I8-Testbed: Introduction

Michael Haden, Benedikt Jaeger*, Sebastian Gallenmüller*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: michael.haden@tum.de, jaeger@net.in.tum.de, gallenmu@net.in.tum.de

Abstract—**There are different ways for testing of network technologies. In this rapidly changing area the reproducibility of the performed experiments is especially important in order to be validated for correctness by other scientists. In this paper we present our I8-testbed, its architecture, tools and workflow for fully automated and reproducible network experiments. Special attention is paid to its main part the internally developed plain orchestrating service which manages the testbed orchestration, execution and evaluation to achieve reproducible experiment results. An important aim of this paper is to be a starting point and to facilitate the entry for new users into the creation of own experiments on the testbed.**

Index Terms—**testbeds, measurement, pos, reproducibility**

1. Introduction

In experimental research the right testing infrastructure plays a crucial role in achieving reproducible results. There are various ways for an implementation depending on the requirements of the particular research focus of the chair. We present the testbed of our chair and its main part the pos, the plain orchestrating service. It executes the fully automated workflow for execution and evaluation of network experiments. So far only a brief web documentation of different parts of the testbed and some example scripts exist [1] which in our experience causes a prolonged learning period for a new user. Therefore it is the motivation of this paper to provide a more detailed documentation that gives an overview and acts as an entry point for new users of the testbed to minimize the time spend learning to create own experiments.

The goals of this paper are therefore to provide a general understanding of the architecture of our specific testbed for a new user, how the experiments are automated, orchestrated, executed and what distinguishes it from different approaches. Moreover this paper should provide a good overview of the different functionalities of the testbed orchestration service pos, when each function is used and how a typical experiment workflow looks like. Finally after reading this paper a new user should be able to understand the functioning of other testing scripts and finally to create experiments on their own.

The paper is structured as follows. Section 2 explains basic terms for the understanding of the further sections. Section 3 reviews how the testbed compares to other testbeds. In Section 4 we provide an overview of our testbed architecture including pos. In Section 5 a typical experiment workflow is shown on basis of an example script, before the paper is concluded in Section 6.

2. Background

The ACM distinguishes experiment results into 3 main categories [2]:

Repeatability means that the same experiment result can be repeatedly obtained by the same researcher using the same measurement setup each time. As the weakest classification every well-controlled experiment should fulfill this condition.

Reproducibility has been achieved by an experiment if the same results can also be obtained by a different researcher but using the same measurement setup. This presupposes that the original setup is made available for other researchers by publishing all used tools and scripts or by providing access to the infrastructure that was initially used.

Replicability is the final goal for every experiment result. Different researchers obtain the same measurements by using their own self developed experiment setup which differs from the initial setup.

3. Related work

In this section we provide an overview of the differences between our testbed and some others that specify on reproducible research mainly based on the research done by Nussbaum [3].

A main characteristic of our testbed are the dedicated hosts that are used for every test node together with the live images that are booted for every experiment. This puts it in a common ground with Emulab which also uses a cluster of bare metal systems that are allocated exclusively to users. In contrast to PlanetLab which uses a container technology to share one host between different users resulting in changing testing conditions.

Chameleon, CloudLab and Grid'5000 also support a customized configuration by users like our testbed but not all provide information about its implementation.

Compared to Planet-Lab where the Internet has a direct impact on the experiment result and thus making the results not reproducible [4] Emulab and our testbed are in no influence by outside networks. That means that all traffic which is needed for the experiment needs to be generated by a load generator like MoonGen [5] on one of the test nodes in the testbed.

