A Semi-Virtualized Testbed Cluster with a Centralized Server for Networking Education

Mark Schmidt, Andreas Stockmayer, Florian Heimgaertner, and Michael Menth
Chair of Communication Networks, University of Tuebingen, Tuebingen, Germany
Email: {mark-thomas.schmidt,andreas.stockmayer,florian.heimgaertner,menth}@uni-tuebingen.de

***Abstract*—Hands-on computer networking labs are essential in many computer science curricula. They are conducted either on physical testbeds consisting of PCs, routers, switches, cables, etc., or on fully virtualized testbeds. The latter consist of only virtual machines (VM) that can be interconnected via software configuration. Fully virtualized testbeds require less resources (hardware, space, energy) than physical testbeds but students miss important hands-on experience with networking equipment. In this work, we present a semi-virtualized testbed: students are given physical access to networking interfaces of VMs via patch panels so that they can interconnect them through cables. Similarly to virtualized testbeds, the semi-virtualized testbed requires only little hardware and maintenance effort while preserving the hands-on experience of physical testbeds. We present a Python-based orchestration platform for several virtual student workspaces on a single physical server. Each virtual student workspace contains several VMs acting as clients, servers, and routers that can be configured by students. It is made available to a physical workspace on a 19-inch cabinet consisting of a thin client and patch panels allowing students to physically interconnect their VMs with cables.**

I. Introduction

Practical courses are an important part of networking education. Students learn to configure devices and interconnect them with cables and switches. Such courses are traditionally based on physical testbeds consisting of PCs, routers, and switches. While offering real hands-on experience, this approach has the drawback that it requires lots of space, energy, and maintenance effort. With progress in virtualization technology, fully virtual testbeds emerged and were used in some networking courses. In these testbeds, PCs, routers, and switches run as virtual machines (VM) on a server, thereby avoiding some shortcomings of physical testbeds. However, fully virtual testbeds do not provide hands-on experience which is an important learning target and fun factor of practical networking courses. We believe that hands-on experience is an important part of networking labs. We obsevrve students having problems realizing limitations and problems with physical cabling regarding available ports, cables etc. as well as problems organizing their cabeling work. Therefore it is important to give them the opportunity to use real cables as part of the learning experience. In this paper we focus on lab systems that allow hands-on experience. We distinguish between three different types of architectures depicted in Figure 1. Traditionally, a student workspace consists of a physical testbed that allows to interconnect several computers and routers. An entire lab system is composed of several such testbeds and an additional server to provide infrastructure services (IS). As an alternative to a physical testbed, a semi-virtualized testbed with a dedicated server per student workspace may be used. Computers and routers run as VMs on the dedicated server and a patch panel allows to interconnect their interfaces with cables. The testbed consists of a physical and a virtual workspace (PW, VW) and the VW runs on the virtualization server. Due to the PW, the testbed is only semi-virtualized. This preserves all benefits of physical testbeds. In this work, we present a semi-virtualized testbed cluster with a central server for multiple student workspaces. The central server hosts the IS and the VWs for multiple testbeds that are mapped to different PWs.

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.

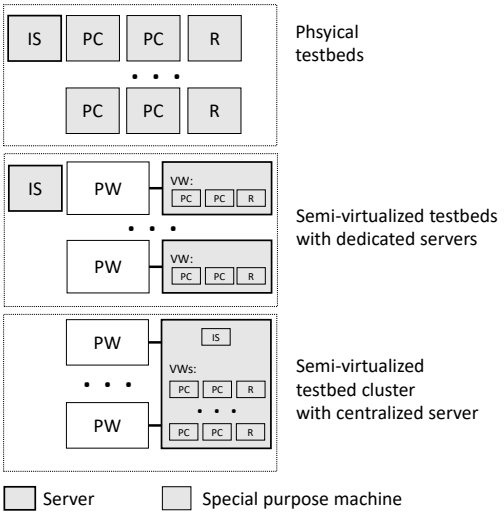


Fig. 1: Three types of lab systems for networking education with hands-on experience.

At the University of Tuebingen we offer practical networking courses since 2004. Initially, our curriculum was based on the concept of Liebeherr and Zarki [1] using physical testbeds. In 2012, we reworked the lab content and substituted the physical testbed by semi-virtualized testbeds with dedicated servers [2], [3]. Recently, we further elaborated that approach

towards a testbed cluster with a centralized server.

Virtual testbeds can be easily managed with the help of an orchestration framework. Such frameworks are widely used and a common technology today. In contrast, orchestrators for semi-virtualized testbeds which fit the requirements for practical networking courses are hardly found. A reason for that is that orchestrators for semi-virtualized testbeds are not trivial to implement as they depend on the actual hardware platform and components used in the testbed. Moreover, they need to map virtual components to physical hardware.

In this work we present an architecture for a semi-virtualized testbed cluster with a central server offering multiple student workspaces. It facilitates automatic orchestration and simplified management and introduces multiple new features that are enabled through the new architecture. The single server is based on the x86 platform and makes use of current virtualization techniques such as VT-x [4], VT-d [5], VT-c [6], and SR-IOV [7]. They enable VM performance close to physical machines, in particular regarding network throughput. For orchestration purposes, we developed the "Lab Orchestrator for Semi-virtualized Testbeds" (LOST). It is a Python-based platform for orchestration of VMs in a semi-virtualized environment. It makes use of the kvm [8] hypervisor to run VMs and leverages features of libvirt [9] to manage them.

LOST groups several student VMs into virtual workspaces and maps them to physical workspaces. The physical workspaces enable the students to interact with the VMs in a similar way as with physical machines and configure network topologies with the help of cables and switches. In addition, LOST supports usage of USB devices plugged into physical workspaces by associating them with student VMs.

