# A Virtualized Testbed with Physical Outlets for Hands-on Computer Networking Education

Mark Schmidt Florian Heimgaertner Michael Menth
{mark-thomas.schmidt,florian.heimgaertner,menth}@uni-tuebingen.de
University of Tuebingen, Dept. of Computer Science, Tuebingen, Germany

## ABSTRACT

Many computer science curricula include practical courses to undergraduate and graduate students to offer them hands-on networking experience by connecting PCs, switches, and routers in a testbed. Such testbeds are expensive, bulky, energy-intensive, and cause heat problems. Virtualization of PCs and routers on commodity hardware is a solution to those problems. A challenge is to provide physical interfaces for the virtualized components so that students still have the hands-on experience including cabling. In this work, we propose a solution based on inexpensive hardware that can be mounted in a standard 19-inch cabinet. As WiFi adapters, headsets, or additional serial interfaces are needed for advanced experiments, we provide means to connect them as USB devices to virtualized PCs and routers. The system is configured so that students have only access to the virtual machines and their physical interfaces.

## Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education; C.2.3 [**Computer-Communication Networks**]: Network Operations

## Keywords

Laboratories; Networking; Virtual Machines; Computer networking education

## 1. INTRODUCTION

While practical networking courses are not mandatory for most computer science and electrical engineering degrees, they are very popular among students. A major reason for that is the insight that applied networking knowledge may be helpful for their future career but also the fact that interconnecting devices with cables and switches is fun for most students. Such hands-on networking courses were offered at the University of Tuebingen since 2004 based on the concept of Liebeherr and Zarki [15]. The testbeds consisted of 6 PC and 2 routers that can be connected via two unmanaged desktop switches, as well as via a wireless access point and WiFi adapters attached to the PCs. The PCs were connected to a central server through an additional switch. This setup is depicted in Figure 1.

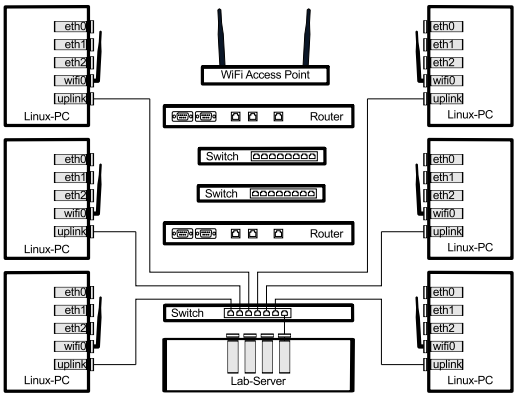



**Figure 1: Initial situation. Testbed with physical nodes.**

In 2012, a fundamental renewal was necessary, because the testbed hardware was both outdated and degraded. The following conditions had to be taken into account. The networking lab was so successful that the three testbeds, operated in time-sharing mode, were no longer sufficient for the large number of interested students, but our lab had not enough space for more physical testbeds. As no sponsors for hardware could be found and the testbed hardware is expected to be renewed in the future again, the hardware and software expenses should be low. Resources for scientific computing were needed at the same time.

At that time, new advances in hardware support for virtualization of the x86 architecture stipulated academic experiments in our group. As a result, we designed and implemented a testbed setup where the 6 PCs and 2 routers were implemented as virtual machines (VMs) on a commodity server. The Ethernet interfaces of the virtual nodes are connected through the Ethernet ports of the testbed host via a managed switch to a patch panel using Virtual Local Area Network (VLAN) technology. This way, the interfaces of the virtual nodes are physically accessible. Likewise, USB devices can be plugged into a USB hub and mapped to specific

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-1. The authors alone are responsible for the content of the paper.


*SIGITE/RIIT'14,* October 16–18, 2014, Atlanta, GA, USA.

ACM 978-1-4503-2686-5/14/10 ...$15.00.
http://dx.doi.org/10.1145/2656450.2656451.

virtual machines. In such a testbed, the hands-on experience for students is retained as they can interconnect the physical interfaces of the virtual nodes with real switches and cables or attach USB devices. This is not the case in fully virtualized or emulated environments.

