Prototyping Prototyping Facilities: Developing and Bootstrapping Testbeds

Sebastian Gallenmüller, Eric Hauser, Georg Carle
Department of Informatics, Technical University of Munich
Garching near Munich, Germany
{gallenmu, hauser, carle}@net.in.tum.de

***Abstract*—The creation of prototypes is a convincing approach, demonstrating the feasibility of scientific ideas. Testbeds act as enablers for such prototypes, contributing the facilities to their construction. In this paper, we apply a prototype-driven approach to the development of the testbeds themselves. Thus, we select abstractions and APIs to modularize testbeds to allow a selective adaptation or substitution of specific components. To minimize costs, our approach aims to consolidate all components into a single system. Hence, we named it *testbed on a single system (toast)*.**

The single-server approach demands the recreation of entire components in software such as networks or experiment nodes. Simultaneously, the softwarization of components enables flexible network topologies and scalability. At the same time, we try to keep the behavior and the performance as close to a hardware-based testbed as possible. Therefore, we rely heavily on hardware acceleration of IO using techniques such as single root IO virtualization (SR-IOV). A case study compares the accelerated IO of *toast* to a hardware-based testbed and a testbed without IO acceleration. We want to use *toast* as a training and teaching environment and a prototype facility for future research infrastructures.

***Index Terms*—Testbed, Nework Experiments, Virtualization, SR-IOV**

I. Introduction

Testbeds act as a workbench for scientists where new ideas can be designed, tested, and refined in a controlled environment. A testbed's capabilities set the frame in which scientists can operate. Thus, the testbed defines the limit of scientific experiments and experimental outcomes. To enable further progress, existing testbeds must be continuously upgraded with new components, or entirely new testbeds must be created to reflect technological progress.

Because of their central importance to experimental research, we propose a concept to prototype new testbed components or entirely new testbeds. Instead of building a prototype out of real systems or components, many scientists rely on emulation, simulation, or virtualization of components or entire distributed systems [1]. An advantage of this approach is its affordability, as a single server replaces multiple, if not all components, of the real-world system. An additional feature is the increased flexibility, network topologies can be adapted, and virtual components such as participating nodes can be scaled easily. However, virtualizing multiple nodes on a single server leads to shared resources between them, thus, may impact the measured performance on virtualized nodes.

In this work, we create a testbed that combines affordability with a limited impact on performance. These two goals are achieved by utilizing virtualization techniques of modern CPUs and NICs that limit the computational overhead. We aim to create a *testbed on a single system (toast)*, ensuring affordability by folding entire virtual network topologies and systems onto a single physical server. This single-server testbed can be used as a stand-in replacement for a testbed to be built at a later point in time. The experiment scripts written for the *toast* can also be used in the future testbed. Testbed developers can use *toast* for prototyping various testbed components; future testbed users can be trained on the single-server testbed before the actual setup becomes available. The single-server testbed can also be used in teaching to attract potential future users.

In the following, we outline our goals aiming to

- design a concept to modularize testbeds to simplify the development or the replacement of testbed modules,
- create affordable testbed prototypes that rely on virtualization to realize flexible topologies as well as realistic performance,
- present a case study to reproduce experiments using the same experiment scripts of the *toast* approach to other experimental approaches and compare the performance, and
- introduce an online demonstrator for our platform that can be used by others.

The paper is structured as follows. Section II investigates related concepts and solutions. In Section III, we identify requirements for a flexible, virtualized testbed and outline a matching architecture. Our prototype implementation of the virtualized testbed is presented in Section IV, and a case study based on this prototype in Section V. As part of our effort towards reproducible research, we publish the experimental artifacts presented in this paper. Their reproduction is explained in Section VI. Section VII concludes the paper.

II. Related Work

Current testbeds for network experiments use pure software, hardware-accelerated software, or pure hardware. For the pure software and hardware-accelerated software approaches, simulation, emulation, or virtualization techniques are used.

a) Simulation: Creating a computer network model that is disconnected from reality. Mathematical models determine how packets circulate in the network by abstracting protocols and traffic. Rather than processing packets in real-time, simulations require more time than the actual measurement. Simulators such as *OMNeT++* [2] and *ns-3* [3] are scalable and easy to deploy. Due to the high level of abstraction, these purely software-based approaches do not process packets in real-time or at a significantly lower bandwidth [4]. Therefore, measured latencies only depend on mathematical models that cannot regard undiscovered effects.

