Reproducibility In Network Experiments With Simulators

Mariette Souppe
UC Santa Cruz
msouppe@ucsc.edu

Kerry Veenstra
UC Santa Cruz
veenstra@ucsc.edu

Ivo Jimenez
UC Santa Cruz
ivo.jimenez@ucsc.edu

Katia Obraczka
UC Santa Cruz
katia@soe.ucsc.edu

ABSTRACT

In the mobile and wireless network domain there are a lot of simulations and domain tools that are required to be produce results of an experiment. Our approach is to minimize the amount of time spent to set up experiments and assumptions for experiments and be able to reproduce experiments with available tools for any user to conduct an experiment on their own environment with no clashing dependencies. Using Popper, a convention and CLI tool, and Docker, a container image, allows any user to reproduce experiments from their peers or a research paper with the exact parameters and assumptions has the author made allowing for easy replication and reproducibility. This methodology is used on the

CCS CONCEPTS

• **Software and its engineering** → **Software performance**; **Software testing and debugging**; **Acceptance testing**; *Empirical software validation*; • **Social and professional topics** → *Automation*;

ACM Reference Format:
Mariette Souppe, Kerry Veenstra, Ivo Jimenez, and Katia Obraczka. 2018. Reproducibility In Network Experiments With Simulators. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering, April 9–13, 2018, Berlin, Germany*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3184407.3184422

1 INTRODUCTION

The approach that *Reproducible Network Experiments Using Container-Based Emulation* by Handigol, Heller [**???**] uses a container based solution. However, since it using a container-based emulation this method of reproducibility is using an emulator and not the actual simulator that was originally used to conduct the experiments. Conducting an experiment on an emulator does not have all of the bells and whistles compared to using the actual simulator. As a result, using an emulator to rerun an experiment would have some limitations.

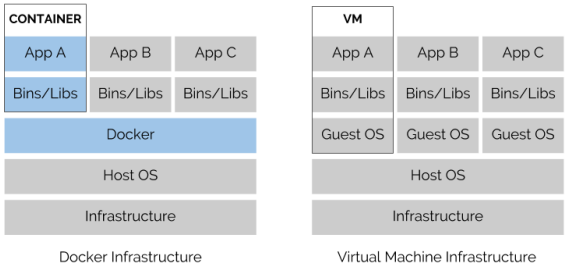

Figure 1: figure caption goes in here

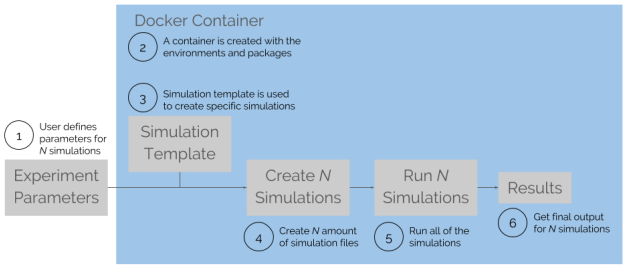

Figure 2: figure caption goes in here

2 METHODOLOGY

2.1 Popper

Popper

2.2 Docker

In order to achieve a reproducibility model for experiments so that an experiment can truly run on any personal machine the technology tool called Docker is used to accomplish this goal. Docker, a

ICPE '18, April 9–13, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5095-2/18/04…$15.00
https://doi.org/10.1145/3184407.3184422

technology container, creates an environment which packages an application with all of the application's dependencies. In the experiment, which is further explained [in section], this tool is used the experiment needs Java and Python installed on a machine in order for the whole experiment to run properly. Normally, one would have to make sure that both of these dependencies are installed on one's personal machine, however with the use of Docker an environment is created with those dependencies

This enables the portability of an experiment which further helps achieve reproducibility.

2.3 Contiki

2.4 Cooja

For the experiment, to be described in the next section, uses Cooja to conduct the main experiment. Cooja is a network simulator that is used in wireless sensor networks which allows to simulate small or large networks.

3 EXAMPLE

3.1 Background

The experiment where this methodology is used on the paper *Guiding Sensor-Node Deployment Over 2.5D Terrain* by Veenstra [**???**]. The idea of this experiment is to initially place nodes on a specified terrain map with a given range. Then using the algorithm [**???**] continually moves the nodes around the given terrain until a final cumulative visibility value has been computed. As a result, the output of the experiment is the final cumulative visibility value where of all of the nodes are placed in a way that maximizes the coverage a given terrain.

In order for this experiment to run there are certain parameters that need to be defined. The first parameter in the configuration file is a file name for different simulations. This enables to differentiate between the different simulations. The second parameter is the allows the user to choose if they would like to use a random seed which enables to produce the same sequence of random numbers for testing purposes. This parameter is a boolean where *random* can be turned on or off. Depending on whether a random seed has been initialized to True or False, if True was chosen then a random seed value will need to be initialized. If the random is set to False, then is random seed parameter will be ignored. The last parameter that needs to be defined is the number of nodes desired to put on a terrain which can range from 1 - 50 nodes. As of right now the algorithm can handle a maximum of 50 nodes.