The testbed was shown as a demonstration [23] at SIGCOMM'14. This paper presents the setup in detail.

The paper is structured as follows. Section 2 reviews various approaches for networking lab testbeds. Section 3 describes the testbed architecture and explains solutions to special challenges. Section 4 summarizes this work and draws conclusions.

# 2. RELATED WORK

As practical networking courses are quite common in computer science and IT education, various designs for labs and testbeds have been published. In this section we provide an overview of different implementation approaches.

In [15] Liebeherr and Zarki describe a hands-on Internet course. The book contains both the syllabus and instructions for setting up a testbed. The testbed consists of 4 physical Linux PCs and 4 Cisco routers. In [3] a testbed design based on 3 PCs, one laptop computer, and two multiprotocol routers is described. In addition, central networking equipment is proposed for management purposes and inter-testbed connectivity.

Emulab [26] is a platform for networking testbeds that allows to connect physical nodes or VMs over virtual networks. Topologies and link characteristics are modelled using VLANs and transparent traffic shaping nodes. In addition, simulated nodes can be integrated into experiments. The authors of [24] describe a similar implementation. To enable remote learning, the physical nodes are connected by configuring VLANs instead of plugging cables.

Other hands-on labs are entirely built on VMs. In [2] a setup is described where 120 PCs and routers run as Xen VMs on a single host and are connected by virtual switches. dVirt [17] is a virtualized BGP router testbed consisting of Xen VMs connected by Open vSwitch. V-Lab [27] is a remote access lab with VMs on a XenServer cluster connected using VLANs.

In [1] a reconfigurable network lab based on VMware is described. Each physical interface can be assigned to at most one VM to provide physical access to Ethernet interfaces of VMs. This approach requires at least as many physical Ethernet interfaces as virtual Ethernet interfaces. As the number of pluggable network interface cards (NICs) per host is limited, this is a severe limitation on the number of virtual interfaces supported by the host. Therefore, we use a different approach which can support more virtual interfaces than the number of Ethernet ports on a host.

# 3. TESTBED ARCHITECTURE

This section starts with an overview of the testbed and a hardware description. We explain the virtualization platform and the software infrastructure of the networking lab. Finally, we describe our realization of physical access to interfaces of VMs, and present solutions to problems encountered during the actual implementation of our concept.

## 3.1 Overview

The new testbed provides the same features as the old testbed given in Figure 1. It consists of a "testbed host" and a cabinet with networking hardware, which are illustrated in Figure 2. The "testbed host" hosts 6 virtualized Linux PCs and 2 routers. The networking hardware is placed in a 6U wall mount cabinet (covers and front door are removed in the figure). At the bottom a managed switch is mounted facing backwards. Above the switch, two front panels carry DB9 and RJ45 outlets for the virtual routers. Unmanaged desktop switches for student exercises and other networking devices are placed on a rack-mount shelf. The topmost rack unit is covered by a labeled patch panel, serving as network outlets for the virtualized PCs. A USB hub is attached to the left mounting post. The WiFi adapters for the PCs are provided as USB devices that can be passed through to the VMs. In the same way physical headsets for voice-over-IP (VoIP) exercises can also be connected. On top of the cabinet, a wireless access point running the OpenWrt embedded Linux distribution is available for experiments involving IEEE 802.11 technology.

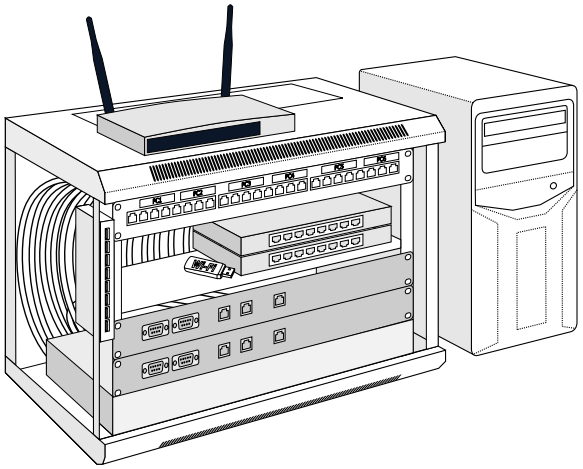


**Figure 2: Networking cabinet and testbed host running the virtual nodes.**

An additional "lab server" provides common services (see Section 3.4.2) to all the testbeds.