b) Emulation: An integrated environment that tries to appear as a real computer network. Devices that connect to the emulator do not detect a functional difference compared to a real network. For the emulation of single links, *NetEm* [5] has been available in the *Linux* kernel since version 2.6. Links emulated by *NetEm* are configurable in terms of latency, packet loss, and duplication. Moreover, *Mininet* [6] is a widely used network emulator for research and teaching. Thereby, *Mininet* allows multiple lightweight virtualized hosts to form a custom network topology on a single computer. Even though the emulated network behaves like a real network, *Mininet's* performance is limited.

c) Real Hardware: Physical components are used to build a testbed, usually consisting of multiple machines. Due to the use of real hardware, these testbeds provide realistic measurement setups combined with high-speed performance. Vahdat et al. [7] present *ModelNet*, which abstracts a large network topology on a few computers. The testbed is separated into edge nodes and router nodes connected over a physical network. The edge nodes are regular computers that run user-specific applications. The computers that act as router nodes form the *ModelNet* core. Every physical core router forms a chain of pipes that represent an individual topology. Every pipe corresponds to a single hop in the emulated network, allowing multiple hop paths. These pipes have configurable characteristics like queueing model, buffer size, bandwidth, latency, and loss rate. White et al. [8] introduce *Netbed*, a testbed consisting of 218 experiment hosts distributed over two geographical locations. While *Netbed* features a maximum in realism, the system is complex because it depends on many components, which increases maintenance overhead. Gallenmüller et al. [9] present the *plain orchestrating system (pos)*, a testbed framework that follows a specific workflow to create inherently reproducible network experiments. There are two instances of *pos* available, an HW testbed and a virtualized clone. We use these two approaches for a direct comparison to the *toast* approach presented in this paper.

d) Virtualization: Abstracting network resources from hardware into a software network. Therefore, for example, real network interfaces become available to software consumers. The hardware support increases the performance significantly. Virtualization with hardware access to networking devices offers a tradeoff between realism, testbed complexity, and performance. Hibler et al. [10] show how multiple virtual machines on a single host access physical NICs to build scalable experiment networks.

A study by Emmerich et al. [11] demonstrates the performance of different VM topologies using *Open vSwitch*. They measured a performance decrease of up to $90\,\%$ when switching from a setup based on physical NICs to a virtualized setup.

Single-root IO virtualization (SR-IOV) is a technique that splits NICs into several virtual NICs called virtual functions (VFs). These VFs can be passed through to VMs, forming a hardware-accelerated IO path for VMs. Lettieri et al. [12] compared different techniques for VM IO. They identified SR-IOV as one of the fastest techniques with the lowest CPU utilization. Wiedner et al. [4] utilized SR-IOV to build and measure virtual topologies. They demonstrate that an SR-IOV-based system offers more realistic latency measurements than an emulated *Mininet* environment.

To differentiate our work from the already available techniques, we try to create a system that behaves more similar to real hardware than the available simulation and emulation-based solutions [2], [3], [6], or even virtualized solutions that still involve software packet processing [10]. At the same time, we want to provide a more convenient testbed experience that some systems lack [4], [8]. Therefore, we want to combine testbed controllers and hardware-accelerated virtualization [11], combining convenience with accuracy.

III. Testbed Requirements, Design and Architecture

This section introduces the requirements for a testbed, i.e., the preconditions necessary to execute effective experiments. Based on these requirements, we outline the design of the *toast* approach, a high-level overview of the different modules of a testbed and the interaction between them. The high-level design lays the foundation to deduct the *toast* reference architecture that defines interfaces between modules to simplify their development and substitution with other modules.

A. Requirements

We choose *Mininet* [6] as the starting point of our requirement elicitation. *Mininet* was developed to be *flexible*, supporting arbitrary topologies, OSes, and programming languages; *deployable*, requiring little changes between prototype and real-world system; *interactive*, running in real-time like a real network; *scalable*, supporting a multitude of network nodes; *realistic*, demonstrating real network behavior; and *sharable*, creating prototypes that can be shared with others. These requirements were defined for an emulation-based solution. However, we consider these goals as prerequisites to create meaningful, reproducible experimental results approximating real-world systems. Thus, the requirements can be considered universal for non-emulation-based testbed approaches.