LOST supports different types of VMs with different roles (clients, servers, and routers). It instantiates them from templates that are derived from a base image. To improve the manageability and to speed up orchestration of VMs, we developed a layered concept for file system access of VM images: a jointly used base image and additional layers for typing and individualization reduce memory copies upon instantiation of VMs.

The rest of the paper is structured as follows. Section II gives an overview of different lab environments. In Section III, we explain the general concept of the semi-virtualized testbed cluster architecture. LOST and its features are introduced in Section IV. The hardware and software platform of the testbed are presented in Section V. Section VI concludes this work.

II. Related Work

Hands-on networking courses are quite common in computer science and IT education. Therefore, various concepts for networking lab infrastructure have been published. This section reviews different approaches.

In [1] Liebeherr and Zarki provide a manual for a hands-on Internet lab. The book contains a course syllabus and instructions for setting up the lab hardware. The student workspaces consist of 4 physical Linux PCs and 4 Cisco routers each. The authors of [10] describe a networking lab testbed based on 3 PCs, one laptop computer, and two multi-protocol routers. Additionally, central networking equipment is proposed for management purposes and inter-testbed connectivity.

Emulab [11] is a testbed platform for networking research. It allows to connect physical nodes or VMs over virtual networks. Topologies and link characteristics are modeled using VLANs and transparent traffic shaping nodes. In addition, simulated nodes can be integrated into experiments. A similar implementation for teaching purposes is described in [12]. The physical nodes can be connected by configuring VLANs in a web interface instead of plugging cables.

Other hands-on labs are entirely built on VMs. The authors of [13] propose a setup with 120 PCs and routers running as Xen VMs on a single host, connected by virtual switches. dVirt [14] is a virtual BGP testbed using Xen VMs as routers. V-Lab [15] is a remote access lab with VMs on a XenServer cluster. VMs are configured and connected using VLANs via a web-based management panel.

Mininet [16] is a virtual network experiment testbed based on lightweight containers instead of VMs. Its main focus is SDN. Mininet emulates multiple nodes and links in multiple containers on a single machine. Their virtual interfaces are directly connected to a software switch.

The authors of [17] describe a reconfigurable network lab based on VMware. Ethernet interfaces of the physical machine can be assigned to VMs to provide physical access to network interfaces of VMs. This approach requires at least as many physical network interface cards (NICs) as virtual NICs. As the number of pluggable NICs per machine is limited, we use a different approach which can support more virtual interfaces than the number of Ethernet ports on a host.

In previous work [2], [3], we described the predecessor of the lab platform presented here. In contrast to this architecture, it required one server per workspace. Each server hosted eight VMs acting as clients, servers, and routers. All VMs are individual Linux installations managed with puppet [18]. Dynamic orchestration is not possible. The students have access to the VMs directly through the server which is equipped with an additional graphics card, i.e. the server is used as a workstation.

III. Concept

In this section, we present the overall architecture for the semi-virtualized student testbed cluster. A single lab server hosts multiple virtual workspaces that are mapped to physical workspaces for physical user access. Additional infrastructure VMs are used to provide services for the virtual workspaces and the students. We suggest an optimized, layered storage organization for VMs to reduce management overhead and to improve software consistency. The technical realization of the mapping is described in Section V as it depends on the hardware platform of the lab server. Finally, we compare the new concept for semi-virtualized student testbed cluster with other approaches regarding acquisition cost, energy consumption, and maintenance effort.

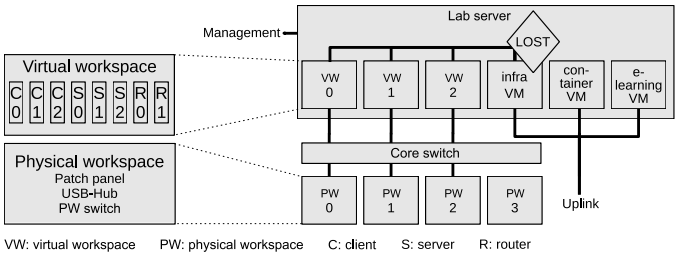


Fig. 2: Architecture of the semi-virtualized lab infrastructure on a single server.

A. Architecture of the Semi-Virtualized Testbed

Figure 2 shows the overall architecture of the semi-virtualized testbed. A single lab server hosts virtual workspaces (VWs) with student VMs (SVMs) as well as infrastructure VMs (IVMs). A physical workspace (PW) consists of a set of devices giving access to the SVMs of a virtual workspace. The "Lab Orchestrator for Semi-virtualized lab Testbeds" (LOST), presented in detail in Section IV, is a collection of scripts that run on the host and on IVMs. It sets up the virtual workspaces and maps them through a core switch to the physical workspaces so that students can interact with SVMs and interconnect them with cables and unmanaged switches.

A *virtual workspace* is a set of SVMs that represent a students' workspace on the server. Figure 2 illustrates a typical setup which consists of three client (C) SVMs, three server (S) SVMs, and two router (R) SVMs. All virtual workspaces use the same setup which is defined by a template for LOST.

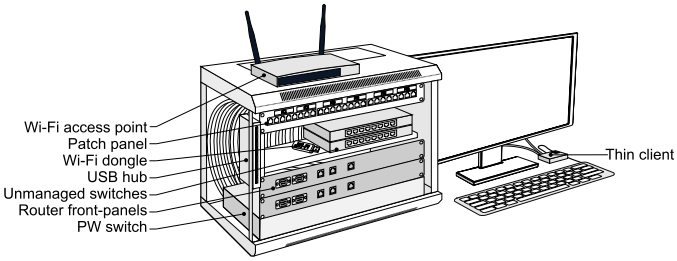


Fig. 3: The physical workspace provides access to the desktop and outlets of the virtual workspace.