## 3.2 Hardware Description

We use commodity server hardware based on the Intel Xeon platform (Xeon E3 Sandy Brigde) for the testbed host. The machine is equipped with 32 GB RAM, RAID-5 disk storage, and an Intel I350-T4 Ethernet adapter. Multiple USB WiFi adapters, two USB/Serial converters, and two USB headsets can be connected through a powered 8 port USB hub.

To provide access to the Ethernet interfaces of the VMs, we use a managed HP Procurve 48 port Ethernet switch in combination with a 24 port patch panel. The students use two unmanaged desktop switches and large quantity of Cat5e twisted pair patch cables to create and modify network topologies in the course of their exercises.

The lab server relies on the same hardware platform as the testbed hosts, but it is equipped with additional Ethernet interfaces so that every testbed can be connected.

## 3.3 Virtualization Platform

We describe the hardware and software support for the virtualization concept of the testbed host.

### 3.3.1 Hardware Support for Virtualization

The hardware of the testbed host has to provide support for multiple hardware virtualization features.

Firstly, the CPU has to provide extensions that enable the x86 architecture to be virtualized with hardware support, which is not possible by default. This is necessary as pure software virtualization is very CPU intensive and lacks performance. Those extensions are marketed as VT-x [25] by Intel. Different implementations are available from other CPU manufacturers.

Secondly, an Input/Output Memory Management Unit (IOMMU) is required as we want to pass through physically available hardware, e.g., Ethernet devices, to the VMs. An IOMMU provides features such as translation from virtual device addresses to physical device address, which is needed for remapping of interrupts and DMA. In the case of Intel hardware this feature is summarized as VT-d [7].

As a testbed host does not provide enough PCI slots to plug in four dedicated Ethernet devices per VM, we have to be able to realize more than one virtual Ethernet interface per physically available NIC. Two additional technologies are used to realize and subsequently map these virtual interfaces to VMs. Virtual Machine Device Queues (VMDQ) [8] enable multiple queues per NIC. They are connected by an internal bridge, and implement the packet forwarding to the VMs in hardware. PCI-SIG Single Root I/O Virtualization (SR-IOV) [18] provides an extension of the PCI Express standard that allows multiple so-called virtual functions (VF) on a single physical function (PF). A PF is a full-featured PCI device whereas VFs are lightweight PCI devices that are managed by a PF. Each VF can be individually passed through to a VM. Together with VMDQ, it is possible to provide multiple virtual Ethernet interfaces, each with its own queue, on a single physical device, which is called VT-c [6] for Intel hardware.

### 3.3.2 Software Support for Virtualization

Software support is required to make the virtualization mechanisms available. The hypervisor used to run the VMs needs support for the extensions described above. Especially, it should be possible to pass through both Ethernet and USB devices to the VMs. Additionally, we need to limit the resources used by each VM so that misconfiguration of a VM by students does not affect the host system.

We use the Kernel-based Virtual Machine (KVM) [10] of Linux as hypervisor on the testbed host for the VMs. The basic idea of KVM is to implement the hypervisor as part of the Linux kernel on the host machine instead of using a dedicated software like Xen. That means, the host kernel has direct access to the hardware and is responsible for regulating VM access to the hardware. However, KVM cannot be used directly as it is realized as a kernel module and does not provide an interface for user interaction. QEMU [19] is the user space software to run the VMs. It is a multipurpose virtualization tool which can emulate various types of hardware such as CPU, hard disk, or network adapters. Additionally, it can use KVM as a backend to benefit from hardware virtualization features of modern x86 CPUs and chipsets. In the presence of hardware support, entire hardware devices like PCI or USB devices can be passed through to a VM.