When considering the different approaches, i.e., emulation, simulation, virtualization, and hardware-based testbeds, these platforms offer specific advantages and disadvantages. Therefore, developers must compromise leading to a platform-specific relative weight of the previously mentioned requirements. Emulators, such as *Mininet*, offer a high degree of

TABLE I: Requirements and their relative weight

	flexible	deployable	interactive	scalable	realistic	sharable
Mininet [6]	○	○	○	○	○	○
Simulation	○	○	--	○	○	○
HW testbeds	--	○	○	--	++	--
toast approach	○	○	○	-	+	○

realism from a functional perspective, but they cannot achieve the performance of real networks [4]. Simulators, for instance, sacrifice interactivity as simulation time rarely matches real time. Hardware testbeds offer the highest degree of realism, recreating a real system. However, flexibility, scalability, and sharability may be limited due to finite testbed resources such as servers or connectivity.

B. Design

The goal of our paper is the design of a stand-in replacement for an HW testbed. Therefore, we propose a design that values the different requirements in a similar way to hardware-based testbeds. To achieve similar behavior, our approach relies heavily on virtualization. Thus, we can achieve flexible topologies, running realistic deployments interactively that can be shared with others. Virtualization hypervisors allow fine-granular control over resource sharing between VMs. Hence, we expect a higher degree of realism compared to an emulation-based approach. However virtualization comes with limitations; consolidating multiple nodes onto the same physical machine leads to shared resources between VMs, impacting performance. To limit the performance impact, we have to limit the number of VMs on a physical machine. This restriction keeps performance close to an hardware-based testbed with separate physical instead of virtual machines.

Table I summarizes *Mininet's* original requirements and how the requirements of different approaches compare against them. *Mininet* presents the starting point of our comparison (represented by ○), + and - represent beneficial or sacrificial trade-offs in comparison. Out of the presented solutions in Table I, the *toast* approach offers the highest similarity with the HW testbed. *toast* offers benefits in terms of flexibility, scalability, and sharability. The main disadvantage is its degree of realism. However, this problem is inherent to any system that models another system. Therefore, only a testbed relying on real HW will provide the highest degree of realism. Out of the remaining non HW solutions, the *toast* approach offers the best compromise.

C. Architecture

Figure 1 shows the high-level architecture of our proposed approach. The entire testbed runs on a single physical machine or host system. The figure further shows the four modules that comprise *toast*: *(i)* the access module, *(ii)* the controller module, *(iii)* the experiment nodes, and *(iv)* the experiment network.

The **access** module is a dedicated interface providing an entry point to the testbed for its users. One of its main tasks

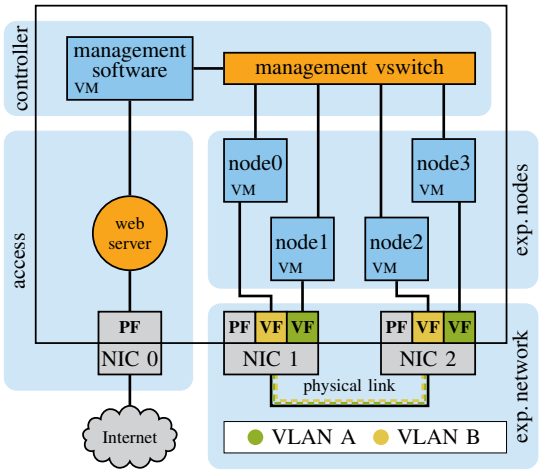


Fig. 1: *toast* architecture overview

is user authentication. We use an authentication webserver that provides access via browser to lower the entry barriers for new users; an alternative solution would be an SSH server. The entire authentication process is handled on the host system, entirely separated from the controller module. The authentication can be implemented, updated in case of security issues, or even replaced without touching the fully independent testbed controller. Another task of the access module is providing Internet access, enforcing bandwidth limitations, or port filtering for the controller module and experiment nodes.

The **controller** of the testbed is hosted on its own VM running the testbed management software. Encapsulating the management software into its own VM, ensures a high degree of flexibility for the controller. The controller developer can choose dependencies and OS independently of the host system. The controller must provide capabilities to create, reboot, and configure the experiment nodes. In the *toast* approach, the experiment nodes are connected to the controller via a virtual software switch (management vswitch). This virtual management network is exclusively used to control the experiment nodes; the experiment network is separated.