A *physical workspace* is the actual workspace that allows students to interact with the VMs. Figure 3 depicts a physical workspace. It consists of a 19 inch cabinet, a thin client for the student user interface, and I/O devices. The cabinet contains the following components. The PW switch is a managed switch at the bottom of the cabinet. It connects the cabinet via the core switch to the lab server hosting the VMs like shown in Figure 5. The PW switch demultiplexes the interfaces of the VMs in the corresponding virtual workspace to individual switch ports. The ports for the network interfaces of clients and server VMs are connected to a patch panel at the top-most position in the cabinet. The ports for interfaces of the routers are connected to custom-designed front-panels that look like traditional routers (see Figure 3). To enable the students to set up more complex network topologies, two additional unmanaged switches are placed on a compartment sheet in the middle position of the cabinet. For experiments involving wireless technology based on IEEE 802.11, a Wi-Fi access point is placed on the top of the cabinet and USB Wi-Fi dongles for selected VMs are provided. These dongles can be plugged into a USB hub that is mounted on the left side of the cabinet and connected to the server.

B. Interconnection Network and VW-PW Mapping

Figure 4 gives an overview of the internal structure of the single server lab infrastructure and the connection between the different components. In the figure, the server hosts three VWs and each VW holds three VMs. Each SVM of a VW contains one infrastructure network interface (INI) and at least one student network interface (SNI) which can be configured by the students. The number of SNIs depends on the role of the SVM. All INIs are connected through an internal bridge per VW on the host to an IVM that provides infrastructure support (see Section III-C). The patch panel of a PW contains outlets for the SNIs so that they are physically accessible and the students can perform the cabling for the labs. To that end, all SNIs are realized as virtual interfaces on a two-port NIC that is connected to the core switch. The mapping of virtual interface to physical interface depends on the absolute numerical identifier of the virtual interface. All even numbers are mapped to the first port of the two-port NIC and all odd numbers are mapped to the second port of the two-port NIC.

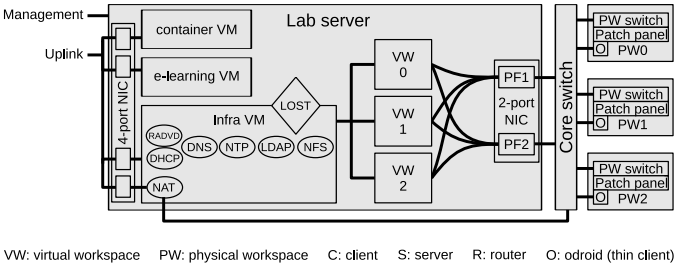


Fig. 4: Overview of componets, services, and connections of the single server testbed.

Figure 5 illustrates the mapping of the SNIs to the patch panel and the connection of the thin clients to the lab server. To distinguish the different interfaces, each is placed in a unique VLAN [19] with the following tag scheme: `$pw$svm_num$interface_num`. That means, all VLAN tags consist of a three-digit number with the most significant digit specifying the physical workspace, the second-most significant digit specifying the SVM, and the least significant digit specifying the interface within an SVM. VLAN tags are automatically added/stripped at the transition from SVM to host. The mapping of `$pw` to actual PW is dynamically configured by LOST (see Section IV).

The host forwards incoming packets to the corresponding SVM according to the VLAN tag. In the other direction,

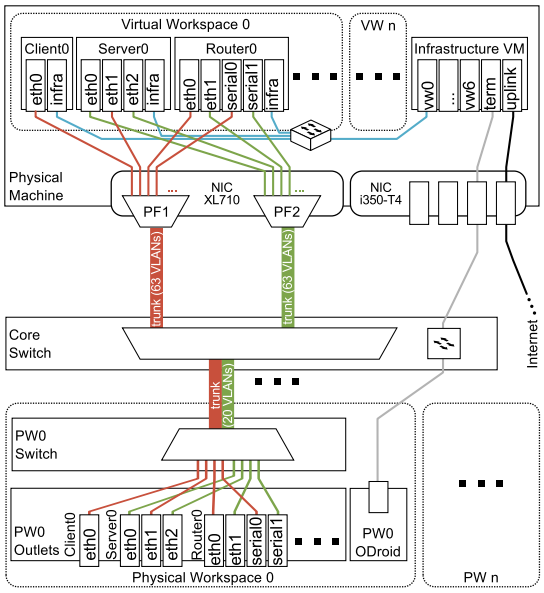


Fig. 5: Interconnection of SVMs on the lab server with patch panels on PWs.

packets from the SVM are forwarded to the core switch as two big VLAN trunks via the two-port NIC.

On the core switch, the VLAN trunks are demultiplexed into sub-trunks that are forwarded to the PWs specified in the VLAN tag. Within a PW the VLAN trunk is processed by a managed switch. The switch is used to re-order the VLANs to the corresponding port in the patch panel and to strip the VLAN tags. This mechanism ensures that the students do not see any VLAN tags.

C. Infrastructure Support

We run three supplementary IVMs on the lab server that host the lab platform and provide services. The first IVM facilitates basic network services for both the SVMs and the thin clients in the PWs. Each VW gets its own dedicated subnet whereas all thin clients are placed in the same subnet. To that end, a DHCP server and a router advertisement daemon distribute IPv4 and IPv6 addresses, routes, and information about additional services like DNS and NTP. The DNS server uses the following canonical naming scheme for names in the lab: `${host_name}.tb${vw_num}.inetlab`. Queries to resources in the Internet are forwarded to an external DNS server. The local NTP server syncs its time information with an external time server and ensures that the time in all VWs is in sync. This IVM also acts as NAT gateway so that the thin clients of the physical workspaces can connect to the Internet. The first IVM also includes a central LDAP [20] directory and an NFS [21] server. The LDAP directory stores the accounts for the students as well as their user rights within the SVMs. An NFS server holds the home directories of the students as well as some initial configuration data and scripts that the students use during the exercises. The other two IVMs host the e-learning platform for the course and provide LXC [22] containers for special home exercises, respectively.