For improved maintainability a virtualization framework should be used to create, manage and run the VMs, instead of starting QEMU directly. We use the libvirt [21] framework which provides among other tools the command line frontend `virsh` and the graphical tool `virt-manager`. Hardware and resources assigned to a VM are configured using XML files which are convenient to understand and modify for both humans and computers. Furthermore, libvirt allows to add and remove hardware like USB devices at VM runtime. Thereby we can connect WiFi adapters, headsets, or USB/Serial adapters to the VMs.

The VMs can either be accessed by a serial text terminal or graphically using "virt-viewer" which can use VNC or spice [22] to provide the screen of the VM. We prefer spice because of better performance and additional features such as clipboard sharing between testbed host and VM as well as automatic adjustment of screen resolution for the VMs depending on the window size of the viewer.

As we want a Cisco-IOS-like experience for the virtual routers, we do not provide graphical access to them, but only a command line interface. This interface is realized by the `vtysh` shell of the Quagga [20] routing software suite.

## 3.4 Networking Lab Software Infrastructure

We describe the software support for the testbed host and the VMs and the services provided by the lab server.

### 3.4.1 Software Support for Testbed Host and VMs

All machines, physical and virtual, are powered by the Ubuntu [4] Linux operating system. The testbed hosts are based on a minimal installation and contain only the software required to run and access the VMs, that means in particular QEMU and libvirt with their dependencies, but also a simple user interface (see Figure 3). The interface provides a menu to start, stop, attach USB devices and access the VMs and a web browser. The VMs run on a default Lubuntu installation, the LXDE (a lightweight desktop environment) flavor of Ubuntu, and are equipped with additional software such as wireshark to monitor traffic or services, which is needed for practical exercises. The virtual routers are also based on a minimal installation and additionally provide the Quagga router suite.

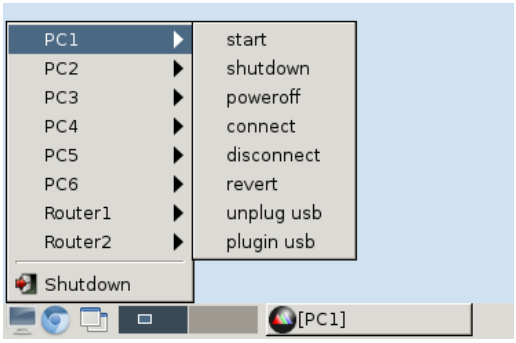



**Figure 3: User interface of the testbed host.**

To simplify the management of the network interfaces, we decided to rename the network interfaces by the use of udev [11] in a more structured way instead of the new default `biosdevname` [16] based naming scheme. Figure 5(b) shows that the interfaces within the VMs are named according to the traditional scheme, e.g. `eth{0,1,2}`, which provides a better mapping to the physical network outlets. Before a

VF `eth`$i$ of VM $j$ is passed through to that VM, it is named `vm`$j$`-eth`$i$ on the host. The VFs `eth`$i$ of all VMs are hosted by the same PF which is named `vm-eth0`.

### 3.4.2 *Services Provided by the Lab Server*

The lab server provides several services for the testbed hosts as well as for the VMs, which is shown in Figure 4. In addition to infrastructure services, such as DHCP or DNS and gateway, we also use services for a central account and configuration management.