The **experiment nodes** in our architecture are realized as VMs. These nodes can be configured to match the target hardware of a real testbed. VM hypervisors allow setting the number of CPU cores, the amount of storage or RAM, or the number of NICs available to an individual experiment node. It is also possible to create node-specific settings. Sharing hardware resources such as RAM or CPU cores between the experiment node VMs is possible. Though these techniques allow affordable scaling, the shared resources may become overloaded, causing performance impacts that would not be present on physically separated experiment nodes. Therefore, the number of experiment nodes and their assigned resources must be carefully configured to avoid impacting experiment results caused by virtualization artifacts.

The **experiment network** relies on dedicated, physical NICs exclusively used for experimental traffic. These NICs are

split into several VFs that are attached to the experiment nodes. The actual NIC itself is represented by a physical function (PF) that remains attached to the host machine. For security reasons, certain settings such as the isolation between different VFs or rate limits can only be set by the owner of the PF. This way, only the host machine can perform changes. If necessary, the PFs can be attached to the controller VM to enable a VF reconfiguration using the testbed controller. Experimental traffic exclusively uses hardware-accelerated networks. Figure 1 shows a possible configuration of NICs and VFs. Each of the four experiment nodes is attached to a specific VF. The VFs are distributed to two different single-port NICs; the two NIC ports are connected to each other via a physical link. In Figure 1, we use a configuration that isolates the network traffic handled by VFs on the same NIC from each other. This can be done by assigning two different VLAN IDs, VLANs A and B, to the two VFs. VLAN tagging and untagging can be handled transparently in the NIC, i.e., VLAN tags and handling are entirely hidden from the experiment nodes. The same VLAN configuration can also be realized on the second NIC, thereby creating two VLAN-isolated connections across both NICs and the connected experiment nodes, respectively. Without VLAN isolation, all experiment nodes connected to the same NIC would also see each other using the switch integrated into SR-IOV NICs. Using VLAN isolation, we force the traffic to use the wired connection, thereby we ensure the usage of real hardware, thus creating realistic performance.

A more detailed description of such a VLAN-based network, relying on SR-IOV and VLAN isolation, was presented by Wiedner et al. [4]. They also provide measurements demonstrating the performance achievable in such a scenario.

IV. Prototype Implementation

To demonstrate the feasibility of the proposed architecture, we created a prototype implementation. Our implementation uses commercial off-the-shelf server hardware: the *Supermicro X10SDV-7TP4F* mainboard, featuring an *Intel Xeon D-1537* (8 cores, 1.7 GHz), 128 GB RAM, and a quad-port *Intel X710-DA4* NIC (10 Gbit/s per port). We use *Debian bullseye* (*Linux* kernel v5.10) on the host machine and the controller VM, utilizing *KVM* as a hypervisor and *libvirt* to create and manage VMs. Our setup relies on the *pos* testbed controller [9]; this controller allows fully automated, reproducible experiments and simplifies the release of experimental scripts and data.

We use a *GitLab* instance as an authentication provider. *GitLab* allows the creation of user accounts and groups, which we use as an access control mechanism for the testbed. Authentication relies on the standardized *OpenID* API [13], which allows a transition to other authentication providers with little additional effort. *OpenID* is an authentication service offered by numerous providers; the *OpenID* foundation lists certified providers [14].

pos was initially designed to manage physical servers. To reboot physical servers, *pos* uses the Intelligent Platform Management Interface (IPMI), a standardized out-of-band management API for servers. *VirtualBMC* [15] is a software implementation of IPMI that allows the remote management of VMs. We use this implementation to avoid any changes to our testbed controller. The network boot process and SSH access to the experiment VMs also remains unchanged compared to a hardware testbed.

Our prototype currently offers four experiment node VMs with 8 GB vRAM and four CPU cores. The configuration of the VMs is currently static, i.e., regular testbed users cannot change the VM resources. We decided to rely on a static configuration with reasonable resource allocation to ensure a stable testbed operation. At the same time, we limit the performance impact of resource sharing when performing experiments.

Like the static configuration for the experiment node VMs, our prototype only uses a static configuration for the experiment networks. The current configuration uses two ports of the *Intel X710-DA4*. Both ports are configured to provide four VFs. Each VM is attached to two VFs, one from the first and one from the second port. Our configuration does not use VLAN isolation. This means that VMs can either connect via the same NIC port, or via the remote NIC port. We intentionally disabled VLAN isolation to provide a highly flexible network config, without providing access to the network configuration on the host machine. In this case, the VFs act as the ports of a switch connecting the machines to each other.