D. Simplified VM Maintenance

We first explain the need for simplified VM maintenance in the context of testbeds, then we review concepts related to our solution, and finally we introduce the new VM maintenance method.

1) Motivation: In our testbed, many VMs need to be maintained, i.e., configuration changes and software updates need to be applied. Manual maintenance is a lot of effort and error-prone. Therefore, maintenance work is often automated with scripts or configuration management tools like puppet [18]. Different maintenance times or additional changes of individual VMs by the administrator lead to diverging VM images. However, different VM images are undesirable in testbeds where at least all VMs of the same type should be identical at the beginning of an exercise.

2) Related Concepts: QCOW2 is an updated copy-on-write hard disk container to provide hard disk storage for Qemu-based VMs. It allows to save a VM image as a stable base image file and to store later changes to the VM image in a snapshot file. This facilitates the reset of the VM by deleting the snapshot file.

In [23], this technique has been used to share a major portion of a VM image on hard disk among multiple VMs. They are booted from the same QCOW2 base image and record their image changes in individual snapshot files. Various base images supported different VM types. This technique was used in [23] to save disk space and in particular to reduce copy operations for VM setup.

OverlayFS for Unix/Linux implements a stacked file system. It combines a lower and upper file system, i.e., the lower file system serves as a stable base and the upper file system accounts for differences. Thus, the upper file system tracks changes, i.e., file generation, deletion, and modification. File requests are served from both combined file systems like from a single file system. OverlayFS can be applied recursively, i.e., the lower file system may be another OverlayFS file system.

3) Multi-Layer VM Images: We present a novel maintenance method for VMs with similar configuration. It is based on the observation that the VMs share a major portion of their images and the differences result from a moderate number of additional configuration actions. It is helpful for maintaining VMs with different host names, network configuration, and services. In [23], several base images are needed to support different VM types. Now, the objective is to utilize only a single base image for different VM types.

We diversify VMs by leveraging OverlayFS and providing configuration changes in the upper file system of OverlayFS. To simplify the provisioning of the upper layer file system, we define templates for SVMs with root file systems for clients, servers, and routers that are further adapted to the specifics of individual VMs.

However, OverlayFS becomes active only after loading of the operating system kernel has completed. Until then, VMs

write all data to the lower file system. If multiple VMs leverage the same lower file system, inconsistencies will occur. To avoid inconsistencies due to writes from different VMs to the same base image, QCOW2 is utilized as protection layer. As a result, VMs are booted from a joint base image containing the lower file system and modifications are tracked in individual QCOW2 snapshots. When OverlayFS is started, VMs are diversified through the configuration contained in the upper file system, the adaption layer.

When student work on VMs, they apply configuration changes. It is desirable to easily undo them and restore the VM to a defined starting point. To that end, we start another OverlayFS after VM diversification to track all further modifications to the VM in a second upper file system, the user layer.

Technically, the base image combined with the secure base appear one disk, the adaption layer appears as another disk, and the user layer appears as a third disk. These three disks are input to OverlayFS whereby their order matters. However, the disks do not necessarily always appear in the same order, which affects their numbering. Thus, the disk numbers cannot be used as input for OverlayFS. We fix that problem with the following workaround. The disk contains metadata including a disk label as another identifier. This disk label is set after creation of the disk and remains stable. Therefore, we specify the inputs for OverlayFS using the disk labels rather than the disk numbers. In our prototype, we use ext4 as file system type and e2label [24] as disk label implementation.

Figure 6 illustrates the proposed concept. A single LVM (Logical Volume Manager [25]) volume constitutes the base image jointly used by all VMs. The protection layer is individual for all VMs and achieved through QCOW2. Its snapshot (secure base) intercepts initial runtime modification of the VM to protect the joint base image. The adaption layer is also individual for all VMs and implemented through OverlayFS. It combines a VM's base protection layer with the initial upper file system holding the VM's configuration data and minor runtime modifications. Finally, the user layer is also implemented through OverlayFS and holds all student changes in its upper file system.

After completion of an exercise, the user layers can be deleted to reset VMs to the states defined by their adaption layers. To maintain all VMs in the testbed, only the base image needs to be updated provided that maintenance operations do not affect the folders with the configuration data of the adaption layer. Therefore, this concept provides high flexibility, minimizes maintenance overhead, and reduces storage requirements.

E. Comparison: Acquisition Cost, Maintenance Effort, and Energy Consumption

In this section we provide a comparison regarding acquisition cost, maintenance effort, and energy consumption of the following three lab system approaches: 1) a physical testbed, 2) a semi-virtualized testbed with a single server per testbed, 3) a semi-virtualized testbed cluster as proposed in this work.

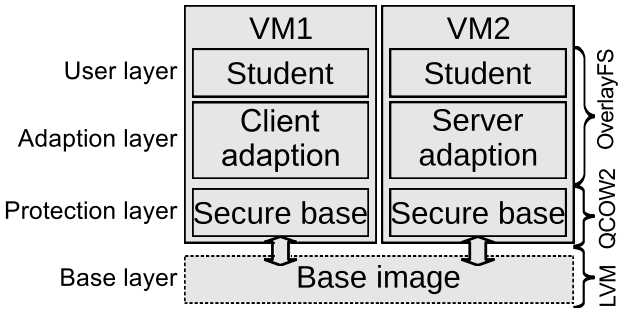


Fig. 6: File system access based on layered storage.

1) Acquisition Costs: We estimate acquisition costs for the components of the three different lab system approaches. We assume reasonable, cost-efficient prices as performance is not crucial.

