan.sh" "$@"
5 "$dir/vtn_firewall.sh"
```

Listing 4.2: creating bridges for Xen with /etc/xen/scripts/vtn.sh

The original Xen network-bridge script is called in line three to set up xenbr0 and attach dom0. If updates of Xen are applied, they do not affect the networking capabilities of dom0. Line four executes the vtn_lan.sh script. It is based on network-bridge and creates xenbr1 and xenbr2. For comparison, xenbr0 to xenbr2 substitute the uml_switches in Figure 4.1. In the last line, the firewall rules defined in vtn_firewall.sh are applied. The firewall script is based on the old script from the UML network. Details are explained in section 4.6.

Connecting VMs to the network

When starting a VM, Xen calls the vif-bridge script to attach the virtual interfaces to a specific bridge. The configuration file of each VM defines the interfaces and the bridge they are connected to. As an example, the network section of xen02 is shown:

```
vif =
[ 'mac=00:FF:00:00:01:02, bridge = xenbr0, ip = 192.168.1.2',
   'mac=00:FF:00:00:02:02, bridge = xenbr1, ip = 192.168.2.2' ]
```

Listing 4.3: excerpt from /home/vtn/vtn02/vtn02.cfg

This machine is equipped with two virtual interfaces. One has the MAC address 00:FF:00:00:01:02 and the IP address 192.168.1.2. This interface is connected to xenbr0. The other interface is placed in another subnet with IP address 192.168.2.2 and MAC address 00:FF:00:00:02:02. It is connected to a different bridge namely xenbr1.

4.3.4 Migrating a VM from UML to Xen

The **UML** VMs consist of three components:

1. start-stop - a script used to start and stop a virtual machine
2. a file with the root filesystem in ext3 format
3. a swap filesystem

To make this VM usable with Xen, it is necessary to write a configuration file. A minimal version looks like the following:

```
1 name = vtn01
2 vif = [ 'mac=00:FF:00:00:01:01, bridge = xenbr0, ip = 192.168.1.1' ]
3 disk = [ 'file:/home/vtn/vtn01/vtn01.img.ext3, hda1, w',
4           'file:/home/vtn/vtn01/swap.file, hda2, w' ]
5 root = "/dev/hda1 ro"
6 kernel = "/home/vtn/vmlinuz.2.6.16.46-xen-domU"
7 memory = 128
```

Listing 4.4: minimal Xen configuration file

Line three describes which files are used as hard disks, the guest sees them as the device specified after the first comma, also the mode how the guest can access the device is set. In this case there is a root filesystem which is mounted in write mode as hda1. The second file is a writeable swap file mounted as hda2. Line five specifies the domU kernel to be used for this VM. The VM can now be started using the command `xm create xtn01.cfg`. To shut down a VM, the command `xm shutdown vtn01` is used.

4.3.5 Installing KVM

KVM is included in the official Linux kernel since version 2.6.20. Nevertheless it is still under heavy development causing several releases a week. For this reason the kernel switch ‘Device Drivers -> Virtualization -> Kernel-based Virtual Machine (KVM) support’ is **NOT** activated. In this example, release 28 is compiled as a module, the system is configured and KVM is loaded into the kernel:

```
~$ emerge =gcc-3.4.6-r2
~$ emerge app-emulation/kqemu

~$ tar -xzf kvm-28.tar.gz
~$ cd kvm-28
~$ ./configure \
~$  --prefix=/opt/kvm \
~$  --qemu-cc=/usr/i686-pc-linux-gnu/gcc-bin/3.4.6/gcc
~$ make
~$ make install

~$ cp -rp kvm-28/scripts/65-kvm.rules /etc/udev/rules.d/
~$ groupadd kvm
~$ usermod --groups=vtn,kvm vtn

~$ modprobe kvm-intel
~$ modprobe kqemu
~$ echo kqemu >> /etc/modules.autoload.d/kernel-2.6
```

```
19 ~$ echo kvm−intel >> /etc/modules.autoload.d/kernel−2.6
```

Listing 4.5: installing KVM on the host system

The KVM module is compiled using gcc-4.1.1 whereas QEMU can only be compiled using gcc-3.4.6. The appropriate gcc version and the QEMU accelerator module are installed in the first two lines followed by the configuration of the sources to install into /opt/kvm/.

The file 65-kvm.rules in line 12 contains a udev rule to allow members of the group *kvm* to use the device /dev/kvm without administrative privileges. The group *kvm* is created and the user *vtn* is added to it in the following two lines.

Loading the modules on every boot is done by lines 18 and 19. Because of the high update frequency of KVM, the script /root/kvm/build.sh can be used to manage the update process of KVM. Please set the variable `KVM_VERSION` to the KVM module version you would like to install. The script becomes unnecessary when KVM is compiled into the kernel or a Gentoo ebuild file becomes available.

QEMU in the KVM context

QEMU is capable of emulating a PC system including different processor types[1] and peripherals. Please note that QEMU used in KVM does not provide Dynamic Binary Translation included in official QEMU distribution. Instructions are directly executed on the processor. Due to the virtual network, it is possible to attach instances of the official QEMU distribution. Even though the performance of such a system is not comparable to KVM, it offers new possibilities. Using this approach, a Sparc based Solaris system could be integrated. Embedded router software running on emulated ARM or MIPS processors might replace the current Linux-based routers. The possibility of emulating these architectures might proof valuable since exploits can be related to a certain hardware architecture. QEMU can also convert different disk image types. This feature allows migration from other technologies since the most popular types, including VMware, are supported.

4.3.6 Migrating the network to KVM

Concerning network support, QEMU supports several User Mode Linux tools. In this scenario, the uml_switch is replaced by vde_switch. The VDE distribution needs to be installed first by issuing `emerge vde`. A vde_switch is created using the following command:

```
~$ vde_switch \
        −−sock /home/vtn/vde/switch1.ctl \
        −−mgmt /home/vtn/vde/switch1.mgmt \
        −−pidfile /home/vtn/vde/switch1.pid \
        −−numports 5 \
        −−tap tap0 \
        −−daemon \
        −m 666
```

Listing 4.6: starting a vde_switch for KVM

[1]ARM, Sparc 32bit/64bit, PowerPC, ColdFire, MIPS

The option in line two specifies the access socket to the switch. Other switches or QEMU instances can connect to the switch by referring to this directory. The second option declares the management interface for this switch. It is not mandatory for the switch to work but can be useful to get and set properties of the switch using the `unixterm` utility. The pidfile in line four contains the process identifier which is used to shut down the switch.

By default, a switch is started with 32 accessible ports. The option in line five decreases this number to five ports. The switch in Listing 4.3.6 uses the tap0 interface of the kernel to exchange network packets with the host system. Other QEMU instances use this switch as a gateway. This is possible because an IP address is assigned to tap0 **after** the switch is connected to it. The daemon flag in line seven ensures that the switch keeps running after the starting process has ended. The last line sets the permissions to allow global access to the switch for everybody.

4.3.7 Migrating a VM from UML to KVM

QEMU emulates different hardware (see A.1) than used in the host system. This makes a custom kernel necessary. It is located in /home/vtn/kvm_kernel/bzImage to be used by all VMs thus saving space and making kernel upgrades easier. The script located in /usr/src/compile_kvm_kernel.sh is used to compile and deploy the KVM guest kernel using kernel configuration files located in /root/TOOLS/.

