The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments

Sebastian Gallenmüller*, Dominik Scholz*, Henning Stubbe, Georg Carle
Department of Informatics, Technical University of Munich, Germany
{gallenmu,scholz,stubbe,carle}@net.in.tum.de

ABSTRACT

In scientific research, the independent reproduction of experimental results is *the* source of trust. The release of experimental artifacts enables the reproduction of results; however, additional efforts of researchers are required to prepare and document their experiments accordingly. To honor this increased effort, multiple initiatives were implemented to incentivize the creation and release of experimental artifacts, e.g., awards for papers that provide experimental artifacts.

In this work, we want to propose a novel approach toward a reproducible research—a structured experimental workflow that allows the creation of reproducible experiments without requiring additional efforts of the researcher. Moreover, we present our own testbed and toolchain, namely, plain orchestrating service (pos), which enables the creation of such experimental workflows. The experiment is documented by our proposed, fully scripted experiment structure. Further, we consider the entire experimental workflow from experiment orchestration, to data measurement, to result evaluation. In addition, pos provides scripts that enable the automation of the bundling and release of all the created experimental artifacts. In this case study, we release one of our own experiments together with the necessary tools so that others can reproduce our experiment. Additionally, we provide an interactive environment where pos experiments can be executed and reproduced, which is available at https://gallenmu.github.io/pos-artifacts.

CCS CONCEPTS

• **Networks** → **Network experimentation**; • **General and reference** → *Experimentation.*

KEYWORDS

Repeatability, Reproducibility, Replicability, Experiments

ACM Reference Format:
Sebastian Gallenmüller, Dominik Scholz, Henning Stubbe, Georg Carle. 2021. The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments. In *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21), December 7–10, 2021, Virtual Event, Germany*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3485983.3494841

*Joint first authorship

CoNEXT '21, December 7–10, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-9098-9/21/12…$15.00
https://doi.org/10.1145/3485983.3494841

1 INTRODUCTION

Re*search* is ultimately referred to as the search for truth. Typically, the scientific truth cannot be revealed through a single measurement or observation, but through experimental validation of others who independently recreated their own measurements, verifying the original results. In computer science, this essential recreation is often neglected due to various reasons, such as low appreciation of the replicated results in the scientific community or the requirement of an additional effort of researchers to adequately prepare experimental data. To foster the recreation of experimental results, the ACM [6] introduced badges—awards for papers that invest the additional time and effort to make their results reproducible. Based on the quality of the provided artifacts, the badges come in different types depending on how and by whom the results can be recreated. The ACM considers three different stages for reproducibility: *repeatability*, the *same* people use the *same* setup to repeat results; *reproducibility*, *different* people use the *same* setup to reproduce results; *replicability*, *different* people use *different* setups to replicate results. Please note that we used the revised version of the ACM's definition released in August 2020. In the revised version, the definitions for reproducibility and replicability are swapped to match the more common definitions of NISO [6].

In the scientific community, many agree with the benefits of reproducibility and its incentivization owing to the increased effort in the preparation and release of artifacts for reproduction [13, 29, 37]. Among the proposed solutions, the common theme is the introduction of incentives as a form of compensation for the increased workload. We aim to approach the problem from a different perspective: reduce the amount of work that researchers have to put into making their experiments reproducible. However, our proposed solution is not a replacement but rather a complement to the current reward-based approaches.

We present a methodology for network experiments that relies on an automated experimental workflow. To implement such a methodology, we created our own testbed and software toolchain, which is called plain orchestrating service (pos). Our testbed depends on fully scripted experiments to document and automate the experimental workflow to enable the experiments to be repeatable. Our methodology further introduces an experimental structure that divides the experiments into setup, experiment, and evaluation phases, in addition to the division between experiment scripts and experiment parameters. This structure facilitates in the documentation of experiments that can be easily shared with others to achieve reproducibility, which requires little additional effort. Using the structured experimental approach and our toolchain, the experiment can be prepared for publication to enable others to replicate the experiments.

Our paper aims to *(i)* introduce a methodology that enables inherently reproducible experiments by following our proposed experimental workflow, *(ii)* establish experimental designs that support the researcher to create publishable artifacts with only little additional effort required, and *(iii)* demonstrate how researchers can easily understand and recreate the results of our sample network experiments. In this paper, we present an example experiment and provide all experimental artifacts, including the experimental results, the scripts that enable their creation, and the evaluation. In addition, we provide access to a virtual instance of our testbed infrastructure, which demonstrates that the provided artifacts can be easily reused by others and easily created and published by experiment creators.

The remainder of this paper is structured as follows. Sec. 2 investigates the state of the art in the area of reproducible network experiments and testbeds. We define our requirements and present our design for a purpose-built testbed in Sec. 3 and 4. A case study demonstrates a network experiment in this testbed (Sec. 5). In Sec. 6, we compare pos to other testbeds and Sec. 8 concludes the paper. Finally, Appendix A documents the experimental workflow for the reproduction of our case study.

2 BACKGROUND AND RELATED WORK

For many years, the SIGCOMM community [12] has been discussing the topic of reproducibility. Here, we discuss recent developments particularly focusing on tools and testbeds that facilitate in the creation of reproducible experiments.