For the physical testbed we assume that cheap all-in-one PCs are used and upgraded with a 4-port network card. This will result in around 200 € per PC and 20 € for the network card. In addition, two routers are needed. Recent or high-performance router models are not needed since all basic functions used by students are implemented in router firmware for more than 20 years. Therefore, cheap, old models may be utilized which are available for 50 € each. In total a single testbed costs 1440 €. An infrastructure server providing services can be acquired for around 1200 € in total. A typical lab system consisting of 6 testbeds cost roughly 9500 €.

The semi-virtualized testbed with a dedicated server needs a physical workspace which costs around 800 € each, including the PW switch. In addition, a dedicated server per PW is needed as host machine to run the hypervisor for the VMs. We use the same machine type as for the infrastructure server for this purpose. A lab system with 6 work-spaces amounts $7 \cdot 1200$ € $+ 6 \cdot 800$ € $= 13200$ €.

The proposed architecture only needs one big server for the entire testbed cluster which costs around 7000 €. The physical workspaces are the same as for the semi-virtualized testbed with dedicated server and also cost 800 €. In addition, a core switch is needed which costs around 400 €. This results in total costs of 7000 € $+ 6 \cdot 800$ € $+ 400$ € $= 12300$ €.

2) Maintenance Effort: In the past, we encountered a maintenance effort of about 1 hour per physical machine and semester in all three lab system approaches. That means, for the physical testbed, each server, PC, and router requires that maintenance effort. For the semi-virtualized testbeds, the servers require that maintenance effort, but in addition, the VM images need to be kept up to date. In case of one server per PW, care needs to be taken for the VMs on all servers. Experience has shown that the required effort scales with the number of PWs although there are options for automation. In practice, we needed about 3 hours per PW and semester for

maintenance. In case of a single server for all PWs, only a single VM needs to be maintained. That requires only 3 hours maintenance effort per testbed cluster and semester.

3) Energy Consumption: The power consumption of small all-in-one PCs is estimated with around 20 W. We observe power consumption of 200 W for small servers and 300 W for a big server. The PWs are powered with 25 W each for all active components (PW Switch, small switches, USB).

Table I compares the three approaches in terms of cost, maintenance and power consumption.

TABLE I: Comparison of acquisition cost, power consumption, and maintenance effort for physical testbeds, virtualized testbeds with a dedicated server, and virtualized testbeds with a single server.

Testbed type	Acquisition cost (€)	Maintenance effort (h)	Power consumption (W)
physical testbed	9500	49	1160
semi-virtualized testbed	13200	18	1550
semi-virtualized testbed cluster	12300	3	450

IV. A Lab Orchestrator for Semi-Virtualized Testbed clusters (LOST)

As no existing orchestration platform like OpenStack provides the features that are required for our use case, we developed *Lab Orchestrator for Semi-virtualized lab Testbed clusters* (LOST) as a new one. It is especially designed to support the VM storage concept, the simplified VM maintenance, and the template system for SVMs and VWs with their mapping to PWs presented in Section III. It mainly consists of a set of Python scripts with some additional bash scripts.

LOST is configured via config files including the following parameters:

- The templates for the different SVMs include the adoption layers and specify the the hardware resources like the amount of network interfaces (NICs), CPU, and RAM. We have templates for clients, servers, and routers. E.g., the routers provide a Cisco-like CLI, the clients have a graphical desktop environment, and the servers are terminal machines that run several services.
- The composition of a VW consisting of different SVMs is defined. In our use case, we have three clients, three servers, and two routers (like depicted in Figure 2) as that fits best to the different exercises in the courses. It is easily possible to define other composition, the available hardware resources on the lab server are the only limitation.
- The amount of available PWs is configured, so that a VW for every PW can later be instantiated. We have 6 workspaces.
- A default mapping of VW to PW is provided. Figure 8a depicts such a default mapping.

With the configuration files as input, LOST instantiates the different VMs and configure the core switch as well as the PW switches according to the confi files. E.g. the different VLANs are configured. The configuration of the switches leverages software-defined networking (SDN) technology. Generally, LOST supports different southbound interfaces like SNMP [26], OpenFlow [27] or NETCONF [28]. For compatibility reasons, we currently selected SNMP.

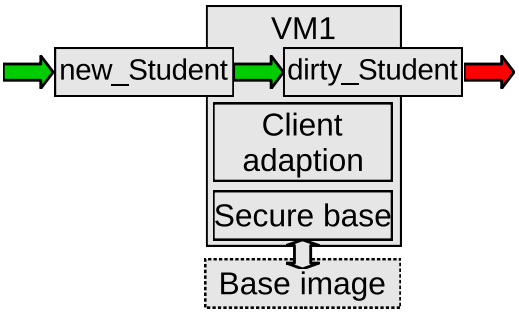


Fig. 7: Wiping the user layer restores the original SVM states.

The instantiation process includes the initialization of the protection layer and the adoption layer (see Section III-D) for the individual VMs and assigning hardware resources such as NICs, CPU, and RAM to them. LOST also manages, i.e., this includes starting and stopping of VWs, and monitors the different VMs. VMs can either be managed individually or as bulks. After a lab day or in case of a heavily misconfigured SVM, LOST can wipe the student layer so that the students can start over again with a well-defined VM states (see Figure 7).

In case of a hardware failure on a PW, it is possible to move a VW to another PW as depicted in Figure 8b. To that end, LOST reconfigures the affected PW switches and the core switch. The advantage of this mechanism is that students only have to re-apply the cabling on the new PW, but all their configuration and data of the VMs are available at the new location.