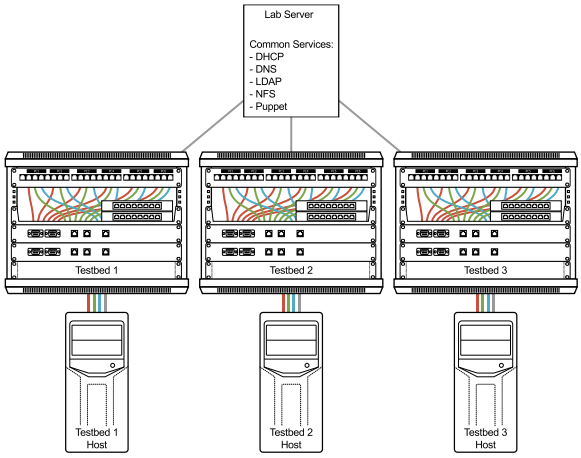



**Figure 4: Lab setup with 3 testbeds.**

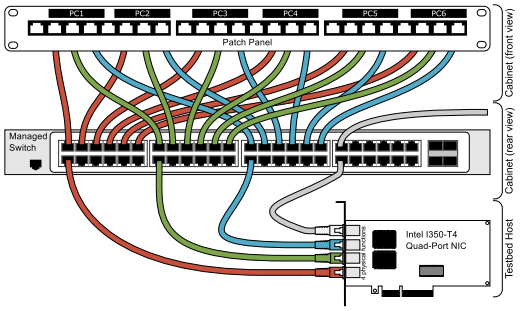
To allow flexible and efficient management of both VMs and testbed hosts, large parts of the system configuration are managed centrally on the lab server. For this purpose we use the puppet [9] configuration management software. Using a declarative domain specific language, the desired configuration of a node can be specified in so-called manifests. This way we manage software packages, services as well as configuration files of testbed hosts and VMs. The puppet "master" is running on the lab server, providing the configuration information. On the testbed hosts and VMs, puppet "agents" are fetching and applying the configurations.

The user accounts for the students are managed using an Lightweight Directory Access Protocol (LDAP) service and their home directories are provided using the Network File System (NFS). That way, the students can log in at any available testbed and have access to their stored data. This is especially useful in the case of testbed maintenance or in the case of hardware problems. In addition to the home directories, we use NFS to provide initial configuration files and scripts for the students to use in the exercises.

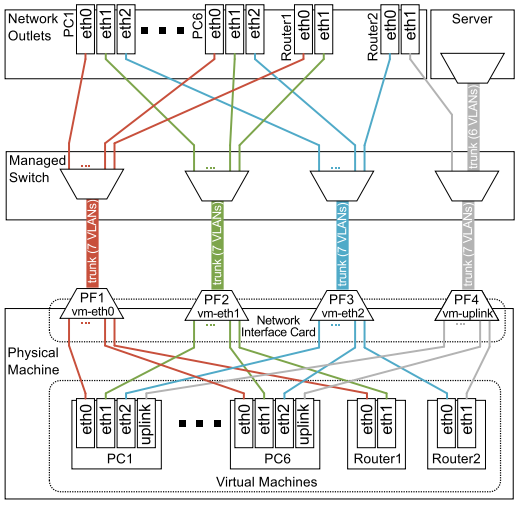
Some commands in the exercises, such as starting or stopping a service, require special permissions – they have to be run as the *root* user. This can be achieved using the `sudo` tool. However, for security reasons, students should not be allowed to execute arbitrary commands as root, but only a restricted set of commands. To realize this, `sudo` allows to define these permissions for each command for both users and groups. Therefore, students are members of a special student group which has limited root permissions defined in the sudo configuration. As we do not want to manage this configuration by hand on each VM, we also use LDAP to provide these rules for all VMs which requires `sudo-ldap`, a special version of `sudo`, on the VMs.

## 3.5 Providing Physical Access to Virtual Ethernet Interfaces

One of our main goals is to enable students to physically connect VMs with real cables and switches. To that end, physical access to Ethernet interfaces of the VMs is required.




(a) Physical setup; cables for the virtual routers are omitted.




(b) Logical overview.

**Figure 5: Providing physical access to Ethernet interfaces of VMs.**

Each of the 6 virtual PCs has 4 Ethernet interfaces and each of the 2 virtual routers has 2 Ethernet interfaces. Figure 5(a) illustrates that the Ethernet interfaces of the VMs are multiplexed over an Intel I350 quad-port NIC and demultiplexed by a managed switch. From there, they are connected to a patch panel where physical access to each of them is provided.