Reproducibility. In 2015, Collberg and Proebsting [13] studied the replicability of computer science publications. They concluded that the lack of replicability is considered to be sociological, not technological, as little reward can be gained from replication. This issue can be solved in the long term only by a cultural change toward the appreciation of replication. However, as a short-term solution, they suggested that the funding agencies should provide funds dedicated to making publications repeatable. More recent developments have suggested that in computer science, reproducibility is gaining considerable attention. In 2017, a workshop focusing on replicable network experiments was held at the SIGCOMM conference [11]. In 2019, a Dagstuhl seminar on this topic was held, in which guidelines for replicable network experiments have been established [8]. These guidelines consider the entire lifecycle of an experiment: from the experiment description and execution, to evaluation and finally the publication of artifacts. The push toward reproducibility is not limited to SIGCOMM. Other scientific communities are also currently establishing a culture of experiment replication, as well as Artifact Evaluation Committees [19]. Recently, Saucez et al. [29] evaluated the Artifact Evaluation Committees of CoNEXT'18 and other SIGCOMM-sponsored conferences and journals. A survey conducted on the authors and reviewers considered the gained experience to be useful and interesting. However, the evaluation can be time-consuming for reviewers. Although considerable attention is paid to reproducibility, achieving it is difficult. Zilberman recently conducted a case study [36], demonstrating that even papers awarded with ACM's reusable badge may not paint a complete picture of the investigated system behavior. She found that low robustness, i.e., small variation from the original input, such as the investigated packet size, could lead to a significantly different performance.

Testbeds. Currently, there are numerous initiatives for maintaining and providing testbeds for distributed computing and networking research, e.g., Fed4Fire (EU) [1], OneLab (EU) [4], Grid'5000 (France) [3], Planetlab (global) [5], or GENI (USA) [2]. The included testbeds specialize in different areas, such as wired networks, 5G, Internet of Things, or Internet-scale measurements. For this paper, we consider testbeds that target a domain similar to pos, i.e., wired networks and nodes that scale across multiple racks and sites. Nussbaum [22] investigated three testbeds (Chameleon [20], CloudLab [28], and Grid'5000 [9]) that are similar to pos. He focused on their ability to execute reproducible network experiments and concluded that the investigated testbeds are indeed capable of performing such experiments. However, the testbeds neither guarantee nor enforce the creation of reproducible experiments. Zhuang et al. [35] evaluated the testbeds Emulab, PlanetLab, Seattle, and GENICloud for teaching. They found several problems, such as involuntary configuration changes during the running experiments and fluctuating bandwidths that may impact the repeatability of the experiments running on these testbeds.

Methodologies. In contrast to testbeds, methodologies introduce concepts to structure and execute experiments. OMF [26] is a testbed controller, which has its own domain-specific language (DSL) to program network experiments. Similar to pos, OMF enables the creation of reproducible experiments relying on automation. The DSL of OMF allows the specification of complex experimental workflows as Petri nets. The pos methodology assumes a simpler experimental workflow, indicating that its API is easier to learn and use. Peuster et al. [24] propose SNDZoo, a repository for reproducible network experiments and a toolchain to reproduce them. Their tools and experiments are focused on containers and VMs. Thanks to the tight integration of methodology and testbed, pos additionally supports low-level hardware experiments. In 2011, Quereilhac et al. [25] presented NEPI, a network experimentation framework that supports various backends such as PlanetLab, netns, or ns-3. pos' methodology goes a step further in experiment design than NEPI, considering subsequent experiment steps such as the evaluation and later publication of data.

pos consists of two entities—its methodology and a testbed implementing this methodology. Both cojointly designed with reproducibility in mind. While other testbeds only offer the possibility of creating repeatable experiments, the pos methodology enforces repeatability. Given the access to a pos-capable testbed and experiment files, others can reproduce the experiments, a property that we call reproducibility by design. By following the experimental workflow of pos, reproducible experiments can be created with the requirement of little additional effort (cf. Sec. A). pos cannot ensure replicability; however, the created experimental artifacts document the experiments to enable other researchers to easily replicate such experiments. The full automation of the experimental workflow further attempts to address the issue of low robustness.

3 REQUIREMENT ANALYSIS

The following analysis introduces the requirements for reproducible network experiments.

Heterogeneity (R1). Networks involve a wide variety of participants, such as embedded, resource-constrained devices, packet processing software on off-the-shelf hardware, smartNICs built to accelerate specific network tasks, and switches with dedicated ASICs. To achieve effective experiments, a testbed must be able to adapt to different devices.

Isolation (R2). Network experiments are inherently distributed across devices, with each device potentially impacting the behavior of other devices and, therefore, the experimental outcome. We aim to observe device behavior only influenced by devices that are part of the experimental network. Therefore, our testbed must provide means to isolate the experimental network from non-investigated devices that may introduce unwanted effects into the experiment.

Recoverability (R3). Experiment-driven science often involves an investigation of objects using the trial-and-error approach. For network experiments, such an approach requires constant alteration and modification of the experimental network and its devices. Such an approach may ultimately cause network malfunction or failure. The testbed must be able to recover from a fully configured, a misconfigured, or even an error state, so that experiments can be restarted or restored from the same, well-defined, initial state.

Automation (R4). Accurately setting up network devices may involve several steps or complex configuration files. A correct setup is crucial to the performance of network devices and subsequently to the recreation of experiments. Full automation makes the setup phase less error-prone and time-consuming than a manual setup.

Publishability (R5). The availability of scripts, results, and evaluation tools is essential to achieve replicability. Well-documented experimental artifacts provide others with the necessary information to replicate experimental results. A testbed should lower the barrier for researchers to publish the experimental artifacts.

4 EXPERIMENT DESIGN

We are not aware of a testbed that adheres to a methodology meeting all identified requirements, inherent to pos' design.

4.1 Experiment Methodology

We define the *network experiment* as the entire process that configures, performs, evaluates, and optionally publishes measurements. Our network experiments are parameterized with *variables* or *vars*. We investigate different instances for the variables during an experiment. We call the execution of one concrete instance *measurement run* or short *run* and the outcome of each measurement run a *result*.