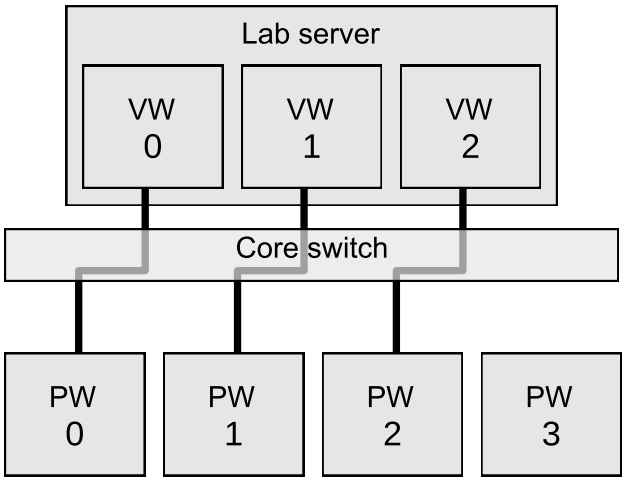
V. Implementation

In the following, we describe the hardware, software, and virtualization platform that we used to implement the semi-virtualized testbed.

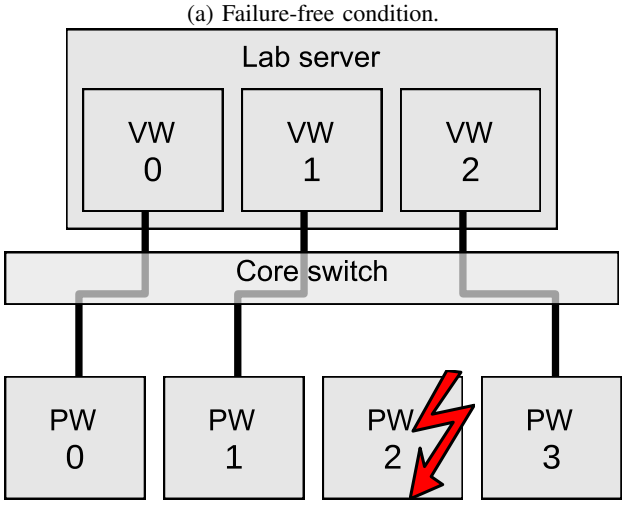
A. Hardware Platform

First, we outline the components included in the single lab server and the additional equipment used to provide the physical interaction with the VMs.

1) Server: We use a DELL PowerEdge T430 based on the current Intel server platform as base system for the lab server. The server is equipped with two Intel Xeon CPU E5-2660 v4, 128 GB RAM, and a hardware RAID with level 6. Figure 5 shows that two additional Intel network cards provide the network interfaces for the SVMs and IVMs. An Intel XL710 40GbE SFP+ two-port NIC which supports 64 virtual interfaces per physical interface, relays the network interfaces for the student VMs. An additional Intel i350-T4 NIC provides the uplink to the Internet for the three IVMs and connects one of them to the lab network. The onboard NIC of the server is used as management access for the server.



(a) Failure-free condition.



(b) Re-mapping of in case of failed PW.

Fig. 8: Mappings of VW to PW.

As we want to be able to equip the student VMs with USB devices like Wi-Fi dongles or headsets, a USB3 hub is attached to the server. It connects the USB hub in each PW via a fast port USB to the lab server. The hub allows to attach USB devices like Wi-Fi dongles or headsets to the SVMs.

2) Additional equipment: The core switch connecting the different PWs to the server is a Netgear S3300-28X ProSAFE 24-Port Gigabit. It includes two SFP+ 10G uplink ports which are connected to the Intel XL710 at the lab server. Each physical workspace has two connections to the switch: one is used for the thin clients and the other is used to connect the cabinets with the server. Each cabinet contains a managed HP/Aruba switch as PW switch to de-/multiplex the testbed uplink to different physical ports on the switch. They are connected to a patch panel containing keystones for the network outlets of the VMs. The USB hub in each cabinet is connected to the USB3 hub at the lab server via an active USB cable. A Hardkernel odroid c2 is used as thin client. This platform provides enough computational power and RAM to act as graphical client or terminal client for 8 VMs and to run the Chrome web browser. The odroids are connected to a 24 inch full-hd (1080p) monitor, keyboard, and mouse for interaction.

B. Software Platform

We briefly outline the software used on the host and the different VMs.

1) Host: The server itself only acts as host for VMs. For stability purposes, more efficient use of resources, and reduced security risks, the server contains a minimal environment. We chose Gentoo Linux [29] as operating system (OS) on the server as it gives full control to the administrator. With the help of USE-flags it is possible to define which parts of a certain package should be installed. This way, only the parts and features of a package that are really needed and wanted are installed. Additionally, Gentoo does not force the user to use a certain component, e.g., the user can freely select the init system. Furthermore, in our experience Gentoo is more stable compared to other Linux distributions like, e.g., Ubuntu.

We chose kvm [8] as hypervisor because it is already part of the Linux kernel. As kvm cannot be used directly, a supplementary virtualization tool like qemu [30] is required. Qemu makes use of kvm and provides additional virtual devices by itself. Also parts of LOST (see Section IV) run on the host. To ease the handling of the VMs in LOST, we designed LOST to utilize the libvirt [9] framework which already implements functions to create, start, stop, and delete VMs. These operations are extended with custom operations, like applying a template to a VM during creation as needed.

2) Infrastructure VMs (IVMs): As already explained in Section III-C, we run three additional server VMs providing services for the lab and hosting the lab platform. The first IVM runs a DHCP server and a radvd [31] daemon to configure the network addresses. A BIND [32] DNS server is used to resolve the IP addresses for internal and external resources. The LDAP directory runs on an OpenLDAP [33] server and the NFS kernel server provides NFS directories. The reference implementation [34] from the NTP project is used as NTP server.