The setup used in this context calls the `vdeq` program first. It is a wrapper for QEMU which provides the connection to the switches. It is not necessary to change the VM in order to support KVM. There exists no configuration file.

```
~$ vdeq \
        /opt/kvm/bin/qemu \
        −hda /home/vtn/vtn01/vtn01.img.ext3 \
        −hdb /home/vtn/vtn01/swap.file \
        −snapshot \
        −boot c \
        −kernel /home/vtn/kvm_kernel/bzImage \
        −append root=/dev/hda \
        −M pc \
        −m 128 \
        −k de \
        −vnc :1 \
        −net nic,macaddr=00:FF:00:00:01:01 \
        −net vde,sock=/home/vtn/vde/switch1.ctl \
        −pidfile /home/vtn/vtn01/vtn.pid \
        −no−reboot
```

Listing 4.7: starting a KVM VM in the VDE environment

The following is a short explanation to give an impression how QEMU is used in this context. Please refer to the QEMU command reference [Bel] for more details.

In lines one to six, QEMU is instructed to start the VM vtn01 in snapshot mode which does not memorise changes to hard disk hda or the swap device hdb. Line seven provides the link to the custom kernel which is started using the kernel options provided in the following line. QEMU is instructed to emulate a PC with 128MB of RAM and a German keyboard layout (lines nine to eleven). By default, QEMU uses the sdl library to offer access to the machines display. This works fine if the process is started in a graphical environment. This is not the case in this setup. As an alternative, line 12 creates a VNC display on port 5901 which can be accessed using a VNC client program. In lines 13 and 14, the network interface card is assigned the MAC address 00:FF:00:00:01:01 and connected to the switch. To manage the VM later on, the Process ID (PID) is written into a file. It is not possible to reboot this VM. It is shut down in case of a reboot.

4.3.8 Overview

After describing how to migrate the VTN to Xen and KVM, it is decided to use KVM in the final implementation. Both technologies are able to run the VTN and extend it by supporting other operating systems like Microsoft Windows. The supported network features are the main difference. In this area, there is no other choice than taking KVM because of the support of VDE. Due to the architecture of Xen, it is not possible to attach a vde_switch. The author is not aware of other possibilities to use hubs in Xen networks and requests to the Xen mailinglist did not provide hints to other solutions.

A solution might become available when the kvm-xen module, developed in [HDL07], supports I/O. The new module runs a Xen VM using KVM and QEMU which offers VDE support.

4.4 Integrating the network

This section describes the process of integrating the VTN with the host operating system and the web front-end. The original topology has been altered by adding a new VM hosting the web interface. It is placed into an additional subnet (192.168.72.0) to separate it from the VTN. Figure 4.2 shows additional changes described in Chapter Five.

4.4.1 Integrating the VTN network infrastructure

The network infrastructure for the VTN is created when the host system starts. The script to manage the automatic start and shut down of the network infrastructure is located in /etc/init.d/vtn-net and utilizes the Gentoo Linux init facilities. The init facilities manage the boot and shut down process of the system by starting and stopping services in the proper order. The call dependencies are visualized in Figure 4.3 to provide an overview.

4.4.2 Integrating the VTN virtual machines

The virtual machines are started if a session is scheduled by the web application. The main script is located in /home/vtn/vtn.sh. It uses the commands described in section 4.3.7 and is not explained for this reason.

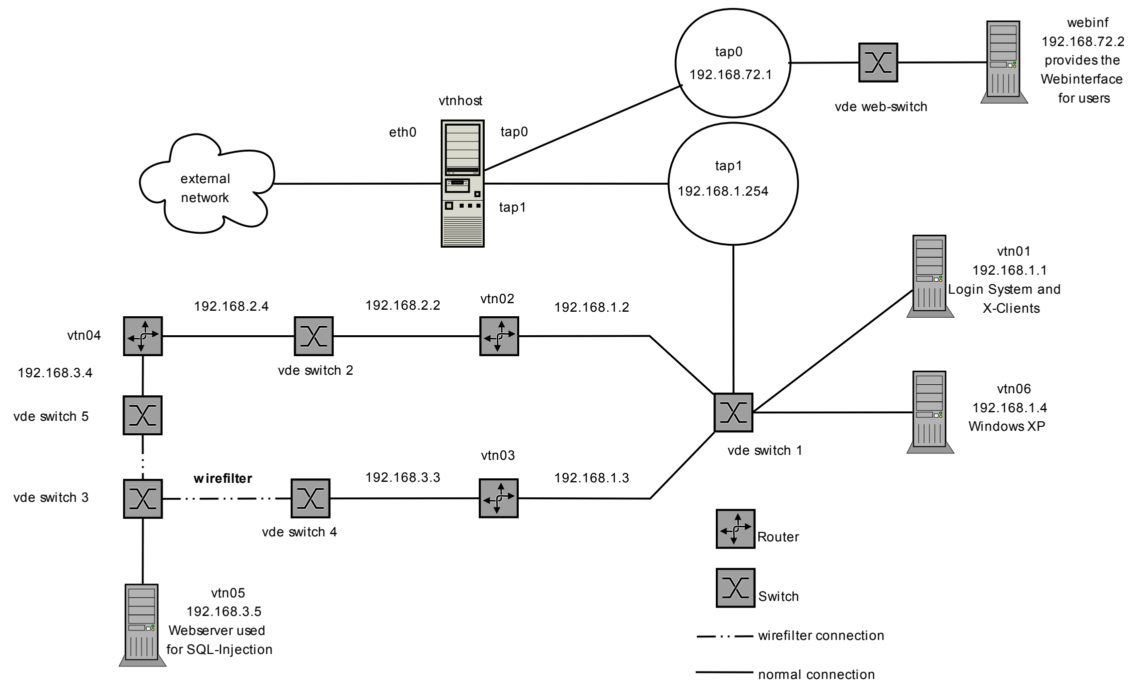


Figure 4.2: topology of the new Virtual Teaching Network

4.5 Integrating the web interface

The web interface of the VTN is the first place for every user to register and schedule usage times. The new VTN is located on a new server making an installation necessary. The web interface controls the operation of the virtual network. It is responsible for creating and assigning a root password to each VTN user, send emails and configures the login machine vtn01 appropriately. These tasks should not be carried out directly on the host system as it is also described in the third point of chapter 9.1 from the Xen user manual:

> Do not allow users to access domain0.

This quote also holds for KVM. The front-end runs in an own VM on a different virtual network to provide a higher degree of isolation thus improving security. Another benefit of this approach is the portability of the web interface in case of a migration or a change in the architecture of the system. Besides that, updates for the host machine and the services of the VM can be incorporated separately again improving availability and security. The approach of running services in a virtual machine and using the host system only to forward packets is described in [CN01].

4.5.1 Installation

The VM for the web interface features MySQL 5, Apache 2 and PHP 5 as it is required in the installation guide [Zam07]. The installation is well documented. The following points describe which issues are encountered and how they are solved.