Fig. 1 presents typical entities in a network experiment. The testbed controller manages the entire experimental workflow. The two other roles in this experiment are experiment hosts. One device is the device under test (DuT), which is the object of the investigation for an experiment and the other device is the load generator (LoadGen), which generates the traffic sent to the DuT and receives processed traffic from the DuT. The number of experiment devices can be scaled. Here, we focus on a minimal topology.

4.2 Testbed Implementation

To support the different experiment devices (R1), pos implements two APIs: an initialization and a configuration interface. An example of the former used by pos to reset and boot servers is IPMI.

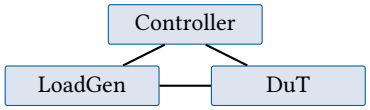


Figure 1: Typical testbed topology

Our testbed controller does not depend on the availability of IPMI: alternatives are other management APIs, such as Intel's vPro or AMD's Pro features, or a remotely switchable power plug that triggers a device reboot. The mentioned APIs allow the initialization of a device to be triggered out of band (R3), i.e., the devices can be reinitialized in the case of configuration errors.

After initialization of an experimental device, the configuration interface is used to configure the device and execute the experiment. For a typical Linux server, we use SSH as the configuration interface. IPMI and SSH are given only as examples; thus, they can be replaced with different protocols, depending on the APIs provided by the experiment hosts. pos supports configuration and initialization APIs for devices via SNMP or HTTP. To support devices requiring other protocols, an implementation for the respective API can be added to pos. The entire initialization process and configuration of a network device is automated via user-defined scripts (R4). To avoid any shared state between the different executions of the experiment, pos relies on live-boot images. Such images enforce repeatability, as the OS repeatedly starts from a well-defined state, and the researcher must automate and thereby document the device configuration (R3 and R4). To prevent any influence of switches or hubs on the observed results (R2), our testbed employs direct wiring between experiment hosts.

Load Generators. One of the essential elements of a network experiment is the used traffic source. Typical experiments in our testbed use synthetic traffic created at runtime by a packet generator [17, 18, 30]. However, other experiments use pcaps of recorded traffic. Most of our experiments use MoonGen [16] owing to its ability to support user-defined scripts to generate packets during runtime or to replay pcaps. Its precision and accuracy for packet generation and latency measurements is superior to other software packet generators [15]. However, pos does not solely depend on MoonGen. Other software packet generators, such as iPerf, can be run on off-the-shelf or even virtualized experiment hosts.

Our flexible testbed architecture also enables the integration of hardware packet generators, such as OSNT. OSNT is based on the NetFPGA platform [7], which can be integrated into experiment hosts as PCIe cards. Packet creation or pcap replay can be managed by experiments through their respective host servers. Hardware packet generators may also come in the form of tightly integrated systems, e.g., Intel's Tofino ASIC built into switches. In that case, the entire device can be added to the testbed as a new experiment host and managed through the provided configuration APIs.

4.3 pos Experimental Structure

To achieve replicability, other researchers must understand the experimental artifacts. Therefore, we enforce a specific structure to program pos experiments.

The user-programmable experiment scripts distinguish two different file types: script and parameter files. This idea is inspired

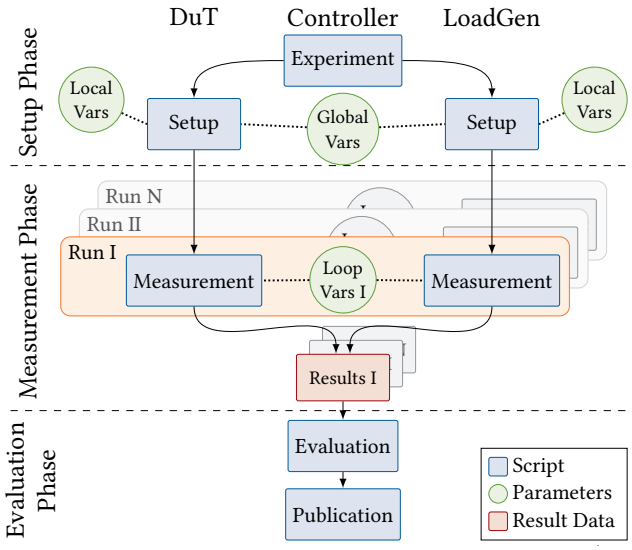


Figure 2: Experimental workflow

by HTML and CSS; HTML defines the structure of a text, CSS its design. In the pos experiment structure, the scripts define the individual steps of the experiment, and the variables define the concrete instance of a measurement run. For instance, a script file defines the initialization of a network port with the name `$PORT`, the variable file assigns `$PORT` the value `eno1`. This separation allows the experiments to be executed in a different setup by merely adapting the variables without the need to change the script files.

To further elucidate the experimental structure, we used different scripts for the different participating experiment hosts and for the different phases of an experiment. Each experiment host requires two exclusive script files: *setup*, which defines the experiment host configuration, and *measurement*, which defines the active phase of a measurement run that generates results. Thereby, a script can be any executable, e.g., python or bash, that can be executed on the target device. The script contains the sequence of commands to execute, e.g., starting or stopping iPerf.

To parameterize the experiment scripts, pos provides three different kinds of variables, depending on which experiment host has access and where the variables are utilized in the experimental workflow: *global vars*, accessible from all experiment hosts; *local vars*, defined for each experiment host; and *loop vars*, shared across all experiment hosts, but continuously changed between different measurement runs.