3.2 Pipeline

Fig. 3, we show the pipeline

- **.popper.yaml**
  This file consists the ordering in which popper executes the main scripts; setup.sh, run.sh, post-run.sh to run the entire experiment with out extra input.
- **.git**
  Currently the full experiment is not on Git, which it will be soon, however having the whole pipeline on Git will achieve the portable and reproducibility goal. Any user will be able to clone the repository and be able to reproduce the experiment with out having to individually download dependecies.
- **Dockerfile**
  Retrieves the docker image with Java and installs Python, pip, java-random, pyyaml, and jinja2
- **setup.sh**
  Builds a docker container with the environmnents, dependencies, and packages that are needed to run the pipeline.
- **run.sh**
  Will run the simulations in the environment that has been previously defined.
- **sim_config.yaml**
  The simulation configration file allows the user to run multiple experiments with different values for the parameters. The parameters that the user can define are the filename, decide to use a random generator true/false for nodes placement, random seed, and number of nodes (maximum of 50). The filename is used to name the different experiments so when looking at the final output it will be evident which experiment produce certain results. As of right now the algorithm has hard coded values for the initial placement of the nodes, but as the experiment is further developed the random generator will radomly generator the node placement. [Add in about random seed and point of that] The user can also define the amount of nodes desired for the experiment with a maximum of 50 nodes, since that is the maximum the algorithm of the experiment can handle.

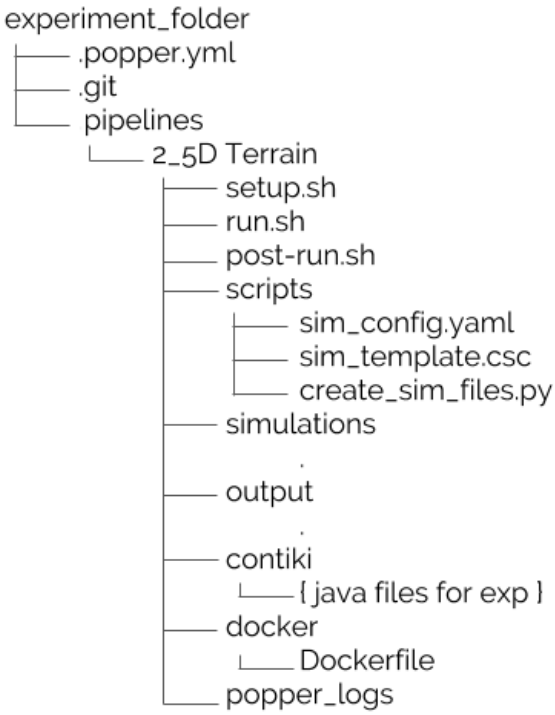

Figure 3: figure caption goes in here

- **sim_template.csc**
  The template file consists of how the simulator reads in script to create the simulations based on the defined parameters for each experiment.
- **create_sim_files.py**
  For the N amount of simulations defined in the sim_config.yaml, this file will merge the parameters with the template to create N amount of simlations.
- **simulations directory**
  All of the simulation files that were produced from create_sim_files.py which are then run in run.sh.
- **output directory**
  The output for each of the simulations containing the final visability value between the nodes [Need better wording and explain that a bit more]
- **contiki/tools/cooja**
  The main experiment files are located, where Cooja is the simulator used for the experiment.

4 FUTURE WORK

The main limitation in *quiho* is the requirement of having to execute a test on more than one machine in order to obtain IRUPs. On the other hand, we can avoid having to run `stress-ng` every time the application gets tested by integrating this into the infrastructure (e.g., system administrators can run `stress-ng` once a day or once a week and make this information for every machine available to users).

We are currently working in adapting this approach to profile distributed and multi-tiered applications. We also plan to analyze the viability of applying *quiho* in multi-tenant configurations and to profile long-running (multi-stage) applications such as a web-service or big-data applications. In these cases, we would define windows of time and apply *quiho* to each. The main challenge in this scenario is to automatically define the windows in such a way that we can get accurate profiles.

In the era of cloud computing, even the most basic computer systems are complex multi-layered pieces of software, whose performance properties are difficult to comprehend. Having complete understanding of the performance behavior of an application, considering the parameter space (workloads, multi-tenancy, etc.) is challenging. One application of *quiho* we have in mind is to couple it with automated black-box (or even gray-box) testing frameworks to improve our understanding of complex systems.

Acknowledgments: We would like to thank Bernardo Gonzalez for his feedback on a preliminary version of this paper, as well as all the anonymous reviewers. Special thanks go to our ICPE shepherd. This work was partially funded by the Center for Research in Open Source Software[ˆcross], Sandia National Laboratories, NSF Award #1450488 and DOE Award #DE-SC0016074. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

???{veenstra_terrain_2015, title = {Guiding Sensor-Node Deployment Over 2.5D Terrain}, booktitle = {IEEE International Conference} on {{Communications}}, date = {2015}, author = {Veenstra, Kerry and Obraczka, Katia} }

REFERENCES

http://tiny-tera.stanford.edu/~nickm/papers/p253.pdf https://inrg.soe.ucsc.edu/wp-content/uploads/2015/09/icc-2015.pdf