A. Limitations of the Prototype

We are aware that the static configuration of the experiment nodes and the experiment network severely limits the capabilities of our testbed. Freely configurable network topologies or VLAN isolation are one of the key features of *toast*. The same also holds for the configurability of the VM, where our restrictions may unnecessarily limit the scalability of the experiment nodes. However, a wrong network configuration impacting the management vswitch in Figure 1 may disconnect experiment nodes. In addition, overloading the hardware of the physical system may lead to an unstable controller. Wrong configurations of the network or VM resources may compromise the entire testbed's stability.

To allow more flexible experiments for future prototypes, we plan to extend the configurability of VM resources and network topologies in future work. Additional work is needed to identify problematic configurations and provide countermeasures to ensure the stability and reachability of the one-server testbed in the presence of faulty configuration.

B. Availability of the Prototype

Our implementation is currently relying on hardware-specific settings, e.g., the configuration of SR-IOV on the used *Intel X710* NIC. Though the SR-IOV feature is not limited to this specific card or its vendor, configuration differs between *Intel* NICs and even more compared to NICs of other vendors. Therefore, we do not provide a solution that supports other NICs or hardware configurations. However, we provide web-

Fig. 2: Measurement setup

access to our prototype that we host on our infrastructure. Section VI explains how the *toast* testbed can be accessed.

V. Case Study: Performance

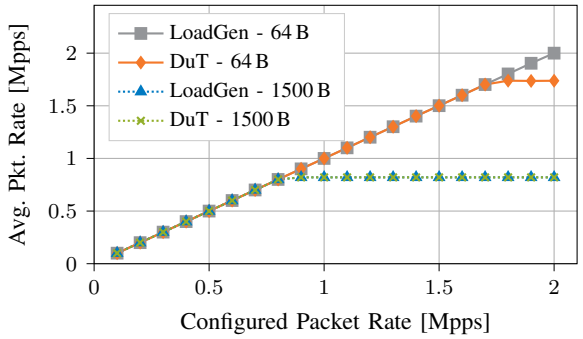
In this section, we want to compare the *toast* approach to two different testbeds that we operate, a hardware-based testbed (*pos*) and a fully virtualized testbed (*vpos*). The results for *pos* and *vpos* were already part of their original publication [9]. There, a two-server two-link topology (see Figure 2) is used. The first node generates packets using *MoonGen* [16] (LoadGen). The second node acts as device under test (DuT) and uses the *Linux* router to forward packets between its two network interfaces. We aim to measure the maximum loss-free throughput that the DuT can handle for packet sizes of 64 B and 1500 B. The *pos* and *vpos* experiments use *Debian buster* (*Linux* kernel v4.19) on a system with two *Intel Xeon Silver 4214* CPUs (12 cores, 2.2 GHz) and an *Intel 82599* dual-port NIC (10 Gbit/s per port). The *toast* experiment uses the software and hardware specified in Section IV. Both setups use the same version of the testbed controller and the same version of the experiment script [17].

Experiment results are plotted in Figure 3. For the three cases (hardware-based *pos*, fully virtualized *vpos*, and hardware-accelerated virtualized *toast*), the generation and the forwarding of the DuT increase linearly up to the maximum threshold the DuT can handle. After that, the forwarding rate remains at a certain level. This maximum rate differs significantly between the investigated setups.

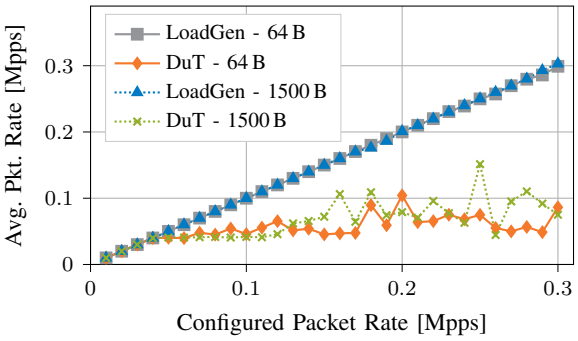
Subfigure 3a shows the results of the hardware-based testbed (*pos*). For a packet size of 64 B, we measure a maximum throughput of 1.7 Mpps. The larger 1500 B packets hit the bandwidth limit of our 10 Gbit/s links at a packet rate of 0.82 Mpps. At line rate, generation and forwarding rate remain at a constant level of 0.82 Mpps. We observe highly stable forwarding rates for both investigated packet sizes.