4.4 pos Experimental Workflow

In Fig. 2, the script, variable, and result files that describe the high-level workflow of an experiment are presented, code examples for each file are available [32]. From top to bottom, the workflow is separated into the three subsequent phases: setup, measurement, and evaluation.

Setup Phase. The testbed controller host executes the main *experiment* script defining the experiment workflow. This script interacts with the pos API to execute multiple actions. First, it allocates the desired devices—the workflow presented in Fig. 2 allocates two devices: the DuT and LoadGen. As we operate a multiuser testbed, we use an integrated calendar to temporally separate the experimental devices between users. Only if the calendar indicates that the devices are free for the planned duration of the experiment, the allocation can be created. This allows sharing testbed nodes between all users and running multiple independent experiments in parallel. Further, using a node in more than one experiment at the same time is prohibited.

Afterwards, the devices are configured by loading the global and loop variables. The local variables are loaded for each experiment host. Moreover, a live image is selected for every device, and boot parameters can be set. For reproducible boots, we use live images and start every experiment using the same clean slate (R3). Utilizing the Debian snapshot project [23], we can create live images with specific version numbers for the kernel and the installed packages. Finally, the experiment script instructs pos to start the devices, whereby the boot is internally executed using the respective initialization interface. This abstraction improves the usability for users, as they do not need to take care of boot specifics (R1) but can still define experiment-specific boot parameters, e.g., for the Linux kernel. Once the experiment hosts have finished booting, pos deploys a set of utility tools before the setup scripts can be loaded and executed to complete the setup phase.

These tools can be used in the setup or measurement scripts; read or communicate variables and synchronize hosts using barriers. Further, any command can be executed via pos' tools. The output of these commands is automatically captured and uploaded to the testbed controller as a result.

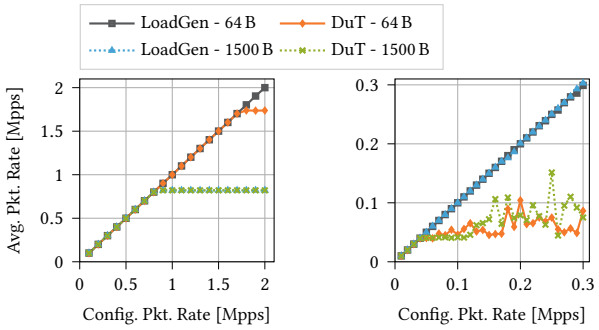
Measurement Phase. During the measurement runs, pos executes the measurement script for every node. The number of executions depends on the number of individual parameters contained in the loop variables file. Each of these parameters can represent either a single value or a list of values. pos experiments perform measurements for each possible combination of loop parameters. If lists are used as parameters, pos automatically generates the cross product over all parameter values to ensure full coverage. For every set of values contained in the calculated cross product, it executes the measurement script once. Parameters must be carefully chosen, as the exponential growth in the measurement runs may cause infeasibly long experiment completion times.

pos automatically queues one run after another, starting the next only after the current run has been completed. The complete output of the experiment script is captured and stored in the result folder of the experiment. This enforced central collection of artifacts, including the output of the utility tools, executed scripts, variables, device hardware and topology information, guarantees publishability (R5).

Evaluation Phase. The evaluation script processes the result files either after all runs have been completed or asynchronously during their runtime. Due to the enforced structure of experiments, pos creates separate result files for each measurement run. Additionally, pos creates metadata for each run, i.e., the loop parameters of a specific run. Based on this metadata, the evaluation script can filter or aggregate specific parameters and values. We integrated a parser for MoonGen's output into our plotting scripts. The MoonGen output, in conjunction with the available metadata, allows

the automated evaluation of experiments. Our plotting scripts can create throughput figures and latency distributions out-of-the-box using a set of different representations (line plot, histogram, CDF, HDR, and violin plot). The generated plots are exported to multiple formats, e.g., tex, svg, and pdf. Researchers can add their own parsers to support other packet generators or output formats. In addition, they can adapt or extend the evaluation script to reflect the concrete experiment setting and to create custom graphs.

Our structured experimental workflow allows all artifacts linked to an experiment to be connected, i.e., executed scripts, generated results, and created plots. The *publication* script bundles these artifacts into a release format, e.g., an archive or a repository. In addition, it generates a website and inserts all the collected artifacts documenting the experimental structure in a format that can be easily read by researchers.



(a) Hardware testbed (pos) **(b) Virtual testbed (vpos)**

Figure 3: Generated (LoadGen) vs. forwarded rate (DuT)

5 CASE STUDY

We demonstrate the capabilities of pos using one of our own experiments. In Appendix A of this paper, we included a detailed description on how to replicate this experiment, including the access to a virtual instance of the testbed and the repository containing the experimental artifacts.

Setup. We use MoonGen as the load generator to measure the forwarding performance of our DuT, the Linux router, for packets with different sizes (64 and 1500 B). We chose the Linux router to demonstrate our experiment workflow, as software routers are typical targets for network experiments [10, 14, 27]. We perform a throughput measurement on two different platforms: pos, which uses real hardware in our testbed and vpos, which is a virtual clone of the testbed.

The DuT in the hardware testbed runs Debian Buster (kernel v4.19) on a system with two Intel Xeon Silver 4214 CPUs (12 cores, 2.2 GHz) and an Intel 82599 NIC (dual port, 10 Gbit/s) that is directly connected to the load generator. The Linux router forwards the traffic between its two network ports to the hardware NIC.

The virtual testbed runs on the hardware and OS of the previously described DuT, using KVM as a hypervisor. The VMs running the experiment are pinned to fixed CPU cores to avoid unwanted resource sharing. We use Linux bridges for the connection between the experiment VMs.