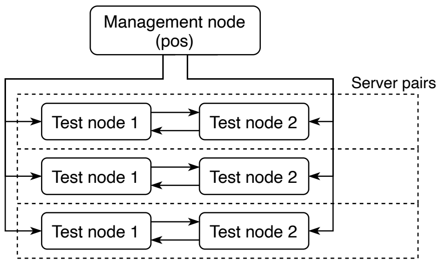

Figure 1: Testbed architecture

4. I8-Testbed

The chair infrastructure we use for our network experiments consists of multiple testbeds with different points of focus. [1] One testbed with a focus on testing Software Defined Networking (SDN) for instance is built up of 15 test nodes (servers) having different network connectivity. [5] Each testbed is structured in a management node and the different test nodes as seen in Figure 1. A dedicated bare-metal server controlled over the Intelligent Platform Management Interface (IPMI) is used for every node instead of using virtualization to prevent a possible distortion of the results by the simultaneous use of a server by several users. IPMI allows the management and control of the server hardware over network also if powered off or unresponsive and therefore eliminates the need for being physically present in front of the server. To encourage reproducibility new images of the desired operating systems are live booted for every test to get reproducible results. A reboot therefore leads to a fresh operating system and a loss of all data on the node but also forces users to plan out their experiments including configuration and result gathering as scripts. Which in turn has the advantages that every experiment by itself is fully repeatable and every test node stateless which results in less administrative workload to maintain each system. Each authorized user has a home directory and remote ssh access to the management node and root access on the test nodes which enables others to replicate the experiments. On the management node runs the management tool pos (plain orchestrating service) daemon which follows the overall sequence of allocating the needed hosts, orchestrating the experiments and collecting the experiment data. The pos daemon offers a REST-API which can be accessed by using the python library `posapi` or its command line interface `pos cli`, both accessible from the management node. They allow reservation, allocation, configuration and management of test nodes and are used to launch commands on them. From the test nodes themselves the communication with the pos daemon happens through the python library `postools` or via the `pos_*` commands on the command line. They are used to synchronize test nodes, start, kill or wait for an additionally process in the background or transferring test results or files that are needed for the experiment between the management node and the test nodes. The configuration of the experiments happens over variables that are set for the experiment to make the configuration easier.

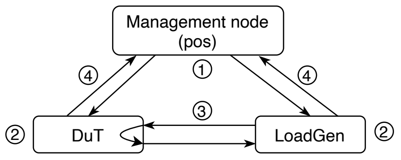

Figure 2: Typical experiment setup [5]

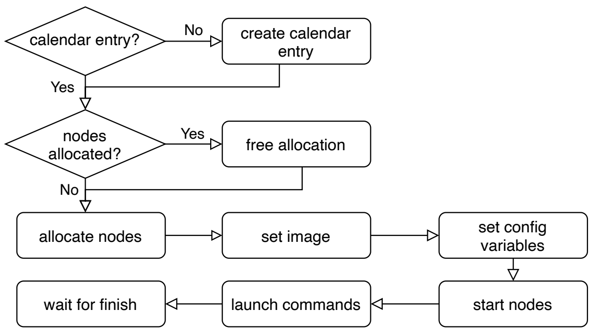

Figure 3: Typical node allocation workflow

5. Workflow of the I8-Testbed

This section provides a high-level overview of the workflow of a typical experiment from setup, execution to evaluation, first from the management node side as seen in Figure 3 and then from the side of the testing nodes as seen in Figure 5. A typical experiment setup consists of two test nodes, one of which is the LoadGen with a load generator like MoonGen [5] and the other the DuT as seen in Figure 2. In the following example we will explain an experiment to measure the impact of restricting the CPU frequency on the network latency between two nodes. After the management node has setup the nodes ① and the experiment is configured ② , one node, the LoadGen, will generate traffic that is sent to the other node, the DuT ③ , who forwards the received traffic back to the sender. The latency of the packets is measured and the results are transferred back to the management node ④ . This setup is often used in various ways to benchmark software like firewalls for latency or data rate of the connection. The corresponding commands that are referenced in the following sections for the management node and the two test nodes are to be found in Listing 1, Listing 2 (DuT) and 3 (LoadGen) respectively. The references per line of the test node scripts follow the pattern (DuT, LoadGen) or (D) (L) e.g. (7,4) or (D7) (L4) when referencing to line 7 in the DuT and line 4 in the LoadGen script.

5.1. Management node

Access to testbed. To get access to the chair infrastructure including the testbed, a chair account is needed which will be created together with the user's advisor. The account is created and the user gets the corresponding password. Due to security reasons the infrastructure of the chair itself is only accessible mostly via ssh key instead of 'password only'. Therefore it is necessary to upload a ssh key to the ssh gateway where it is added into the central ldap.