In Subfigure 3b, the results of the *vpos* measurements are plotted. This setup relies on fully virtualized software switches for packet IO. There, the packet rates are significantly lower compared to any other setup. We already observed a packet loss starting at 50 kpps, independent of the investigated packet size. In addition, throughput rates beyond 50 kpps become unstable.

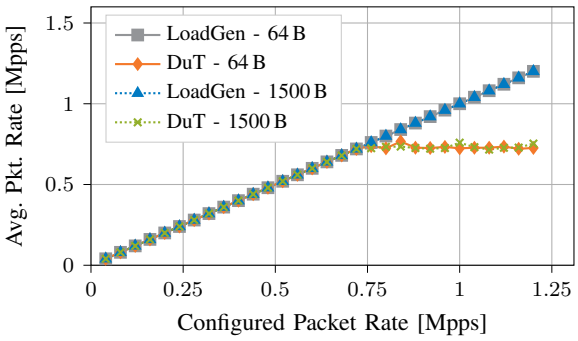
The *toast* results are plotted in Figure 3c. In this setup, we observe the same maximum forwarding throughput of approx. 0.72 Mpps for both investigated packet sizes. Using the VFs of the *X710* NIC, we could also surpass the 10 Gbit/s limit. This is possible because the *X710* NIC uses a controller that supports 40 Gbit/s Ethernet like the VFs. When comparing the forwarding results to *vpos*, we see significantly higher forwarding rates and more stable rates for *toast*. Compared to the *pos* setup, forwarding rates are lower. Please mind the



(a) Hardware-based testbed (*pos*)



(b) Virtual testbed (*vpos*)



(c) Virtual testbed with accelerated IO (*toast*)

Fig. 3: Measurement results for the generated (*MoonGen* as LoadGen) and forwarded rate (*Linux* router as DuT).

different axes scaling between Figures 3a and 3c. The lower rates can be partially attributed to the hardware used by *toast*, using a less powerful CPU running an older architecture and a 500 MHz lower clock rate. Despite its difference in throughput, the stability of the *toast* setup is a better approximation of the real hardware of the *pos* testbed than the *vpos* testbed before. This means that the hardware-accelerated virtualization of *toast* offers a more realistic approximation of the original *pos* results than the purely software-based IO solutions used in *vpos*.

VI. Reproducibility

We created a website [17] that contains the experiment scripts, experimental data, plotting scripts, and plots. Interested users can use the available web instance at

https://testtestbed.net.in.tum.de to reexecute our experiments and reproduce our results. There, we offer access to the one-server version of *pos* through a lightweight registration process.

VII. Conclusion

In this paper, we presented *toast*, an architecture that allows the prototyping of entire testbeds. Our architecture splits the testbed into four modules: the access module, the controller, the experiment nodes, and the experiment network. The modules are designed for extensibility and replaceability using flexible APIs. Though we demonstrate the feasibility of *toast* for the testbed controller *pos*, the concept provides enough flexibility to be applicable to other testbeds. Using standardized *OpenID*, the authentication providers can be easily exchanged, a testbed developer can host any testbed controller inside a VM, the number of testbed nodes and the provided resources can be individually configured, and the network topology and bandwidth can be configured using hardware features of NICs.

A comparison between a real hardware-based testbed and a fully virtualized testbed without hardware IO acceleration demonstrates the benefits of *toast*. *toast* combines the flexible topology configuration similar to purely software-based solutions with the realistic performance typically reserved for hardware-based testbeds.

The *toast* approach was created with simplicity and affordability in mind. Therefore, everything is hosted on a single physical server. This makes *toast* a viable solution for teaching in an academic environment, where students can perform experiments on actual server hardware achieving real-world performance. *toast* can also act as a placeholder for a future hardware-based testbed. The training of users, e.g., the previously mentioned students, and the refinement of the testbed can take place even before the actual testbed has been built. This makes the *toast* approach an ideal learning facility for testbed developers and potential users alike.

Acknowledgment

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101008468. Additionally, we received funding from the Bavarian Ministry of Economic Affairs, Regional Development and Energy as part of the project 6G Future Lab Bavaria. This work is partially funded by the German Federal Ministry of Education and Research (BMBF) under the project 6G-life (grant number 16KISK001K).