Evaluation. The performance results of the experiments naturally differ between pos and vpos. In Fig. 3a, a maximum forwarding performance of 1.75 Mpps (for 64 B packets) and 0.8 Mpps (for 1500 B packets) for the bare-metal Linux router (DuT) is presented. The lower throughput for the larger packet size is caused by the 10 Gbit/s limit of the used NIC. The virtualized Linux router (cf. Fig. 3b) forwards packets without drops at a maximum rate of 0.04 Mpps, regardless of the packet size. Beyond 0.04 Mpps, the forwarding performance becomes unstable if the system is overloaded, which is evident in the throughput differences between the packet sizes. The generation performance is stable between the two setups for the investigated packet rates.

With a decrease in the maximum forwarding throughput by a factor of up to 44 and an increase in variance in the virtualized environment, how can both setups be compared? While the raw performance figures cannot be compared, the underlying tendencies stay the same. We can still see that the number of processed packets affects the drop-free forwarding performance, but not the packet size. For both platforms, bare-metal and VM, the measured maximum throughput is forwarded regardless of the packet size, as long as no bandwidth limits are hit, e.g., Ethernet bandwidth limit.

Another essential feature of our experimental workflow is that the underlying experiment scripts, result file format, and subsequent processing scripts are the same for both setups. The experiment scripts can be developed in the virtual environment. Subsequently, the experiment can be executed on real hardware with no or minimal changes to the experiment script. In addition, the behavior of the virtual testbed is close to the real system; thus, the fundamental tendencies may already be visible in the virtualized setup. We do not assert that this highly similar behavior works for all kinds of network experiments. However, even in this case, the virtual testbed may come in handy as a debugging facility, without the need to occupy experiment nodes in the real testbed.

6 TESTBED VS. METHODOLOGY

For the following comparison we have to consider the two main aspects, our proposed methodology and the testbed that realizes our methodology, separately.

Our testbed offers researchers a high degree of freedom to design and execute experiments, with user-controlled scripts and root access on the experiment nodes. We used pos in the past for entirely different experiments: distributed network experiments involving 15 nodes [34], low-latency analysis of single network nodes [18], and as a mere development environment for packet processing software [16]. As a *testbed*, the capabilities of pos neither stand out nor back compared with its main competitors [9, 20, 28]. Through pos' well defined interfaces, support for new devices can easily be added. While we prefer directly wired nodes, pos also allows adding remote nodes to the testbed, e.g., via VLANs or VPNs.

Our methodology describes a well-structured experiment workflow that provides attractive features for network experiments: First, following the pos methodology, experiments can be reproduced on different platforms with minor additional effort. Second, the pos experiment result files can be prepared for publication [31] with minimal overhead for the researcher. Though researchers *may*

	Testbed			Methodology	
	Heterog. (R1)	Isolat. (R2)	Recover. (R3)	Autom. (R4)	Publish. (R5)
Chameleon [20]	✓	○	✓	n.a.	n.a.
CloudLab [28]	✓	○	✓	n.a.	n.a.
Grid'5000 [9]	✓	○	✓	n.a.	n.a.
OMF [26]	n.a.	n.a.	n.a.	✓	×
NEPI [25]	n.a.	n.a.	n.a.	✓	×
SNDZoo [24]	n.a.	n.a.	n.a.	✓	○
pos	✓	✓	✓	✓	✓

✓ fully supported ○ partially supported × not supported

Table 1: Comparison between testbeds

create reproducible experiments on other testbeds [22], pos' measurement methodology, *by design*, creates reproducible experiments with publishable results. The preparation of easily publishable results minimizes the effort for the researcher and the release of experimental artifacts enables others to replicate the experiment. The clear structure of our experiments can also serve as documentation so that experiments can be easily understood by people who are familiar with the high-level concept of our workflow. The intentionally simple two-node setup of our case study, illustrates the unique strengths of pos as a *methodology*.

Our comparison between pos and other testbeds and methodologies in Table 1 is based on the established requirements (cf. Sec. 3). We split our comparison into testbed-related (R1–R3) and methodology-related requirements (R4–R5). Compared to other *testbeds*, pos offers similar features regarding support for heterogeneous hard- and software (R1) and recoverability from failures (R3). There is a difference in the experiment isolation (R2), where pos provides a stronger isolation due to directly wired, non-switched connections between experiment nodes. However, large-scale experiments may prefer the other testbeds that offer more flexible topologies on switched networks. All *methodologies*, including pos, support experiment automation (R4). However, we noticed differences concerning publishability (R5). In OMF or NEPI, the evaluation of experiments is not part of the experiment workflow. SNDZoo also considers the evaluation part of its workflow, however, pos provides basic, auto-generated throughput and latency plots. In addition pos creates a website listing all experiment artifacts.

Due to pos' methodology the experiments follow a clearly structured workflow resulting in highly structured output files garnished with metadata. The structured workflow documents the experiment, supports evaluation, and simplifies the publication, finally fostering third-party replication. We created the pos testbed as a specialized tool to perform experiments requiring low-level, directly wired access to our DuTs. However, its methodology is not limited to this specific testbed or our preferred two-node setup. Though created in unity, the pos testbed and its methodology are not inseparable. Given a suitable environment (cf. Sec. 3), the methodology may be implemented on other testbeds.