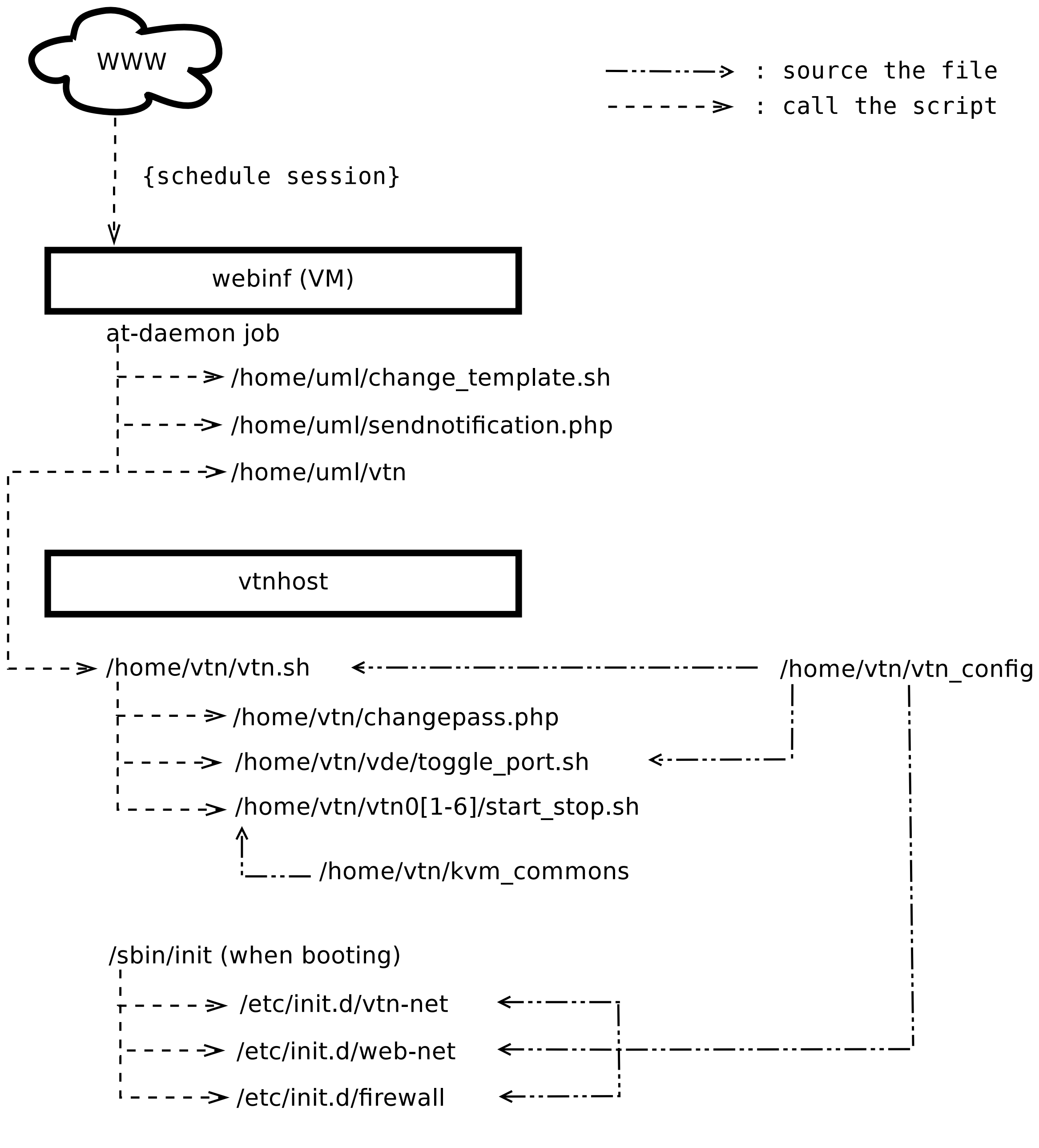


Figure 4.3: dependencies among vtn scripts

/var/utilities/config.php

The configuration file /var/utilities/config.php contains the global parameters for the web interface. The main problem is the e-mail address used to send reservation and reminder e-mails. It refers to the student who wrote the program. The name and password are hard-coded in the configuration file. The password is invalid by now. The username and password are used to authenticate to the SMTP server of the TUHH which uses transport layer security (tls) but also allows sending mails without authentication. Sending via transport layer security (tls) works but not using no authentication which remains an unsolved problem.

In order to deliver mail, the Google mail account virtual.teaching.network@gmail.com is created. Another solution to the problem is to create a separate TUHH e-mail address for the VTN with a password that does not expire.

The change of the configuration file renders the front-end unusable. The problem is hard to locate because the error messages are not displayed. This is a common security measurement but the installation instructions of the web application do not refer to the appropriate sections of the PHP configuration to log these errors to the system. The problem does not become obvious after enabling the logging mechanism. In the end, it turns out that the linebreaks of the files are in DOS format. The configuration file is edited using a Unix file editor which makes the file unreadable for PHP. The front-end is working as expected after converting the files to Unix. All changes to the front-end are summarized in the changelog of the new version.

Changes to the hypervisor interface

In the UML network, the front-end is installed on the host. This allows direct access to the old UML management script /home/uml/vtn. It is used to start and stop the network. In the new network, the front-end has no direct access to this script because it is running in a VM. To enable the management of the VTN, the script is changed to call /home/vtn/vtn.sh on the host system using ssh which is secured by a 1024 bit DSA key.

4.5.2 Integration into the host system

The web interface needs to be available for students to schedule sessions and to manage the VTN at all time. The VM is placed in the subnet 192.168.72.0, see Figure 4.2 and connected to its own switch thus separated from the VTN network. Additional VMs can be added to the same switch if necessary. The script to manage the start and shut down of the VM and subnet infrastructure is located in /etc/init.d/web-net. The script is integrated with the Gentoo Linux init facilities. This way, the web application virtual machine is stopped gracefully when the system is brought down. Even though the virtual machine is started as a system process, it operates with the privileges of the user ‘vtn’ which is another advantage of using the init facilities.

4.5.3 Enhancements to the web application

X client for Microsoft Windows

The UML-based version of the VTN should provide the Xming X client [Har] for Microsoft Windows to present a graphical user interface. Using different network analysis tools becomes much easier. The client was prepared but not integrated into the UML based VTN. While preparing this tool for the new network, it became evident that the client is not reliable. The issue was solved by using the latest version (Xming 6.9.0.24). For each of his sessions, the student receives a custom X client configuration file. It is automatically generated and sent by the system.

Proposed enhancement to the authentication process

The front-end of the VTN requires users to register. The TUHH provides LDAP directory and Kerberos authentication services. One of them or both might be added so members of the TUHH do not need to register. A quick solution to this task is not possible since the program expects these values in the database for scheduling tasks.

A checkbox can be added to the login screen to indicate whether a user should be associated with a TUHH Kerberos / LDAP ID or the signed-up users. Most of the time, the VTN is used by members of the TUHH, so this box is checked by default.

4.6 The firewall

The firewall script is a modified version of the initial UML firewall script. It is located in /etc/init.d/ and integrates into the Gentoo Linux init facilities. The major changes were made due to the additional subnet and tap interface for the web interface which adds more complexity to the firewall. To ease the understanding, the possible paths of packets are summarized in Figure 4.4.

Two IP addresses are assigned to the host. The top address is assigned to the host system and accepts connections on ssh port 22. VNC connections are prepared but need to be enabled.