We use the nginx [35] webserver to host an instance of the iLab e-learning platform [36]. In conjunction with shellinabox [37], nginx also enables access to LXC [22] containers. The containers are used in an introductory assignment which takes place before the practical course. This assignment compiles an introduction for the students how to work on a Linux shell, do basic network configuration and simple experiments.

3) Student VMs: (SVMs) All SVMs are based on a single base image. The template system of LOST derives different kinds of SVMs like client or server from the base image. This base image already contains most of the tools and software for all different types of SVMs. Among this software is the LXQt [38] desktop environment used for clients, services, and daemons used for the servers and the quagga [39] routing suite from which we use vtysh (a Cisco like CLI) for virtual routers. However, the functionality of this software is disabled in the

base image. The templates activate the functionality required for their use case. The thin clients can connect to the SVMs via spice [40] which transports the display and input devices of the VMs.

As an alternative to VMs, containers could be used. However, we decided to use VMs because VMs have their own networking stacks so that VM nodes are isolated against each other and students observe a similar networking performance as with real hardware. The performance obtained in this setting is sufficient for most applications in general and suffices for all exercises carried out by the students. Using VMs instead of containers provides sufficient flexibility for future extensions. We plan to define an data center networking lab where VMs act as servers hosting containers. Implementing data center servers as containers would lead to a rather unrealistic solution. Thus, the decision to utilize VMs instead of containers on the host makes LOST future-proof.

C. Virtualization Platform

In the following, we describe virtualization features and techniques that we use on our infrastructure. First, we briefly outline the basic requirements for virtualization on the x86 architecture. After that, we explain the mechanisms used to pass-through PCI and USB devices from the host to the VMs.

1) Hardware-Assisted Virtualization for x86: The x86 platform itself is not virtualizable in hardware, which means that VMs must be emulated in software. Software-based VMs lack performance. Hardware-assisted VMs require some extensions to the x86 architecture: VT-x [4] enables basic hardware acceleration for virtualization. It includes additional instructions and registers to implement an additional privileged system and hardware-based shadowing for the memory management unit (MMU). This way, VT-x permits entering and exiting a virtual execution mode. In this mode the host OS remains protected while the VM OS perceives itself as running with full privilege. Second level address translation (SLAT) is an additional extension to the MMU which further increases the performance. It basically treats each physical address of a VM as a virtual address on the host. This prevents software lookups to determine the actual physical memory address of VM memory. SLAT is implemented on the Intel platform as extended page tables (EPT) [41]. Additionally, EPT is a requirement to start a VM directly in real mode with unrestricted access. Typically, hypervisors emulate most guest access to interrupts and the advanced programmable interrupt controller which requires the exit and entry of a VM. This is time-consuming and a major source of overhead. Advanced programmable interrupt controller virtualization (APICv) [42] eliminates lots of VM exits and can increase performance significantly.

2) Pass-through of PCI Devices: VT-d [5] provides an IOMMU [7] which allows to pass-through devices, e.g., NICs from the host to the VMs. With the help of IOMMU groups, different devices are isolated against each other and secure memory access is ensured. Linux kernels later than 4.1 require the PCIe Access Control Services (ACS) [43] feature for separated IOMMU groups. VT-c [6] comprises Single Root I/O Virtualization (SR-IOV) [7], an extension to the PCI standard, and Virtual Machine Device Queues (VMD-q) [44]. SR-IOV classifies devices in physical functions (PFs) and virtual functions (VFs). A PF is a full-featured PCI device, e.g. a NIC or a graphics card, which can run on its own. A VF is a lightweight PCI device which cannot run independently of a PF. The VF shares some resources with the PF that manages this VF. VMD-q enables multiple hardware-based queues per NIC which are internally connected to a bridge. Together, VT-d and VT-c instantiate a dedicated hardware queue per VF which appears on the host as a virtual NIC. This NIC can be exclusively passed-through to a VM so that the host does not see the device any longer. With this mechanism the virtual NICs achieve a performance close to dedicated physical NICs. As all VFs communicate over the common physical port of their PF, VLAN tags are used to differentiate traffic from and to different VFs. To be able to distinguish traffic for the different VFs, they are typically mapped to different VLANs [19].

3) Pass-through of USB Devices: To pass-though USB devices, the entire address of a device on the USB bus is mapped to the address space of a VM. This process can be automated with the help of special udev rules [45] that trigger qemu to map a newly plugged in USB device to a specified VM. This requires a lookup in the LOST configuration so that USB devices must be identifiable. Typically, a USB device can be identified by a unique identifier (UUID) in the device description field of its ROM. However, some hardware manufactors and vendors disregard the uniqueness of UUIDs and assign the same UUID for multiple devices or entire batches thereof. That problem pertains to the USB Wi-Fi dongles and USB headsets in our system. We developed the following workaround to connect USB devices with equal UUIDs in VMs.

When a USB device is plugged into the USB bus, the host loads the driver for that device. If the USB device is a network device, the host looks up its unique MAC address. If that MAC address is assigned to some VM in the LOST configuration, the host unloads the USB driver for that device and passes its USB bus address through to the configured VM. This workaround allows to assign a specific Wi-Fi dongle to a VM, which is needed because some scripts and udev rules in the VM contain their MAC address. The described workaround works for Wi-Fi dongles, but not for USB headsets as they do not have MAC addresses. Fortunately, a VM does not require a specific headset, but can work with any headset connected to the VM. Therefore, the $n-th$ plugged-in headset is passed-through to a specific VM that is configured with LOST depending on n.