7 LIMITATIONS

The following limitations only apply to the testbed not to its methodology. Network experiments depend on the topology of the investigated network. Typically, we are interested in measuring the behavior of our DuT without any impact on other interconnecting devices, such as switches or routers. Therefore, we use networks with direct non-switched connections. Direct connections have the disadvantage that topologies cannot be automatically created or recreated. Moreover, it requires the researcher to physically wire the network topology. There exist optical L1 switches that allow the fibers to be optically linked, which add a constant delay offset due to the internal wiring of the switch. The impact of such a switch on forwarding delay is lower than 15 ns [21], which is significantly lower than that of an L2 cut-through switch, which is approximately 300 ns [33]. Such a setup allows the automation of the topology with a predictable, low impact on delay. However, our testbed is not equipped with such an optical switch due to its high cost.

The recreation of results is currently limited to configurations accessible from the OS. However, there may be configurations that influence the packet processing performance, such as BIOS settings or NIC firmware. Setting these configurations via pos would be possible. However, BIOS configurations or flashing firmware differs across different manufacturers. Currently, due to the lack of standardized interfaces, pos does not support automated configurations.

8 CONCLUSION

Numerous researchers have proposed ways to embed and foster the spirit of reproducibility in our scientific community through different measures, such as badging, awarding of replicability prizes, or merely allowing appendices that explain experiment replication. We propose a fundamentally different approach: rather than increasing the incentives for researchers to make their experiments reproducible, we reduce the amount of effort that they have to invest in making their experiments reproducible.

Despite the different approach, our methodology does not replace the incentives for replicable research but complements them. We perceive our methodology as an additional cobblestone besides the incentives in the street toward a community where experiment replication is the rule rather than the exception.

We operate a virtual testbed as a service to enable other researchers to try out pos in their browsers. We do not require researchers to set up their own instance of pos. Instead, we provide them browser access to a virtual instance, which we call vpos. A significant advantage of our experimental workflow is that the virtualized experiments can be executed on any pos-driven testbed. Scientists can register experiments at https://virtualtestbed.net.in.tum.de, so their experiments can be executed on pos based on real hardware. Recreating the vpos experiment on pos, further demonstrates that this experiment can be reproduced successfully.

ACKNOWLEDGMENTS

The German Research Foundation (CA595/11-1), and the German-French Academy for the Industry of the Future, supported this work. This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101008468 (SLICES SC). We thank our colleagues Florian Wohlfart and Daniel Raumer, their work was fundamental to the creation of pos.

REFERENCES

[1] [n.d.]. Fed4Fire. https://www.fed4fire.eu Last accessed: 2021-10-24.

[2] [n.d.]. Geni. https://www.geni.net/ Last accessed: 2021-10-24.

[3] [n.d.]. Grid'5000. https://www.grid5000.fr Last accessed: 2021-10-24.

[4] [n.d.]. OneLab. https://onelab.eu/ Last accessed: 2021-10-24.

[5] [n.d.]. Planetlab. https://www.planet-lab.org/ Last accessed: 2021-10-24.

[6] ACM. 2020. Artifact Review and Badging Version 1.1. https://www.acm.org/publications/policies/artifact-review-and-badging-current Last accessed: 2021-10-24.

[7] Gianni Antichi, Muhammad Shahbaz, Yilong Geng, Noa Zilberman, G. Adam Covington, Marc Bruyere, Nick McKeown, Nick Feamster, Bob Felderman, Michaela Blott, Andrew W. Moore, and Philippe Owezarski. 2014. OSNT: open source network tester. *IEEE Netw.* 28, 5 (2014), 6–12. https://doi.org/10.1109/MNET.2014.6915433

[8] Vaibhav Bajpai, Anna Brunström, Anja Feldmann, Wolfgang Kellerer, Aiko Pras, Henning Schulzrinne, Georgios Smaragdakis, Matthias Wählisch, and Klaus Wehrle. 2019. The Dagstuhl beginners guide to reproducibility for experimental networking research. *Comput. Commun. Rev.* 49, 1 (2019), 24–30. https://doi.org/10.1145/3314212.3314217

[9] Daniel Balouek, Alexandra Carpen-Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. 2012. Adding Virtualization Capabilities to the Grid'5000 Testbed. In *Cloud Computing and Services Science - Second International Conference, CLOSER 2012, Porto, Portugal, April 18-21, 2012. Revised Selected Papers (Communications in Computer and Information Science, Vol. 367)*, Ivan I. Ivanov, Marten van Sinderen, Frank Leymann, and Tony Shan (Eds.). Springer, 3–20. https://doi.org/10.1007/978-3-319-04519-1_1

[10] Raffaele Bolla and Roberto Bruschi. 2007. Linux Software Router: Data Plane Optimization and Performance Evaluation. *J. Networks* 2, 3 (2007), 6–17. https://doi.org/10.4304/jnw.2.3.6-17

[11] Olivier Bonaventure, Luigi Iannone, and Damien Saucezi (Eds.). 2017. Reproducibility '17: Proceedings of the Reproducibility Workshop. (2017).

[12] Georg Carle, Hartmut Ritter, and Klaus Wehrle (Eds.). 2003. Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research. (2003).

[13] Christian S. Collberg and Todd A. Proebsting. 2016. Repeatability in Computer Systems Research. *Commun. ACM* 59, 3 (2016), 62–69. https://doi.org/10.1145/2812803

[14] Mihai Dobrescu, Norbert Egi, Katerina J. Argyraki, Byung-Gon Chun, Kevin R. Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. 2009. RouteBricks: exploiting parallelism to scale software routers. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009*, Jeanna Neefe Matthews and Thomas E. Anderson (Eds.). ACM, 15–28. https://doi.org/10.1145/1629575.1629578

[15] Paul Emmerich, Sebastian Gallenmüller, Gianni Antichi, Andrew W. Moore, and Georg Carle. 2017. Mind the Gap - A Comparison of Software Packet Generators. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2017, Beijing, China, May 18-19, 2017*. IEEE Computer Society, 191–203.