- Port 22 (ssh)
- Port 5901-5907 (vnc - TUHH only - disabled)

Because of TUHH firewall rules, the VNC ports are limited to the TUHH network. They can not be accessed from the Internet. The second address is used to distribute traffic to the two virtual networks as follows:

- Port 22 is forwarded to 192.168.1.1 port 22 (ssh)
- Port 80 is forwarded to 192.168.72.2 port 80 (http)
- Port 2222 is forwarded to 192.168.72.2 port 22 (ssh - TUHH only)
- Port 3389 is forwarded to 192.168.1.4 port 3389 (rdp - TUHH only - see 5.2.1)

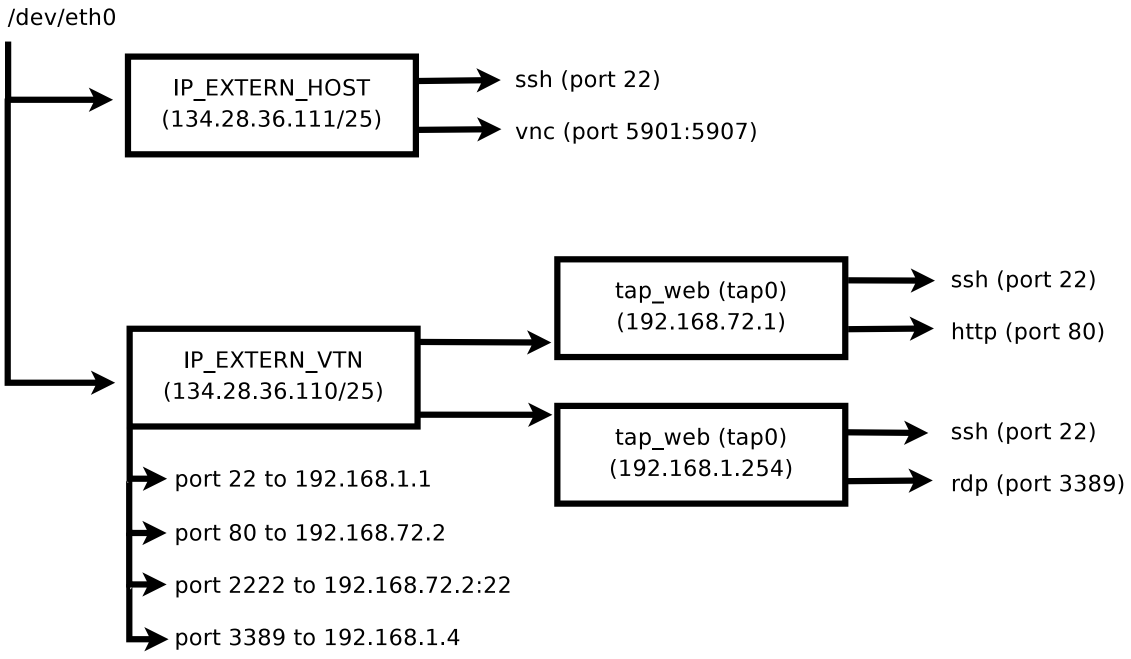


Figure 4.4: possible paths of IP packets

5 Extending the Virtual Teaching Network

This chapter describes extensions to the VTN. The first two sections deal with dynamic changes of network properties. Section three deals with the setup of Microsoft Windows while the last part describes ways to provide access to Microsoft Windows.

5.1 Introducing wirefilter

Wirefilters can be used to change the properties of network links. In the current version of VDE, they can not be attached to a connection between a VM and a vde_switch (see section 3.7.3). This issue is resolved by adding vde_switch4 between vde_switch3 and vtn03 and vde_switch5 between vde_switch3 and vtn04 (shown in Figure 4.2). Switch three connects to switch four and five using a VDE cable which features a wirefilter. The vtn_config file needs to be edited if wirefilter options are subject to change. Note that properties can also be changed while the network is running.

It is important to point out that vde_switch is compatible to UML. It is possible to extend the UML network with wirefilters. The missing support of other operating systems besides Linux can not be solved using UML. A combination of the Linux virtual machines running on UML and other operating systems running on KVM, connected by VDE, is possible.

5.1.1 Dynamic network failures

The target is to extend the network with dynamic network failures. To simulate the failure, a link between switches is shut down while another link on another switch is activated. The result is a change of the routing tables.

Closing a wirefilter connection can be done using the VDE utility `unixterm`. In order to issue commands to the management sockets, `unixterm` needs to accept input from STDIN. This feature is not part of VDE-2.1.6, but a patch exists[1]. The patch is integrated into the Gentoo Linux ebuild for VDE and a bug is filed to make this enhancement available to the Gentoo Linux community[2]. The same `emerge` command described in 4.3.6 reinstalls VDE with the new feature. Commands can now be issued to the management socket of a switch or wirefilter:

```
echo port/allprint | unixterm −i −s /home/vtn/vde/switch1.mgmt
```

Listing 5.1: sending instructions to a switch or wirefilter

The command `port/allprint` prints a summary of all connected ports and their status respectively. For a list of commands please refer to the VDE documentation or connect to a management socket and type ‘help’.

[1]http://wiki.virtualsquare.org/images/b/ba/Unixterm_20070403.patch.bz2
[2]http://bugs.gentoo.org/show_bug.cgi?id=184429

The additional script /home/vtn/vde/toggle_port.sh is necessary in order to automate the connection and disconnection of the wirefilters. It is called when the network starts. In this scenario, vtn03 and vtn04 toggle their connection state. In the initial state, the host vtn03 is disconnected while vtn04 is online. After a certain time, vtn04 is disconnected while the wirefilter between switch 3 and 4 is activated. The time between link failures is set in /home/vtn/vtn_config.

5.2 KVM and Microsoft Windows

KVM supports a broad variety[3] of Microsoft Windows guests ranging from Windows95 to Windows Vista Ultimate. In this scenario, Microsoft Windows XP Professional Service Pack 2 is used. Before running the installation, a new hard disk is created in line one:

```
1  ~$ /opt/kvm/bin/qemu-img create -f qcow winxp\_sp2.fs 10G
2
3  ~$ vdeq \
4          /opt/kvm/bin/qemu \
5                  -hda /home/vtn/winxp/winxp_sp2.fs \
6                  -cdrom /home/vtn/iso/winxp_sp2.iso \
7                  -boot d \
8                  -M pc \
9                  -m 512 \
10                 -k de \
11                 -vnc :7 \
12                 -net nic,macaddr=00:FF:00:00:01:01 \
13                 -net vde,sock=/home/vtn/vde/switch1.ctl \
14                 -pidfile /home/vtn/winxp/vtn.pid \
15                 -no-reboot
```

Listing 5.2: installing Microsoft Windows XP Professional using KVM

The main difference, compared to the command used in 4.3.7, is line six and seven. It instructs QEMU to use an ISO file as CD-ROM device and boot from the device using the *boot d* option. The option is changed to *boot c* (boot from hard disk) after the first reboot of the VM.

The new Windows machine can be used for various tasks. One possibility is to use the VM as an additional attack target. This case is not discussed in this text because too many other issues need to be considered. Another possibility to integrate Windows into the VTN is to offer a graphical interface besides the existing X-Server.

5.2.1 Offering an additional loginserver

The idea to create an additional graphical interface has two reasons. The first one is to evaluate the general behaviour of Windows XP in the virtual environment. The only issue found is the mouse cursor

[3]http://kvm.qumranet.com/kvmwiki/Guest_Support_Status

of the native QEMU VNC display which is inaccurate. There are no problems integrating Windows XP in the VTN. The new machine is named vtn06 and assigned the 192.168.1.4 IP address.

The second reason is to substitute the Windows X-Server client from section 4.3.7 by providing a native client for students using Windows. To connect to the Windows VM, the Remote Desktop Protocol (RDP) is used which is integrated into Windows. The activation of the service is described in the Windows' help. Using RDP solves the issue of the inaccurate mouse cursor when connecting through VNC.