Reservation. To allow a simultaneous use of the different nodes of a testbed by several users and a better organization of the resources over time a reservation for needed

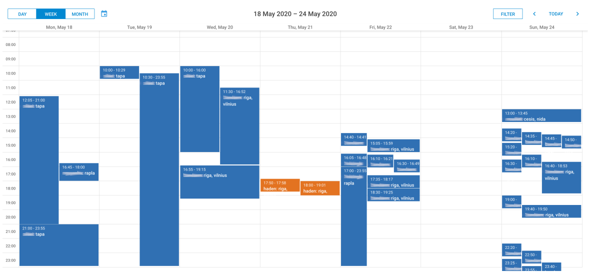

Figure 4: Web calendar reservation [6]

nodes is mandatory for every experiment. This reservation can be done by authorized users through a calendar web view as seen in Figure 4 where all reserved nodes and their users are listed, by using the pos cli on the used testbed itself or by specifying a duration when creating the allocation. The reservation can be extended later on. To get familiar with the system it is desirable to use the nodes with 1 Gbit/s connection because of their low usage. The nodes are getting reserved for the time of the calendar entry during which a claim of the node allocation (which follows shortly) by other users (except admins) is prevented. After the reservation has expired the allocation can be freed by everyone.

Node allocation. With a valid calendar entry the allocations of needed nodes can now be freed if still allocated by their last user (1) and a new allocation can be created with the pos cli (3). An allocation ID and the path to a new corresponding directory for this experiment will be created and returned. By using the ID more nodes can be added later. Admins can also free any reservation by using the `--force` flag.

Configuration. The configuration of the experiment works by using YAML formatted files that are referenced at with the pos cli (4). For most use-cases only simple key value configuration is needed. The variables of the configuration will then be loaded to the specified nodes where they can then be accessed by the nodes with the pos cli. The variables can be either set as local or global.

Image. Because of the live booted operating system the desired image of each node needs to be selected and started (7). There are multiple testbed images available including the most current versions of standard distributions which are updated regularly. Usually a regular Debian image is enough for the experiments. Custom images are also supported by using mandelstamm [7] which is a collection of scripts for building custom images for pos-controlled testbeds.

Booting. The management node can now transfer the selected image to the nodes and boot them by using PXE boot. When `reset` is specified the nodes will be stopped if running and then booted (9).

Command launching. The experiment itself on the test nodes consists of bash or python scripts which contain the commands that shall be executed on the nodes. These scripts are selected via the pos cli (11), automatically transferred to the defined nodes and executed. The commands can be launched as blocking, non-blocking or queued. Now the test nodes run the experiment as described in the next section. When invoked as non-blocking or queued a command ID is returned which can be used to wait for the commands to finish (13).

After the experiment is done the used nodes are stopped and their allocation will be freed (14). The experiment results are now available on the management node where they can be further processed and analyzed.

Evaluate results. The results are placed in the directory as stated at the allocation with a root directory that is named with a unique timestamp of the experiment and under that is the configuration of the experiment and the uploaded results which make then the experiment easier to reproduce. On the second level are folders for configuration, each node that was involved in the experiment and energy measurements if the node supports them. `Config` contains values that define the configuration of the nodes like kernel version, physical address and the configuration variables that were set for the experiment. This is essential to replicate the results with identical resources and configuration. The test node directories contain the program code that was used for the experiment, the produced output of the program and the actual results that were specified to be uploaded. If energy measurements were started during the experiments they are also in a separate directory.

Listing 1: Experiment script on management node [8]

```
1  pos allocations free NODE1
2  pos allocations free NODE2
3  pos allocations allocate NODE1 NODE2
4  pos allocations variables NODE1 PATH
5  pos allocations variables NODE2 PATH
6  pos allocations variables NODE2 PATH --as-global
7  pos nodes image NODE1 debian-buster
8  pos nodes image NODE2 debian-buster
9  pos nodes reset NODE1
10 pos nodes reset NODE2
11 pos commands launch --infile PATH NODE 1 --
     queued
12 pos commands launch --infile PATH NODE 2 --
     queued
13 pos commands await ID
14 pos allocations free ALLOCID
```

5.2. Test node

The scripts on both nodes are now executed and start their procedure.

Setup system. First, the shell is configured to log every command to be able to retrace the sequence of events later (1) and exit when an error occurs (2). Afterwards, various system specific variables of the DuT are logged like the current kernel version (3) that might be of interest in the experiment evaluation later.