References

[1] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible Network Experiments Using Container-Based Emulation," in *Conference on emerging Networking Experiments and Technologies, CoNEXT '12, Nice, France - December 10 - 13, 2012*, C. Barakat, R. Teixeira, K. K. Ramakrishnan, and P. Thiran, Eds. ACM, 2012, pp. 253–264. [Online]. Available: https://doi.org/10.1145/2413176.2413206

[2] *OMNeT++*: Discrete event simulator. Last accessed: 2022-04-10. [Online]. Available: https://omnetpp.org/

[3] *ns-3*: Discrete event network simulator for internet systems. Last accessed: 2022-04-10. [Online]. Available: https://www.nsnam.org/

[4] F. Wiedner, M. Helm, S. Gallenmüller, and G. Carle, "HVNet: Hardware-Assisted Virtual Networking on a Single Physical Host," in *IEEE INFOCOM WKSHPS: Computer and Networking Experimental Research using Testbeds (CNERT 2022) (INFOCOM WKSHPS CNERT 2022)*, May 2022.

[5] *NetEm*: Network emulator to modify a link's performance properties. Last accessed: 2022-04-10. [Online]. Available: https://wiki.linuxfoundation.org/networking/netem

[6] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proceedings of the 9th ACM Workshop on Hot Topics in Networks. HotNets 2010, Monterey, CA, USA - October 20 - 21, 2010*, G. G. Xie, R. Beverly, R. T. Morris, and B. Davie, Eds. ACM, 2010, p. 19. [Online]. Available: https://doi.org/10.1145/1868447.1868466

[7] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. S. Chase, and D. Becker, "Scalability and Accuracy in a Large-Scale Network Emulator," in *5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, Massachusetts, USA, December 9-11, 2002*, D. E. Culler and P. Druschel, Eds. USENIX Association, 2002. [Online]. Available: http://www.usenix.org/events/osdi02/tech/vahdat.html

[8] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," in *5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, Massachusetts, USA, December 9-11, 2002*, D. E. Culler and P. Druschel, Eds. USENIX Association, 2002. [Online]. Available: http://www.usenix.org/events/osdi02/tech/white.html

[9] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle, "The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments," in *CoNEXT '21: The 17th International Conference on emerging Networking EXperiments and Technologies, Virtual Event, Munich, Germany, December 7 - 10, 2021*. ACM, 2021, pp. 259–266. [Online]. Available: https://doi.org/10.1145/3485983.3494841

[10] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale Virtualization in the Emulab Network Testbed," in *2008 USENIX Annual Technical Conference, Boston, MA, USA, June 22-27, 2008. Proceedings*, R. Isaacs and Y. Zhou, Eds. USENIX Association, 2008, pp. 113–128. [Online]. Available: http://www.usenix.org/events/usenix08/tech/fullpapers/hibler/hibler.pdf

[11] P. Emmerich, D. Raumer, S. Gallenmüller, F. Wohlfart, and G. Carle, "Throughput and Latency of Virtual Switching with Open vSwitch: A Quantitative Analysis," *J. Netw. Syst. Manag.*, vol. 26, no. 2, pp. 314–338, 2018. [Online]. Available: https://doi.org/10.1007/s10922-017-9417-0

[12] G. Lettieri, V. Maffione, and L. Rizzo, "A Survey of Fast Packet I/O Technologies for Network Function Virtualization," in *High Performance Computing - ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P^3MA, VHPC, Visualization at Scale, WOPSSS, Frankfurt, Germany, June 18-22, 2017, Revised Selected Papers*, 2017, pp. 579–590. [Online]. Available: https://doi.org/10.1007/978-3-319-67630-2_40

[13] GitLab as OpenID Connect identity provider. Last accessed: 2022-04-10. [Online]. Available: https://docs.gitlab.com/ee/integration/openid_connect_provider.html

[14] OpenID Certification. Last accessed: 2022-04-10. [Online]. Available: https://openid.net/certification/

[15] Virtualbmc repository. Last accessed: 2022-04-10. [Online]. Available: https://github.com/openstack/virtualbmc

[16] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, K. Cho, K. Fukuda, V. S. Pai, and N. Spring, Eds. ACM, 2015, pp. 275–287. [Online]. Available: https://doi.org/10.1145/2815675.2815692

[17] S. Gallenmüller, E. Hauser, and G. Carle. Experiment Results and Replication. Last accessed: 2022-04-10. [Online]. Available: https://gallenmu.github.io/single-server-experiment