VI. Conclusion

We presented a semi-virtualized testbed cluster for networking labs on a single server. It consists of student VMs organized in virtual workspaces that can be accessed through physical workspaces. The physical workspaces provide outlets

for the network interfaces of the student VMs and allow to plug in USB devices that are mapped into the student VMs. We developed a "Lab Orchestration for Semi-virtualized Testbeds" (LOST) to manage the student VMs and dynamically define the mapping of virtual workspaces to physical workspaces. LOST implements a template system to realize different types of student VMs such as clients, servers, and routers. To efficiently manage the student VMs, all of them are based on a single base image and served from the same memory. The template system and the writable root file system are realized as multiple overlay layers to the base image which are private to each student VM. The top-most layer holds the data and configuration files made by the students in an assignment. This layer can be wiped to go back to a safe, stable, and deterministic state after an assignment. Additional infrastructure VMs provide the student accounts via LDAP and NFS, as well as basic network infrastructure such as NAT, DNS, DHCP, etc.

Our platform combines the advantages of a fully virtualized testbed and an entirely physical testbed. It allows students to gain hands-on networking experience by configuring network nodes and interconnecting them with cables and switches. In addition, our approach consists of only one server that runs VMs, which enables the typical benefits of virtualization. It is cost-effective and saves both energy and space compared to a physical testbed. Most important, thanks to LOST, it is relatively easy to manage and flexible. The suggested architecture is extensible as we can easily add more virtual and physical workspaces or components. We successfully leverage the presented platform for our hands-on networking courses since winter semester 2016/17.

REFERENCES

[1] J. Liebeherr and M. E. Zarki, *Mastering Networks – An Internet Lab Manual*. Pearson Education, 2003.

[2] 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.

[3] ——, "A Virtualized Testbed with Physical Outlets for Hands-on Computer Networking Education," in *ACM SIGITE*, 2014.

[4] Intel Corp., "Intel Virtualization Technology (VT-x)," 2006.

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

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

[7] PCI SIG, "Single Root I/O Virtualization and Sharing Specification 1.1," http://www.pcisig.com/specifications/iov/single_root/, 2010.

[8] A. Kivity *et al.*, "kvm: the Linux Virtual Machine Monitor," in *Linux Symposium*, 2007.

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

[10] 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.

[11] B. White *et al.*, "An Integrated Experimental Environment for Distributed Systems and Networks," in *USENIX OSDI*, 2002.

[12] S. C. Sivakumar *et al.*, "A Web-Based Remote Interactive Laboratory for Internetworking Education," *IEEE Transactions on Education*, vol. 48, no. 4, pp. 586–598, 2005.

[13] 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.

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

[15] 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.

[16] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *ACM HotNets*, 2010.

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

[18] L. Kanies, "Puppet: Next-Generation Configuration Management," *The USENIX Magazine*, vol. 31, no. 1, 2006.

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

[20] J. Sermersheim, "Lightweight Directory Access Protocol (LDAP): The Protocol," RFC 4511 (Proposed Standard), Internet Engineering Task Force, Jun. 2006.

[21] S. Shepler *et al.*, "Network File System (NFS) version 4 Protocol," RFC 3530 (Proposed Standard), Internet Engineering Task Force, Apr. 2003.

[22] D. Lezcano, S. Hallyn, S. Graber *et al.*, "Linux Containers," https://linuxcontainers.org/, 2008.

[23] A. Stockmayer, C. Kindermann, and M. Menth, "VITO: VIrtual Testbed Orchestration for Automation of Networking Experiments," in *VALUETOOLS 2017*, 2017.

[24] Theodore Ts'o, "e2fsprogs," https://sourceforge.net/projects/e2fsprogs/, 2016.

[25] Red Hat, "Logical Volume Manager (LVM)," https://sourceware.org/lvm2/, 2016.

[26] J. Case *et al.*, "Introduction and Applicability Statements for Internet Standard Management Framework," RFC 3410 (Proposed Standard), Internet Engineering Task Force, Dec. 2002.

[27] Open Networking Foundation members, "OpenFlow Switch Specification," The Open Networking Foundation.

[28] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," RFC 6241 (Proposed Standard), Internet Engineering Task Force, Jun. 2011.

[29] Gentoo Foundation, "Gentoo Linux," http://www.gentoo.org, 2017.

[30] F. Bellard and QEMU team, "QEMU – the FAST! processor emulator," http://wiki.qemu.org/ChangeLog/2.8, 2016.

[31] R. Hawkins *et al.*, "Linux IPv6 Router Advertisement Daemon (radvd)," http://www.litech.org/radvd/, 2016.

[32] Internet Systems Consortium, "BIND," https://www.isc.org/downloads/bind/.

[33] The OpenLDAP project, "openldap," http://www.openldap.org/.

[34] Network Time Foundation, "The Network Time Protocol Project," http://www.ntp.org/.

[35] Sysoev, Igor and Nginx, Inc., "nginx," https://www.nginx.com/.

[36] M. Pahl, "The ilab concept: Making teaching better, at scale," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 178–185, 2017.

[37] M. Gutschke *et al.*, "shellinabox," https://github.com/shellinabox/shellinabox.

[38] The LXQt Team, "The Lightweight Qt Desktop Environment," http://lxqt.org/.

[39] B. Gurudoss, P. Jakma, T. Teräs, G. Troxel, and Quagga developer team, "Quagga Routing Suite," http://www.nongnu.org/quagga/.

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

[41] Intel Corp., "Technology Brief: Intel Microarchitecture Nehalem Virtualization Technology," ttp://download.intel.com/business/resources/briefs/xeon5500/xeon_5500_virtualization.pdf, 2009.

[42] ——, "APIC Virtualization Performance Testing and Iozone," https://software.intel.com/en-us/blogs/2013/12/17/apic-virtualization-performance-testing-and-iozone, 2013.

[43] PCI SIG, "Root Complex Integrated Endpoints and IOV Updates," 2015.

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

[45] G. Kroah-Hartman, "udev – A Userspace Implementation of devfs," in *Linux Symposium*, 2003.