Both nodes then clone the program code that is used for the experiment (8,5), in this case *libmoon* and the tool *MoonGen* which builds upon it. A specific branch is checked out by fetching the corresponding variable that was set beforehand (10,7) and the hash of the exact git commit is written back (11,8) to the management node. Important configuration variables like the exact commit

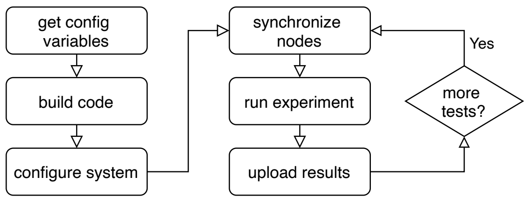

Figure 5: Typical experiment workflow

that is used are again logged for future reference. It should be noted that the access of allocated variables creates traffic on the management interface and should therefore be only used between measurements or during setup to not distort the experiment results. Now the code on both nodes can be freshly built (12,9) and the DuT is configured for the experiment by disabling turbo boost and locking the frequency of the CPU (14) on a value specified in the configuration. Additional variables like the network interfaces that are to be used and the measurement period are fetched. By specifying the minimum and the maximum transfer speed that should be tested in combination with the delta that shall be used between the tests, the LoadGen calculates the individual speed steps that are to be tested in this experiment (L14).

Synchronize nodes. The steps are then synchronized with the DuT by setting the corresponding variable from the LoadGen (L15) and initiating a sync on both nodes by using an instruction that will only return when all nodes called it (20,16). This ensures that the DuT can receive the variable of the steps (21) that was set from the LoadGen which completes the experiment setup.

Run experiment. Now the actual experiment can be started which is divided into multiple test runs with a different transmission speed as specified in the speed steps. Each step begins by specifying a location of the experiment results on the LoadGen named after the current step (L19) and a synchronization of the two nodes (24,20). The freshly built programs can then be launched in the background with the defined configuration (25,22). The LoadGen starts a Lua script that generates traffic with a specified packet rate on a defined port, receives it again on a different one and writes the latency of the received packets to a file. The DuT also executes a Lua script of libmoon that simply forwards the traffic that it receives back to the transmitter. When running the scripts a command id is specified, named after the current step, which is later used as reference point to the launched command.

Upload results. After the measurement period has expired (L24) the LoadGen kills the program per command id (L25) and uploads its results back to the management node (L26). A sync is then initiated (2,27) which informs the DuT that this test run is finished and the program on its side can also be killed (D28). The experiment is now repeated with the rest of the steps that are to be tested after which the script is done and exits.

Listing 2: dut.sh [9]

```
1  set -e
2  set -x
3  pos_set_variable host/kernel-version $(uname -v)
4  pos_set_variable host/os $(uname -o)
5  pos_set_variable host/machine $(uname -m)
6  GIT_REPO=$(pos_get_variable git/repo)
7  DUT=libmoon
8  git clone --recursive $GIT_REPO $DUT
9  cd $DUT
10 git checkout $(pos_get_variable git/commit)
11 pos_set_variable git/commit-hash $(git rev-parse
       --verify HEAD)
12 ./build.sh
13 ./setup-hugetlbfs.sh
14 echo 1 >   /sys/devices/system/cpu/intel_pstate/
       no_turbo
15 echo $(pos_get_variable cpu-freq) > /sys/devices
       /system/cpu/intel_pstate/max_perf_pct
16 echo $(pos_get_variable cpu-freq) > /sys/devices
       /system/cpu/intel_pstate/min_perf_pct
17 RUNTIME=$(pos_get_variable runtime --from-global
       )
18 PORT_TX=$(pos_get_variable port/tx)
19 PORT_RX=$(pos_get_variable port/rx)
20 pos_sync --tag sync_steps
21 STEPS=$(pos_get_variable --remote --from-global
       steps)
22 pos_sync
23 for STEP in $STEPS; do
24     pos_sync
25     pos_run dut$STEP -- ./build/$DUT examples/l2
           -forward.lua \
26     $PORT_TX $PORT_RX
27     pos_sync
28     pos_kill dut$STEP
29 done
```