Implementations of RDP clients and the xrdp server project[4] for Linux are available. Similar to the Xming X Client, sessions can be made available using a file which can be sent to students.

The network sniffer wireshark [Com] is installed on the Windows machine. The access to Windows XP is restricted to the local network due to firewall restrictions. Students can gain access using the TUHH VPN client. An alternative solution is to use ssh port forwarding on vtn01 port 22 to vtn06 port 3389.

[4]http://xrdp.sourceforge.net/

6 Evaluation

This section compares the performance between the old, UML based solution and the new, KVM-based implementation. Some properties are presented to characterize the Virtual Teaching Network. UML and KVM utilize different technologies to implement the network interface. KVM emulates a network interface card using QEMU while UML provides a direct link to the socket of the VDE switch. Emulating a network device ensures compatibility with almost all operating systems but is more complex. The direct socket to the switch works only with UML but is expected to offer better performance. The same tools as in [Sto06] are utilized to allow a comparison.

6.1 Round-Trip-Time

The Round-Trip-Time (RTT) is the time needed to send an ICMP ECHO_REQUEST to the receiver, deliver it through the TCP/IP stack and return an ICMP ECHO_RESPONSE to the sender. The `ping` command is used to measure the RTT. It sends 10,000 ICMP ECHO_REQUEST packets and returns the average RTT. The RTT is comparable to the UML network.

Another test case is to validate that the wirefilter used for vde_switch works as expected. It is mentioned in 3.7.3 that packets can be influenced from left to right and the other way around. This is important for the `ping` utility when lost packets are evaluated since a configuration without the direction sets LR and RL to the same loss. This leads to a combined loss around $LR + RL$ since the `ping` utility does not know if the request or response is lost. It is also possible to set the delay to $a + b$ with a as standard delay and b as a random delay. The `ping` command is called using the following flags:

```
ping −f −c 10000 <remotehost>
```

The `ping` command floods the packets which means at least a hundred requests per second. To get a good average, 10000 packets are send.

The measurements refer to a certain measurement point (mp) shown in Figure 6.1. A measurement point is the destination of the `ping` requests. The problem concerning the mps is that it is not possible to accurately measure, for example, mp3. Packets sent to mp3 never leave the second interface of vtn02. Instead, the ECHO_RESPONSE comes from the TCP/IP stack of vtn02. The consequence is that no statement can be made if the guest systems or the switches are responsible for slower performance. No measurement point is assigned to the Windows host but the measurements are recorded in the tables for completeness.

The first `ping` is sent from the host machine to the machine connected to switch one (mp1). The average time is around 0.31 ms which is comparable to a real network. The Windows host is an exception with a response time of 3.89 ms.

The wirefilter is located between switch 5 and switch 3 (see Figure 4.2). The average delay on this connection is 1.4 ms when the wirefilter does not introduce an additional delay. A delay of 2 ms from left to right is configured and the `ping` test is rerun. It is not possible to achieve the exact number which would be 3.4 ms. The measured average delay varies about +0.3 to +0.5 ms. A possible explanation is that a package with a long RTT takes even longer when it is delayed by the wirefilter which leads to a bigger average than expected.

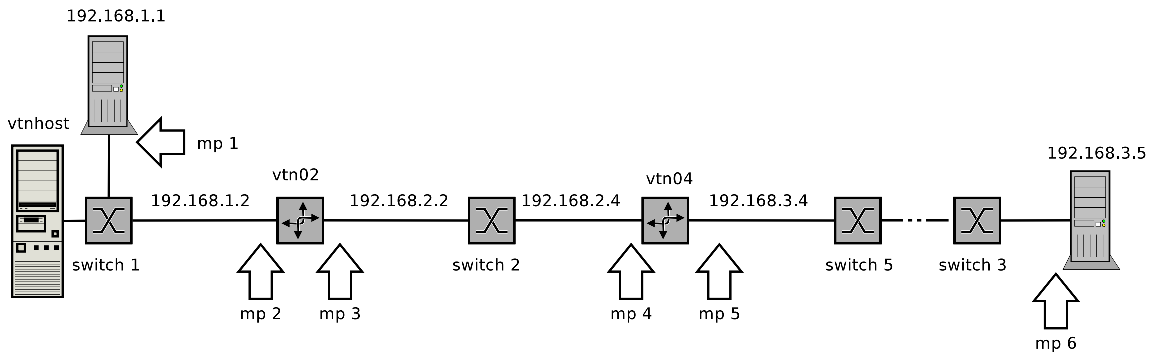


Figure 6.1: measurement points (mps) used in the evaluation

host	remotehost	mp	RTT	bandwidth
vtnhost	192.168.1.1	1	0.31 ms	85 Mbps
vtnhost	192.168.1.4	-	3.89 ms	11 Mbps
192.168.1.1	192.168.1.2	2	0.67 ms	67 Mbps
192.168.1.1	192.168.2.2	3	0.85 ms	62 Mbps
192.168.1.1	192.168.2.4	4	1.35 ms	30 Mbps
192.168.1.1	192.168.3.4	5	1.38 ms	29 Mbps
192.168.1.1	192.168.3.5	6	1.4 ms	27 Mbps

Table 6.1: RTT and bandwidth of the Virtual Teaching Network

6.2 Throughput

Throughput measures the bandwidth available to the network.

```
bing −s 1000 −S 10000 −e 1000 <host> <remotehost>
```

The `bing` command sends 1000 packets with a size of 1000 bytes (-s) or 10000 bytes (-S). The bandwidth from the host to the subnet 192.168.1.0 measures about 85 Mbps while the Windows machine achieves 11 Mbps.

6.3 Investigating Windows performance

Compared to other hosts, the Windows host shows performance figures which are more than ten times higher concerning `ping` and up to eight times lower concerning bandwidth. The first try to increase performance is to add 256MB of memory to the guest which does not affect the measurements.

The next try involves changing the network infrastructure. Instead of connecting the Windows host to the switch, it is directly connected to the tap device of the host. This type of connection should improve performance if the low bandwidth is related to the network infrastructure which is not the case. Measurements stay the same. It is improbable that the different measurements between Windows and Linux hosts are related to the emulated network card since both hosts share the same type of emulated device.

The last action involves returning to the previous network infrastructure and changing the network card. QEMU is able to emulate two different network interface cards. The rtl8029 chipset emulation is the default. The other uses an rtl8139 interface. The rtl8029-based hardware card is used in 10Mbit networks while the rtl8139 features a Fast Ethernet 100Mbit chipset. The rtl8139 network interface card improves the performance near 50 Mbps. The final solution is found by substituting the original Windows XP drivers with the latest version (July 2007). The new driver leads to 113 Mbps. Support for rtl8029 cards is built into Windows XP and updates are not available.

6.3.1 Reinvestigating Linux performance

Because of the results for Windows XP, the Linux results are reinvestigated. The kernel for the guest is recompiled to include the rtl8139 driver while the rtl8029 driver is deactivated. The first run from the host system reveals the full potential of this action. While RTT time is almost 50% lower (0.16 ms), the bandwidth is boosted by more than 250% to 244 Mbps. Table 6.2 summaries the new numbers which are plotted in Figure 6.2.