Each of the 4 ports on the NIC is implemented as a separate PF providing 7 VFs that represent the Ethernet interfaces of the VMs. As explained in Section 3.3.1, they are logically separate PCI devices that are individually passed through to VMs.

A major challenge is the multiplexing and demultiplexing. Figure 5(b) shows that the VFs `eth{0,1,2}` and `uplink` from each VM are multiplexed over one PF (`vm-...`) of the testbed host. We use VLAN technology for that multiplexing so that the managed switch can easily demultiplex the individual Ethernet interfaces. VLAN is defined by IEEE 802.1Q [13] and allows multiplexing and demultiplexing of several virtual LANs (VLANs) over a common

physical link which is then called "trunk" link. To enable multiplexing and demultiplexing, a "tag" is inserted into the header of Ethernet frames indicating the VLAN. In our solution, we use a dedicated VLAN with a unique ID per VF, representing a VM interface. The PFs act as an Ethernet bridge, forwarding data for the VFs as tagged VLANs. The driver on the testbed host configures the PF to transparently add/remove the tags in the Ethernet frames during the transition from/to the VM. The managed VLAN-capable switch also adds/removes the tags while multiplexing/demultiplexing the VLANs from/to the patch panel. As a result, VLAN tags cannot be observed in Ethernet frames neither in VMs nor on the patch panel.

## 3.6 Attaching USB Devices to VMs

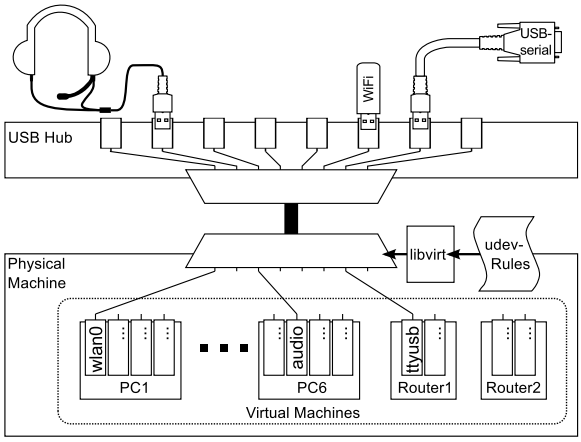



**Figure 6: Connecting USB devices to VMs.**

As shown in Figure 6, selected USB devices can be plugged into a USB hub that is connected to the testbed host and then be passed through to a VM. In the following, we explain how we configure this passthrough.

USB devices have a unique ID and get an address assigned as soon as they are connected to the bus. We implement a fixed assignment of USB devices to VMs inside a testbed. For selected USB devices the assignment of the IDs to VMs needs to be defined in appropriate udev rules. Flexible assignment of special USB devices to VMs is basically possible but not implemented as not needed.

When a USB device is plugged in, the testbed host gets a udev event containing the address and ID of that USB device. The testbed host matches the ID against a udev rule set that triggers an action. We defined as an action for the IDs of the special USB devices the execution of a shell script that generates a configuration file for libvirt. The user can then trigger the passthrough of the USB device to the VM over the GUI illustrated in Figure 3. The trigger calls libvirt which loads the configuration file and possibly interacts with QEMU to pass the address of the respective USB device through to the appropriate VM. This design allows the special USB devices to be plugged in before the VMs are started.

## 3.7 Problems and Solutions

During development and testing of the testbed we encountered several problems. We describe some of them and our solutions.

### 3.7.1 Bridged Virtual Functions

The Intel I350 network adapter provides a feature called *PF Loopback*. This technique is intended to improve network performance by bypassing an external switch in cases where both source and destination are VFs of the same PF. While PF Loopback is a desirable feature for common datacenter applications, it leads to unwanted side effects in the testbed. Under certain configurations, network connectivity between VMs remains available even though the corresponding cables are unplugged.