[16] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. MoonGen: A Scriptable High-Speed Packet Generator. In *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, Kenjiro Cho, Kensuke Fukuda, Vivek S. Pai, and Neil Spring (Eds.). ACM, 275–287. https://doi.org/10.1145/2815675.2815692

[17] Paul Emmerich, Daniel Raumer, Sebastian Gallenmüller, Florian Wohlfart, and Georg Carle. 2018. Throughput and Latency of Virtual Switching with Open vSwitch: A Quantitative Analysis. *J. Netw. Syst. Manag.* 26, 2 (2018), 314–338.

[18] Sebastian Gallenmüller, Johannes Naab, Iris Adam, and Georg Carle. 2020. 5G QoS: Impact of Security Functions on Latency. In *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*. IEEE, 1–9. https://doi.org/10.1109/NOMS47738.2020.9110422

[19] Matthias Hauswirth. 2020. Evaluate Collaboratory - Artifact Evaluation. http://evaluate.inf.usi.ch/artifacts Last accessed: 2021-10-24.

[20] Joe Mambretti, Jim Hao Chen, and Fei Yeh. 2015. Next Generation Clouds, the Chameleon Cloud Testbed, and Software Defined Networking (SDN). In *2015 International Conference on Cloud Computing Research and Innovation, ICCCRI 2015, Singapore, Singapore, October 26-27, 2015*. IEEE Computer Society, 73–79. https://doi.org/10.1109/ICCCRI.2015.10

[21] Molex. [n.d.]. Molex PXC systems. http://www.oplink.com/uploads/files/520cea9a8ecbcceb5d156a1be86b02a1.pdf Last accessed: 2021-10-24.

[22] Lucas Nussbaum. 2017. Testbeds Support for Reproducible Research. In *Proceedings of the Reproducibility Workshop, Reproducibility@SIGCOMM 2017, Los Angeles, CA, USA, August 25, 2017*. ACM, 24–26. https://doi.org/10.1145/3097766.3097773

[23] Peter Palfrader. 2021. snapshot.debian.org. https://snapshot.debian.org/ Last accessed: 2021-10-24.

[24] Manuel Peuster, Stefan Schneider, and Holger Karl. 2019. The Softwarised Network Data Zoo. In *15th International Conference on Network and Service Management, CNSM 2019, Halifax, NS, Canada, October 21-25, 2019*, Hanan Lutfiyya, Yixin Diao, A. Nur Zincir-Heywood, Rémi Badonnel, and Edmundo R. M. Madeira (Eds.). IEEE, 1–5. https://doi.org/10.23919/CNSM46954.2019.9012740

[25] Alina Quereilhac, Mathieu Lacage, Claudio Daniel Freire, Thierry Turletti, and Walid Dabbous. 2011. NEPI: An integration framework for Network Experimentation. In *19th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2011, Split, Croatia, September 15-17, 2011*. IEEE, 1–5. https://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6064394

[26] Thierry Rakotoarivelo, Guillaume Jourjon, and Max Ott. 2014. Designing and orchestrating reproducible experiments on federated networking testbeds. *Comput. Networks* 63 (2014), 173–187. https://doi.org/10.1016/j.bjp.2013.12.033

[27] Daniel Raumer, Florian Wohlfart, Dominik Scholz, and Georg Carle. 2015. Performance Exploration of Software-based Packet Processing Systems. In *Proceedings of Leistungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen, 6. GI/ITG-Workshop MMBnet 2015*. Hamburg, Germany.

[28] Robert Ricci, Eric Eide, and CloudLab Team. 2014. Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications. *login Usenix Mag.* 39, 6 (2014). https://www.usenix.org/publications/login/dec14/ricci

[29] Damien Saucez, Luigi Iannone, and Olivier Bonaventure. 2019. Evaluating the artifacts of SIGCOMM papers. *Comput. Commun. Rev.* 49, 2 (2019), 44–47. https://doi.org/10.1145/3336937.3336944

[30] Dominik Scholz, Sebastian Gallenmüller, Henning Stubbe, and Georg Carle. 2020. SYN Flood Defense in Programmable Data Planes. In *EuroP4@CoNEXT 2020: Proceedings of the 3rd P4 Workshop in Europe, Barcelona, Spain, December 1, 2020*. ACM, 13–20.

[31] Sebastian Gallenmüller and Dominik Scholz and Henning Stubbe and Georg Carle. 2021. pos Experiment Results and Reproduction (Repository). https://github.com/gallenmu/pos-artifacts Last accessed: 2021-10-24.

[32] Sebastian Gallenmüller and Dominik Scholz and Henning Stubbe and Georg Carle. 2021. pos Experiment Results and Reproduction (Website). https://gallenmu.github.io/pos-artifacts/ Last accessed: 2021-10-24.