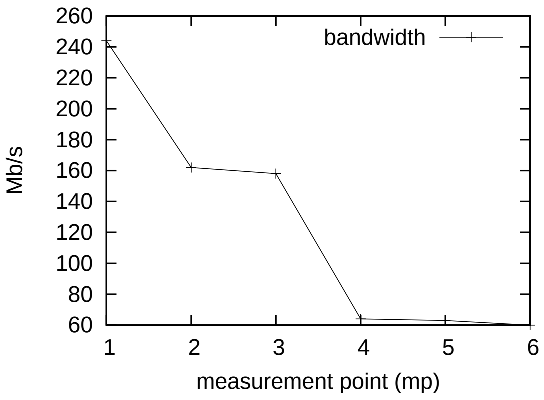
host	remotehost	mp	RTT	bandwidth
vtnhost	192.168.1.1	1	0.16 ms	244 Mbps
vtnhost	192.168.1.4	-	4.12 ms	113 Mbps
192.168.1.1	192.168.1.2	2	0.36 ms	162 Mbps
192.168.1.1	192.168.2.2	3	0.36 ms	158 Mbps
192.168.1.1	192.168.2.4	4	0.63 ms	64 Mbps
192.168.1.1	192.168.3.4	5	0.67 ms	63 Mbps
192.168.1.1	192.168.3.5	6	0.73 ms	60 Mbps

Table 6.2: improved RTT and bandwidth of the Virtual Teaching Network

6.4 Final remarks

The network bandwidth is improved by more than 250% concerning Linux hosts and about ten times for the Windows host. The result is achieved changing the default, rtl8029 based, network interface card to an rtl8139 based emulation which is used with the latest drivers.

The high measurements on Linux based hosts can be explained by Linux driver workarounds due to the fact that the software emulation of the network chipsets is able to send and receive more traffic than the hardware is capable of. Physical limitations of the medium do not allow such a scenario in a real world setup. The driver does not seem to include such a restrictions and allows higher performance which is not the case for Windows XP drivers.



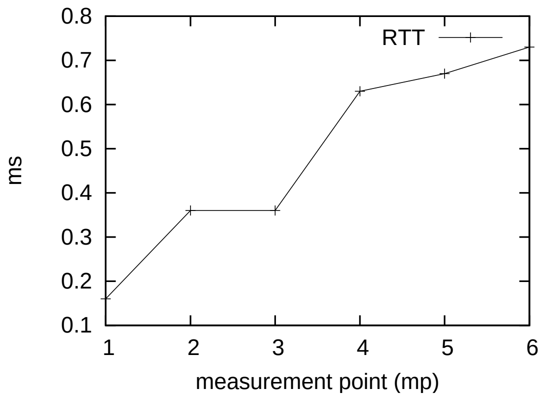


Figure 6.2: graphical representation of Table 6.2

Considering the throughput of the 192.168.1.0 subnet, the VTN is able to transfer more than three times the data of the UML based VTN (77 Mbps). RTT is more than four times smaller (0.16 vs. 0.7). Bandwidth of the VTN is 30% higher and RTT 50% smaller when comparing the last host of the network, 192.168.3.5, to the UML network.

Please note that the hardware for this project is not comparable to the one used in [Sau05] and [Sto06]. An additional measurement is done to find a figure for this difference.

The UML performance is investigated by running vtn01 on UML while the rest of the network is run on top of KVM. RTT is down to 0.07 ms while a bandwidth of 757 Mbps is available. This means the UML runs ten times faster on the new hardware. From a network performance point of view it would be an optimal solution to run the Linux hosts using UML and other guests using KVM. The available bandwidth is not necessary in the VTN.

6.5 Scalability

The physical limitation of the host system, theoretically, allows about 60 Linux software routers to be run at the same time if each machine is equipped with 64 MB of RAM. This leads to $60 * 64 = 3840$ MB RAM leaving around 160 MB to the host if swap space is not considered. If swapping of processes is considered, even more machines can be run. This section investigates if the theoretical performance can be achieved in practice. The setup consists of 60 copies of the vtn02 machine. Each network card is assigned a unique MAC and IP address. The machines are connected to one switch with 65 ports.

The first run starts all machines without interruption. The processor is able to start all virtual machines concurrently. The load during this time is very high resulting in shell commands being accepted with a delay. The machine can be used without problems after all machines are started. A `ping` to some of the machines works in all cases proving the working network connection.

The second run starts one machine every 15 s which is about the time a machine needs for booting. It is possible to work on the host system while the machines are starting. A look at every tenth machine reveals that the machines are always booted before the next VM starts. This result shows us that the performance of the system does not depend on the number of running machines.

Testing the scalability in the way described does not allow a comparison to other systems and is not considered to be a benchmark. A benchmark allowing better results would involve load on the network infrastructure and within the virtual machines. A benchmark called VMmark for virtual environments is presented in [MHS+06]. The benchmark defines tiles which consist of six virtual machines each. The machines simulate a workload of typical mail, Java, standby, web, database and file server. For each tile there needs to be one client connected to a dedicated network which generates load on the network. Results are created by running as many tiles in parallel as possible.

The benchmark is very complex, involves a dedicated setup and consumes a lot of resources. A single tile needs six GB of RAM. The intended audience of the benchmark are system manufacturers. The use of the benchmark is also limited to VMware ESX Server [VI07b].

The software used in VMmark includes commercial and open source products, for example The Software Testing Automation Framework [sta07]. Performance is also benchmarked in [BDF+03] using commercial and open-source products. While commercial products experience regular updates, the latest releases of the open-source projects are older than five years and are not used in this text for that reason. Another approach to measure performance on Xen is taken in [TRK+06]. The author's address SMP scaling, performance tools and Non-Uniform Memory Access (NUMA). The main result with respect to performance is the fact that the available tools do not fit the authors requirements. A new tool is created upon already existing code which restricts measurements to Xen.

Future work in the field of benchmarks in virtual environments might produce less restrictive benchmark suites with respect to licensing, result publication and supported hypervisors.

7 Conclusion

7.1 Related work

Different Universities started similar projects for educational purposes either using UML or Xen. A Xen-based platform using WebLab can be found in [AL06]. It features network topologies to be defined by the student. Scalability seems to be an issue. This is not the case in the XenWorlds project [AD05] since 240 VMs can be run concurrently using a cluster of eight machines. The plan is to increase support for about 1000 VMs by adding RAM to the cluster. A tool called Managing Large Networks (MLN) is explained in [BSJ06]. It is similar to libvirt but supports UML and Xen. It is possible to define networks while the possibilities for topologies are restricted. For larger networks, one might create VMs using a template system with the option to change the contents of the VMs.

7.2 Conclusion

It is shown that migration of a User Mode Linux network to either Xen or KVM is possible. The migration of the virtual machines takes less effort than installing the host machine and setting up the network infrastructure. This is due to the interconnections between the VMs. In general, Xen is a mature environment providing all functions of a modern VMM. KVM is in an early stage of development and offers less features. It is very small and installs fast. The good compatibility to User Mode Linux is shown and the main reason to use KVM for the VTN. The network topology is extended by a Microsoft Windows XP host and dynamic changes of the network links.

The installation of the VMMs proves to be painless with no serious problems. This is mainly due to the fact that unofficial Gentoo Linux ebuild files are available for Xen. Using the well-defined kernel module interface, KVM utilizes a reliable installation mechanism.