Version 4.3.0 and below of the Intel NIC driver [5] did not provide an option to disable this behavior without modifying the source code. Since version 5.0.5 the behavior is configurable and can be changed using the `bridge` tool[1] of the iproute [12] package. It can be used to change the bridge mode of the PF to *Virtual Ethernet Port Aggregator* (VEPA) [14] mode, which enforces the desired behavior.

### 3.7.2 Shared Network Bandwitdh

The bandwidth of a PF is shared by its VFs. This may lead to problems with point-to-point performance measurement. A workaround is to use at most one VF per PF for such experiments. Another solution is to connect the PF to Gbit ports of the switch but to connect the VLANs on patch panels to 100 Mbit ports of the switch. This reduces the p2p bandwidth to at most 100 Mbit/s.

### 3.7.3 Identification of USB Devices

Some vendors do not respect the purpose of unique serial number fields and use the same ID for all devices in an entire product batch. Therefore, it is not always possible to distinguish plugged USB devices only by their IDs.

To still differentiate them, the specific driver, e.g., an Ethernet driver, for the device has to be loaded on the host. Thereby, USB WiFi adapters can be distinguished by their MAC addresses. However, to use the USB passthrough mechanism and assign the USB device to a specific VM, the driver has to be unloaded on the host again. This can be achieved by special udev rules and scripts.

### 3.7.4 Virtual Machine Images

Local file system changes made by students should be persistent during exercises and in particular be kept on power cycles. After the completion of an exercise or in case of a major misconfiguration, it should be possible to reset the file system to a default configuration. We describe how we achieve these two goals.

The VM image is stored on the hard disk on a separate volume. It contains the initial file system. Changes to the file system are stored as increments in a special file on another volume. By resetting that file, the original file system is restored.

We implement this method by using LVM2 to create logical volumes (LVs) of the hard disk which are virtual disk partitions. Each VM image is stored in a separate LV; they differentiate only by minor configuration differences. The incremental storage of file system changes is performed by QEMU and commonly known as overlay. The base volume is the so-called backing file and the file with the incremental changes is called image and uses the qcow2 format.

[1] tested with version 3.12. The bridge tool was introduced in version 3.5.0, 2012

### 3.7.5 Serial Network Interface

The serial link between the virtual routers is realized by a PPP connection over a null modem RS-232 serial cable. By default PPP provides flow control for the communication which leads to the effect that packets are queued if the physical link is broken and all packets are sent if the link becomes available again. This effect can be observed with the `ping` command. Packets are not dropped as expected when unplugging the cable. When the cable is plugged in again, large round-trip times can be observed as they include the downtime caused by unplugging the cable. To enforce the desired behavior, it is necessary to explicitly disable flow control in PPP.

# 4. CONCLUSION

We presented a testbed setup with all PCs and routers virtualized on a single commodity server. The testbed distinguishes from others by the fact that Ethernet and USB interfaces to the virtual machines are accessible on a patch panel and a USB hub, respectively. The implementation combines the advantages of a purely virtualized testbed and an entirely physical testbed. On the one hand, our approach is cost-effective, saves energy and space and causes only little heat, and the testbed host can be reused for scientific computing if not needed for exercises. On the other hand, it retains hands-on experience for students in the sense that they can connect nodes using real switches and cables. The suggested architecture is easily extensible as new devices can be attached to virtual nodes via the USB hub, which allows the integration of new experiments in networking courses.

We successfully operate 6 of these testbeds since January 2013 and gained experience from 3 rounds of successful networking courses since then. The interoperation of the early virtualization-capable versions of the operating system, drivers, as well as other software and hardware was initially rather challenging. However, the software has sufficiently evolved in the meantime so that the operation of the testbed is stable. This experience shows that virtualization of low-cost devices has matured to a degree that they can be plugged together even for non-standard deployment.

# 5. ACKNOWLEDGEMENTS

The authors thank Wolfgang Braun, Michael Hoefling, Andreas Stockmayer and Sebastian Veith for helping assembling the testbeds, Jakob Herrmann, Cynthia Mills and Andreas Stockmayer for support in executing the networking courses, and the Institut fuer Astronomie und Astrophysik Tuebingen (IAAT) Workshop for manufacturing the front panels of the virtual routers.