[33] Omer S. Sella, Andrew W. Moore, and Noa Zilberman. 2018. FEC Killed The Cut-Through Switch. In *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies, NEAT@SIGCOMM 2018, Budapest, Hungary, August 20, 2018*. 15–20. https://doi.org/10.1145/3229574.3229577

[34] Marcel von Maltitz and Georg Carle. 2018. A Performance and Resource Consumption Assessment of Secret Sharing Based Secure Multiparty Computation. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11025)*, Joaquín García-Alfaro, Jordi Herrera-Joancomartí, Giovanni Livraga, and Ruben Rios (Eds.). Springer, 357–372. https://doi.org/10.1007/978-3-030-00305-0_25

[35] Yanyan Zhuang, Chris Matthews, Stephen Tredger, Steven Ness, Jesse Short-Gershman, Li Ji, Niko Rebenich, Andrew French, Josh Erickson, Kyliah Clarkson, Yvonne Coady, and Rick McGeer. 2014. Taking a walk on the wild side: teaching cloud computing on distributed research testbeds. In *The 45th ACM Technical Symposium on Computer Science Education, SIGCSE '14, Atlanta, GA, USA - March 05 - 08, 2014*, J. D. Dougherty, Kris Nagel, Adrienne Decker, and Kurt Eiselt (Eds.). ACM. https://doi.org/10.1145/2538862.2538931

[36] Noa Zilberman. 2020. An Artifact Evaluation of NDP. *Comput. Commun. Rev.* 50, 2 (2020), 32–36. https://doi.org/10.1145/3402413.3402418

[37] Noa Zilberman and Andrew W. Moore. 2020. Thoughts about Artifact Badging. *Comput. Commun. Rev.* 50, 2 (2020), 60–63. https://doi.org/10.1145/3402413.3402422

A APPENDIX

This is the description of the presented sample experiment (cf. Sec. 5), which provides the means to retrace the experimental workflow in a step-by-step manner, thus reproducing the experimental results available [31]. In addition, we use GitHub's website creator to document the contents of this repository [32], where we list and explain these artifacts.

A.1 Virtual Testbed

To reproduce the results of the case study, we provide access to our virtual testbed via the web browser. vpos can be accessed at https://virtualtestbed.net.in.tum.de. This web service allows the creation of separate vpos instances with a single click. After booting one of these instances, a connection to this instance can be established with a second click that starts the web shell of our virtual testbed controller host called *vkaunas*.

A.2 Experiment Reproduction

This section explains the experiment according to the three phases of the pos workflow.

Setup phase All the commands to start the experiment are provided in Listing 1. To reproduce the experimental results, the first step is to clone our repository on vkaunas. The experiment scripts can be found in the experiment subfolder of the cloned repository. The `experiment.sh` file initiates the experiment, requiring two arguments that determine which hosts to use as LoadGen and DuT. All machines in vpos have the same number of CPU cores and RAM; therefore, any two of the available nodes can be selected. For our experiments, we selected *vriga* as the load generator and *vtartu* as the DuT. The experiment script uses the testbed controller pos to allocate both experiment hosts, load all the necessary parameters (namely local, global, and loop), set the image to Debian Buster, and trigger a reboot. After booting, the setup scripts are executed on the DuT and the LoadGen. pos synchronizes the end of the setup phase between the two hosts, i.e., the experiment continues only after all the experiment hosts have completed their setup. The entire experiment runs for approximately 3 h.

Listing 1: Commands to start the experiment

```
cd /home/user
git clone https://github.com/gallenmu/pos-artifacts

cd /home/user/pos-artifacts/experiment
./experiment.sh vriga vtartu
```

Measurement phase At the end of the setup phase, pos schedules multiple runs of the `measurement.sh` files on DuT and LoadGen depending on the `loop-variables.yml` file. For this experiment, we defined two parameters in the loop variables file: packet size (`pkt_sz`) and rate (`pkt_rate`). Both parameters contain lists with two entries for the packet size (64 and 1500 B) and 30 entries for the packet rate (10 000 to 300 000 packets/s). To investigate each possible combination of the packet size and rate, pos calculates the cross product, which results in a total of 60 individual measurements. In addition, pos schedules a separate measurement run for each of the possible combinations. During the execution of the measurement, pos displays a bar visualizing the progress of the experiment. The results of the measurement runs are uploaded to vkaunas into the folder `/srv/testbed/results/user/default/[timestamp]/*`. The pos-artifacts repository contains a copy of our own results in the folder `results/2020-10-12_11-20-32_230471`.

Evaluation phase After the end of the measurement phase, the results can be evaluated. The plotting scripts are contained in the `plot_scripts` folder of our repository. We created our own plotter that can directly parse the output of the MoonGen packet generator and pos result files to generate plots that are iterated over the defined loop parameters. The plotting scripts can use throughput and latency data created by MoonGen. However, in our VM, we cannot generate latency measurements, due to the limited hardware support. Therefore, we only create the throughput plots. The plot scripts put the generated files into the `figures` folder. For this experiment, we do not call the plotting script directly but rather create the `publish.py` script to do the job. In addition to creating plots, this script also prepares the experiment for publishing. Listing 2 contains the command to execute the publication script.

Listing 2: Experiment evaluation/publication

```
cd /home/user/pos-artifacts

pip3 install -r plot_scripts/requirements.txt

python3 publish.py \
     -x /home/user/pos-artifacts/experiment \
     -r /srv/testbed/results/user/default/[timestamp] \
     -g https://github.com/[username]/[reponame]

cp -r /srv/testbed/results/user/default/[timestamp] \
   /home/user/pos-artifacts/results
```

Publication To enable data exchange between the user's computer and vpos, we recommend the use of a GitHub repository. The link to this GitHub repository is used as a parameter for the `publish.py` script where users select their own `[username]` and `[reponame]`. After running the publication script, the figures folder contains the generated plots. To take a look at the generated plots, the created svg files can be uploaded. To preserve all the experimental artifacts, the entire content of the `pos-artifacts` folder can be uploaded to the created repository. If these contents are pushed to the *gh-pages* branch, a website [32] will be created and hosted by GitHub. The website contains a README.md file that lists the contents of the repository. Authors may choose to either add all the created artifacts to the released repository or to specifically select the artifacts they want to publish.