Supported networking capabilities differ among the two technologies. The difficulties in meeting the requirements of the VTN with the capabilities of the networking interfaces are various. The main issue is the possibility to find the right configuration between the host network, the hypervisor, the virtual network, and the guest at the same time. Each of these allows different configurations while other influences like the hosts firewall, the firewall of the TUHH network and different script files need to be considered. Especially the script files do not tend to provide a good overview. These issues arise twice because two hypervisors are evaluated.

The virtual network works flawlessly when the right configuration is found and is easily extended in section 5.1. The installation of Microsoft Windows XP works as on any other, real computer. The main problem of virtualization is the difference between supported network technologies if complex setups are necessary. The Virtual Square project in section 3.7.3 approaches this and other challenges while first steps are taken to build a common virtual device driver in [MSO07].

As of the time of writing, there is no other possibility than using bash script files to manage the virtual machines of the Virtual Teaching Network. A solution is to develop a technology-independent tool to manage virtual machines on Linux hosts like the libvirt project in section 3.7.4.

The migration of a virtual machine from UML to Xen and KVM, is basically reduced to build a custom kernel for the guest systems. This task is routine for an experienced Linux user but differs among virtualization technologies. The task is rather easy for KVM while Xen requires more knowledge on this topic. Building the kernel is an important task with respect to functional, performance and security issues.

Installation of the web application is an interesting task. The installation procedure is not a problem and the front-end is accessible within a short time. It becomes clear that not so many people have thought about managing a hypervisor by a virtual machine which is run by the hypervisor. The solution involves a secure ssh connection to issue commands to the host.

Extensions to the network are undertaken in order to prove the functionality of new features. The possibility to change properties of the network links at runtime are used to simulate link failures and increase the delay between hosts. A new host running Microsoft Windows XP is integrated into the network. New scenarios to exploit typical Windows vulnerabilities can be thought of. Windows hosts should not be accessed through VNC clients since RDP is available and offers better performance.

The evaluation of the system is done using basic tools but brought up a dramatic improvement potential of the network. Bandwidth and RTT are improved by more than 250% and 100% respectively. This result leads to an improved scalability of the system. Tests proved that 60 simple router machines can be run concurrently.

Looking at the possible ways to measure the performance and scalability of virtual environments, it becomes obvious that a standardized benchmark suite for different kinds of hypervisors is necessary.

The relevance of virtualization technology in a security context is summarized in section 2.3. It is shown that virtualization can enhance security on one side while opening security holes on another.

To this point, virtualization technology is not a real issue on the x86 architecture. There are indeed topics to work on but the main issues are solved. This statement might sound inappropriate because virtualization is currently a hot topic. In this thesis it is shown that migrating UML to Xen and KVM is possible without changes to the virtual machines. It is shown in [HDL07] that Xen VMs can also be run using KVM although some extensions to KVM are required. The VMs however are not modified. The decision for a specific technology in this thesis is made upon certain requirements that do not directly concern virtualization but connectivity features. The simple lguest [Rus07] hypervisor contains only 5,000 lines of code but proves that virtualization is becoming easier as standard interfaces like paravirt_ops or hardware support evolve. This standardization needs to be done for other parts of computer systems which are affected by virtualization. This includes a unified network interface, standardized benchmarks and common management tools for virtual environments.

A Appendix

A.1 Hardware emulated by QEMU

QEMU emulates the following peripherals (from README of qemu-doc release 0.9):

- i440FX host PCI bridge and PIIX3 PCI to ISA bridge
- Cirrus CLGD 5446 PCI VGA card or dummy VGA card with Bochs VESA extensions
- PS/2 mouse and keyboard
- 2 PCI IDE interfaces with hard disk and CD-ROM support
- Floppy disk
- rtl8139 or rtl8029 PCI network adapters
- Serial ports
- PCI UHCI USB controller and a virtual USB hub.

A.2 Abbreviations

UML User Mode Linux

KVM Kernel Based Virtual Machines

VM Virtual Machine

VMM Virtual Machine Monitor

VTN Virtual Teaching Network

VLAN virtual local area networks

OSPF Open-Shortest-Path-First-Protocol

VDE Virtual Distributed Ethernet

MMU Memory Management Unit

SVM AMD security and virtual machine

TPM Trusted Platform Module

TLS transport layer security

SMP	Symmetric multiprocessing
VT-x	Virtualization technology for Intel x86 processors
VT-i	Virtualization technology for Intel Itanium processors
RDP	Remote Desktop Protocol
RTT	Round-Trip Time

A.3 Xen terminology

dom0	designates the VM which is used to control guest VMs
domU	designates all guest VMs
vif	virtual interface
HVM	Xen Hardware Virtual Machine

Listings

Bibliography

[AA06] Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. In *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 2–13, New York, NY, USA, 2006. ACM Press. http://www.vmware.com/pdf/asplos235_adams.pdf.

[AD05] B. Anderson and T. Daniels. Xen and the Art of Computer Enginnering Education. Technical report, Iowa State University, 2005. http://www.public.iastate.edu/~hawklan/xenWorldsAsee.pdf.

[AJM+06] Darren Abramson, Jeff Jackson, Sridhar Muthrasanallur, Gil Neiger, Greg Regnier, Rajesh Sankaran, Ioannis Schoinas, Rich, Uhlig, Balaji Vembu, and John Wiegert. Intel Virtualization Technology for Directed I/O. *Intel Technology Journal*, pages 179–192, August 2006. http://download.intel.com/technology/itj/2006/v10i3/v10-i3-art02.pdf.

[AL06] M. Alexander and J. A. Lee. A Scalable Xen-and Web-based Networking Course Delivery Platform. In *Proceedings of the 2006 International Conference on Education and Technology (ICET)*, July 2006.

[AMD07] AMD. *AMD64 Architecture Programmer's Manual Volume 2: System Programming Rev. 3.13*, July 2007. http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press. http://www.cl.cam.ac.uk/netos/papers/2003-xensosp.pdf.

[Bel] Fabrice Bellard. QEMU Emulator User Documentation. http://fabrice.bellard.free.fr/qemu/qemu-doc.html.

[Bel05] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *USENIX 2005 Annual Technical Conference*, pages 41–46, 2005. http://www.usenix.org/events/usenix05/tech/freenix/full_papers/bellard/bellard.pdf.

[BSJ06] Kyrre Begnum, John Sechrest, and Steven Jenkins. Getting more from your virtual machine. *Journal of Computing Sciences in Colleges*, 22(2):66–73, 2006. `http://portal.acm.org/citation.cfm?id=1181914#`.

[But] Timothy R. Butler. Bochs - The Open Source IA-32 Emulator Project. `http://bochs.sourceforge.net/`.

[CFH$^+$05] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*, May 2005. `http://www.usenix.org/events/nsdi05/tech/full_papers/clark/clark.pdf`.

[CN01] Peter M. Chen and Brian D. Noble. When virtual is better than real. In *Proceedings of HotOS-VIII: 8th Workshop on Hot Topics in Operating Systems*, pages 133–138. IEEE Computer Society, 2001. `http://www.cs.toronto.edu/~demke/OS_Reading_Grp/pre-s2002/chen_virtual_hotos8.pdf`.

[Com] Gerald Combs. Wireshark: The World's Most Popular Network Protocol Analyzer. `http://www.wireshark.org/`.

[CWdO$^+$06] Jedidiah R. Crandall, Gary Wassermann, Daniela A. S. de Oliveira, Zhendong Su, Shyhtsun Felix Wu, and Frederic T. Chong. Temporal search: detecting hidden malware timebombs with virtual machines. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 25–36, 2006. `http://wwwcsif.cs.ucdavis.edu/~crandall/asplos06temporal.pdf`.