Listing 3: loadgen.sh [10]

```
1  set -e
2  set -x
3  GIT_REPO=$(pos_get_variable git/repo)
4  LOADGEN=MoonGen
5  git clone --recursive $GIT_REPO $LOADGEN
6  cd $LOADGEN
7  git checkout $(pos_get_variable git/commit)
8  pos_set_variable git/commit-hash $(git rev-parse
       --verify HEAD)
9  ./build.sh
10 ./setup-hugetlbfs.sh
11 RUNTIME=$(pos_get_variable runtime --from-global
       )
12 PORT_TX=$(pos_get_variable port/tx)
13 PORT_RX=$(pos_get_variable port/rx)
14 STEPS=$(calc_steps)
15 pos_set_variable --as-global steps $STEPS
16 pos_sync --tag sync_steps
17 pos_sync
18 for STEP in $STEPS; do
19     OUTFILE="/tmp/histogram${STEP}.csv"
20     pos_sync
21     sleep 2
22     pos_run loadgen$STEP -- ./build/$LOADGEN
           examples/l2-load-latency.lua \
23     $PORT_TX $PORT_RX -r $STEP -f $OUTFILE
24     sleep $RUNTIME
25     pos_kill loadgen$STEP
26     pos_upload $OUTFILE
27     pos_sync
28 done
```

6. Conclusion

The I8-Testbed with pos simplifies the whole testing workflow and eases the realization of reproducible experiments by providing a reliable orchestration for performing and evaluating network experiments. A main problem was that the introduction to reproducible research on our testbed for new users may be difficult because no good entry point and overview was present. We believe we have improved it by starting with presenting the meaning of different reproducibility terms and the characteristics and differences of our testbed in comparison to others. By proving a thorough explanation of the general architecture of the testbed, pos and the interaction between parts of the testbed in combination with a typical workflow we think that a new user gets a good entry and overview into working with our testing infrastructure. Moreover a typical example script that is the foundation for most of the experiments was provided and explained in detail which gives a good overview over all functions of pos, should enable a new user to understand additional scripts and finally implement own experiments on the basis of this one on the testbed.

References

[1] "I8-testbeds Wiki," https://gitlab.lrz.de/I8-testbeds/wiki/-/wikis/home, 2020, [Online; accessed 03-June-2020].

[2] "Artifact Review and Badging," https://www.acm.org/publications/policies/artifact-review-badging, 2020, [Online; accessed 08-June-2020].

[3] L. Nussbaum, "Testbeds support for reproducible research," in *Proceedings of the Reproducibility Workshop*, ser. Reproducibility '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 24–26. [Online]. Available: https://doi.org/10.1145/3097766.3097773

[4] N. Spring, L. Peterson, A. Bavier, and V. Pai, "Using planetlab for network research: Myths, realities, and best practices," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, p. 17–24, Jan. 2006. [Online]. Available: https://doi.org/10.1145/1113361.1113368

[5] S. Gallenmüller, D. Scholz, F. Wohlfart, Q. Scheitle, P. Emmerich, and G. Carle, "High-performance packet processing and measurements," in *2018 10th International Conference on Communication Systems Networks (COMSNETS)*, 2018, pp. 1–8.

[6] "baltikum Calendar," https://kaunas.net.in.tum.de/, 2020, [Online; accessed 15-June-2020].

[7] "mandelstamm - Create OS images for the pos-controlled testbeds," https://gitlab.lrz.de/I8-testbeds/mandelstamm, 2020, [Online; accessed 08-June-2020].

[8] "simple-moongen setup.sh," https://gitlab.lrz.de/I8-testbeds/pos-examples/-/blob/master/tutorials/simple-moongen/bash/setup.sh, 2020, [Online; accessed 15-June-2020].

[9] "simple-moongen dut.sh," https://gitlab.lrz.de/I8-testbeds/pos-examples/-/blob/master/tutorials/simple-moongen/bash/commands/dut.sh, 2020, [Online; accessed 15-June-2020].

[10] "simple-moongen loadgen.sh," https://gitlab.lrz.de/I8-testbeds/pos-examples/-/blob/master/tutorials/simple-moongen/bash/commands/loadgen.sh, 2020, [Online; accessed 15-June-2020].