# 6. REFERENCES

[1] S. Abbott-McCune, A. J. Newtson, and B. S. Goda. Developing a Reconfigurable Network Lab. In *ACM SIGITE*, 2008.

[2] C. Avin, M. Borokhovich, and A. Goldfeld. Mastering (Virtual) Networks - A Case Study of Virtualizing Internet Lab. In *International Conference on Computer Supported Education (CSEDU)*, 2009.

[3] C. E. Caicedo and W. Cerroni. Design of a Computer Networking Laboratory for Efficient Manageability and Effective Teaching. In *IEEE Conference on Frontiers in Education*, 2009.

[4] Canonical Ltd. Ubuntu 14.04 LTS (Trusty Tahr). http://releases.ubuntu.com/14.04/, 2014.

[5] Intel Corp. igb Linux Base Driver for Intel Ethernet Network Connection. http://sourceforge.net/projects/e1000/files/igb_stable.

[6] Intel Corp. Intel Virtualization Technology for Connectivity (VT-c), 2012.

[7] Intel Corp. Intel Virtualization Technology for Directed I/O (VT-d) Architecture Specification, 2012.

[8] Intel LAN Access Division. Intel VMDq Technology. Whitepaper, Intel Corp, 2008.

[9] L. Kanies. Puppet: Next-Generation Configuration Management. *The USENIX Magazine*, 31(1), 2006.

[10] A. Kivity et al. kvm: the Linux virtual machine monitor. In *Linux Symposium*, 2007.

[11] G. Kroah-Hartman. udev – A Userspace Implementation of devfs. In *Linux Symposium*, 2003.

[12] A. Kuznetsov and S. Hemminger. iproute2: Utilities for Controlling TCP/IP Networking and Traffic, 2012.

[13] LAN/MAN Standards Committee of the IEEE Computer Society. *IEEE 802.1Q: Virtual Bridged Local Area Networks*, 2003.

[14] LAN/MAN Standards Committee of the IEEE Computer Society. *IEEE 802.1Qbg: Edge Virtual Bridging*, 2012.

[15] J. Liebeherr and M. E. Zarki. *Mastering networks – an internet lab manual*. Pearson Education, 2003.

[16] Narendra K. Consistent Network Device Naming in Linux. Whitepaper, Dell Linux Engineering, 2012.

[17] I. Oprescu, M. Meulle, and P. Owezarski. dVirt: A Virtualized Infrastructure for Experimenting BGP Routing. In *IEEE Conference on Local Computer Networks (LCN)*, 2011.

[18] PCI SIG. Single Root I/O Virtualization and Sharing Specification 1.1, 2010.

[19] QEMU team. QEMU 2. http://wiki.qemu.org/ChangeLog/2.0, 2014.

[20] Quagga team. Quagga Routing Suite. http://www.nongnu.org/quagga/.

[21] Red Hat. libvirt: The Virtualization API. http://libvirt.org, 2012.

[22] Red Hat. SPICE. http://www.spice-space.org/, 2012.

[23] M. Schmidt, F. Heimgaertner, and M. Menth. Demo: A Virtualized Lab Testbed with Physical Network Outlets for Hands-on Computer Networking Education. In *ACM SIGCOMM*, 2014.

[24] S. C. Sivakumar, W. Robertson, M. M. Artimy, and N. Aslam. A Web-Based Remote Interactive Laboratory for Internetworking Education. *IEEE Transactions on Education*, 48(4):586–598, 2005.

[25] R. Uhlig et al. Intel Virtualization Technology. *IEEE Computer*, 38(5):48–56, 2005.

[26] B. White et al. An Integrated Experimental Environment for Distributed Systems and Networks. In *Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[27] L. Xu, D. Huang, and W.-T. Tsai. V-Lab: A Cloud-Based Virtual Laboratory Platform for Hands-On Networking Courses. In *Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, 2012.