[Dav07] Renzo Davoli. Virtual square, 2007. `http://www.virtualsquare.org/`.

[DB] Máirín Duffy and Daniel P. Berrange. Virtual Machine Manager. `http://virt-manager.et.redhat.com/`.

[DLM$^+$06] Y. Dong, S. Li, A. Mallick, J. Nakajima, K. Tian, X. Xu, F. Yang, and W. Yu. Extending Xen with Intel Virtualization Technology. *Intel Technology Journal*, pages 193–203, August 2006. `http://download.intel.com/technology/itj/2006/v10i3/v10-i3-art03.pdf`.

[Fer06] P. Ferrie. Attacks on Virtual Machine Emulators. Technical report, Symantec Advanced Threat Research, 2006. `http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf`.

[FHH$^+$03] K. Fraser, S. Hand, T. Harris, I. Leslie, and I. Pratt. The Xenoserver computing infrastructure. Technical report, University of Cambridge, January 2003. `http://www.cl.cam.ac.uk/~iap10/general-xeno.pdf`.

[FIa] Gentoo Foundation Inc. Gentoo Linux – About Gentoo. http://www.gentoo.org/main/en/about.xml.

[FIb] Gentoo Foundation Inc. Gentoo Linux x86 Handbook. http://www.gentoo.org/doc/en/handbook/handbook-x86.xml.

[GR05] Tal Garfinkel and Mendel Rosenblum. When virtual is harder than real: Security challenges in virtual machine based computing environments. In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems (HotOS-X)*, May 2005. http://www.stanford.edu/~talg/papers/HOTOS05/virtual-harder-hotos05.pdf.

[Gum83] P. H. Gum. System/370 extended architecture: facilities for virtual machines. Technical report, IBM T.J. Watson Research Center, November 1983. http://www.research.ibm.com/journal/rd/276/ibmrd2706L.pdf.

[Har] Colin Harrison. Xming Notes. http://www.straightrunning.com/XmingNotes/.

[HDL07] Ryan A. Harper, Michael D. Day, and Anthony N. Liguori. Using kvm to run xen guests without xen. In *Proceedings of the 2007 Linux Symposium*, pages 179–188, June 2007. https://ols2006.108.redhat.com/2007/Reprints/harper-Reprint.pdf.

[LW07] Gentoo Linux Wiki. Howto Xen and Gentoo, 2007. http://gentoo-wiki.com/HOWTO_Xen_and_Gentoo.

[Mar] Michael Marineau. Michael Marineau's Overlay. http://overlays.gentoo.org/dev/marineam/.

[MCZ06] Aravind Menon, Alan L. Cox, and Willy Zwaenepoel. Optimizing Network Virtualization in Xen. In *USENIX Annual Technical Conference*, 2006. http://www.usenix.org/events/usenix06/tech/menon/menon.pdf.

[MHS$^+$06] Vikram Makhija, Bruce Herndon, Paula Smith, Lisa Roderick, Eric Zamost, and Jennifer Anderson. VMmark: A Scalable Benchmark for Virtualized Systems. Technical report, VMware Inc., Sept 2006. http://www.vmware.com/pdf/vmmark_intro.pdf.

[Mic07] Microsoft. Microsoft virtual pc 2007, 2007. http://www.microsoft.com/windows/products/winfamily/virtualpc/.

[Mol] Ingo Molnar. KVM Paravirtualization Patches. http://people.redhat.com/mingo/kvm-paravirt-patches/.

[MSO07] Jon Mason, Dwayne Shows, and Dave Olien. Unifying Virtual Drivers. In *Proceedings of the 2007 Linux Symposium*, pages 9–19, June 2007. https://ols2006.108.redhat.com/2007/Reprints/mason-Reprint.pdf.

[MST+05] A. Menon, J.R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. Diagnosing Performance Overheads in the Xen Virtual Machine Environment. In *First ACM/USENIT Conference on Virtual Execution Environments*, 2005. http://www.hpl.hp.com/techreports/2005/HPL-2005-80.pdf.

[Orm07] T. Ormandy. An Empirical Study into the Security Exposure to Hosts of Hostile Virtualized Environments. Technical report, Google Inc., 2007. http://taviso.decsystem.org/virtsec.pdf.

[qua] Quagga Routing Software Suite. http://www.quagga.net/.

[Qum06] Qumranet. KVM: Kernel-based Virtualization Driver. Technical report, Qumranet Inc., 2006. http://www.qumranet.com/wp/kvm_wp.pdf.

[Rot06] J. Rothkowska. Introducing Stealth Malware Taxonomy. Technical report, COSEINC Advanced Malware Labs, November 2006. http://invisiblethings.org/papers/malware-taxonomy.pdf.

[Rus07] Rusty Russell. lguest: Implementing the little linux hypervisor. In *Proceedings of the 2007 Linux Symposium*, pages 173–177, June 2007. https://ols2006.108.redhat.com/2007/Reprints/russell-Reprint.pdf.

[Sau05] H. Sauff. A Linux Based Virtual Teaching Network. Technical report, Hamburg University of Technology, July 2005.

[SJV+05] R. Seiler, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. L. Griffin, and L. van Doorn. Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor. In *21st Annual Computer Security Applications Conference*. IBM T.J. Watson Research Center, 2005. http://www.acsa-admin.org/2005/papers/171.pdf.

[sta07] The Software Testing Automation Framework, 2007. http://staf.sourceforge.net/.

[Ste07] J. Stewart. DNS Cache Poisoning - The Next Generation, 2007. http://www.lurhq.com/dnscache.pdf.

[Sto06] D. Storjohann. Erweiterung eines linuxbasierten virtuellen Übungsnetzwerkes. Technical report, Hamburg University of Technology, October 2006.

[TRK+06] Andrew Theurer, Karl Rister, Orran Krieger, Ryan Harper, and Steve Dobbelstein. Virtual Scalability: Charting the Performance of Linux in a Virtual World. In *2006 Linux Symposium Proceedings*, pages 401–410, July 2006. https://ols2006.108.redhat.com/reprints/theurer-reprint.pdf.

[UNR$^+$05] R. Uhlig, G. Neiger, D. Rodgers, A. Santoni, F. Martins, A. Anderson, S. Bennett, A. Kägi, F. Leung, and L. Smith. Intel Virtualization Technology. *IEEE Computer Society*, pages 48–56, May 2005. `http://www.cs.utah.edu/classes/cs7940-010-rajeev/papers/virt.pdf`.

[Vei] Daniel Veillard. the virtualization API. `http://www.libvirt.org/`.

[VI07a] VMware Inc. Supported guest operating systems, 2007. `http://www.vmware.com/support/ws5/doc/intro_supguest_ws.html`.

[VI07b] VMware Inc. VMware ESX Server, 2007. `http://www.vmware.com/pdf/esx_datasheet.pdf`.

[VI07c] VMware Inc. Whitepaper - Virtualization Overview, 2007. `http://www.vmware.com/pdf/virtualization.pdf`.

[Zam07] A. Zamora. Web Frontend for the Virtual Teaching Network. Technical report, Hamburg University of Technology, March 2007.

[Zov06] D. Zovi. Hardware Virtualization Based Rootkits. In *Black Hat USA*, 2006. `http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Zovi.pdf`.