Reproducing Network Experiments With Popper

Andrea David
UC Santa Cruz
andavid@ucsc.edu

Mariette Souppe
UC Santa Cruz
msouppe@ucsc.edu

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

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

Sam Mansfield
UC Santa Cruz
smansfie@ucsc.edu

Kerry Veenstra
UC Santa Cruz
veenstra@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 for an experiment. Our approach is to minimize the amount of time spent to set up experiments and assumptions. Furthermore, enable a user to reproduce results with available tools on their own environment with no clashing dependencies. In this paper a convention client tool called Popper is used to make reproducibility less of a daunting task. Using Popper, a workflow is developed so a user only needs to define values for predefined parameters. Lastly, this workflow is tested on an existing experiment and compares the results from the original paper and reproduced results.

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:
Andrea David, Mariette Souppe, Katia Obraczka, Ivo Jimenez, Sam Mansfield, and Kerry Veenstra. 2018. Reproducing Network Experiments With Popper. 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

Rerunning an original experiment can be a lot of work since a lot configurations and downloading software are needed in order to reproduce any experiment. In the scientific community, it is important to be able to reproduce existing research paper results to further improve upon or understand that idea.

A tool that is commonly used to rerun experiments is a virtual machine. The base functionality between a virtual machine and using Docker are similar in the ability to create an environment for an application. However, these two tools use their own approaches. In a virtual machine you must define an environment per application, whereas in Docker an environment can be shared among other applications where the applications are run in a container. This allows for sharing resources on your host operating system since Docker dynamically allocate memory. Another added feature to Docker is that modules and packages can also be defined when building the environment which prevents having to be careful of how each operating system handles downloading modules such as *pip* for example.

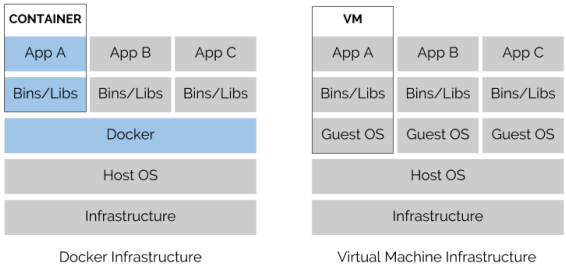

Figure 1: As you can see there is a difference between the virtual machine and the docker infrastructure. In a Docker container, it only contains the application itself and libraries for that experiment since the Docker layer takes care of the experiment environment. In a virtual machine an operating system has to first be defined before running the application.

However, in order configure a virtual machine with all of the correct dependencies and environments to replicate an experiment, there is still some ambiguity in terms of the exact settings of the original experiment. Using Docker and packaging an environment and extra modules, eliminates this ambiguity. Fig. 1 shows the infrastructure of these tools.

The rest of the paper is outlined to show the methodology and tools that are used to reproduce an experiment, an example where this methodology is used in an actual experiment, and lastly our results of reproducing results from the original author's.

2 METHODOLOGY

2.1 Popper

As mentioned previously, Popper is a convention and CLI tool for conducting experiments. Popper helps keep experiments in an organized manner through its convention and uses Git to keep

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

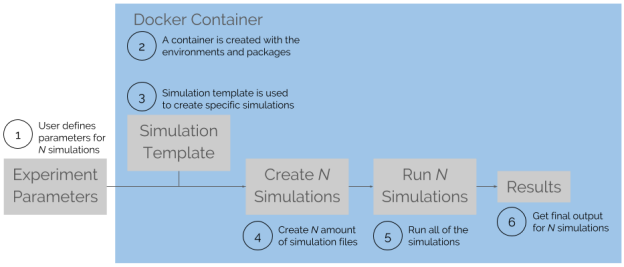

Figure 2: Here is the workflow when using Docker. First, the values of the parameters of the experiment have to be defined by the user. Second, a Docker container is created with the entire environment, modules, and packages for the experiment to run. In the third step, the simulation template gets pulled in to the fourth step when creates *N* amount of simulations that the user has defined. Fifth, those *N* simulations are run and lastly the Cooja.testlog are outputted into the output folder to further evaluate the final result.

experiments self-contained. This allows for easy sharing so that others can reproduce original results.

When first initializing Popper, a directory called Pipeline is created along with a nested directory. This nested directory is how one names the experiment, but it is the contents inside this directory that is important. This directory contains the empty scripts; setup.sh, run.sh, post-run.sh, validate.sh, and teardown.sh. These scripts isolate different steps when making an experiment and helps organize the experiment. It is important to note that *not* all of the scripts are needed for every experiment, so scripts that are not going to be used can be removed from the pipeline and Popper will be able to interpret that. After the experiment has been configured, the experiment can then run with the command *popper check*. This will run all of the scripts that are available.

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 help accomplish this goal. Docker, a technology container, creates an environment which packages an application with all of the application's dependencies. In the experiment, which is further explained in the next section, Docker is used to aid with Java, Contiki, and Python environments installations since the experiment relies on these to run properly. Normally, one would have to make sure that all of these environments are installed on one's personal machine, however, Docker takes care of this issue.

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

2.3 Cooja

Cooja is used to conduct the main experiment. Cooja is a network simulator that is used in wireless sensor networks which allows simulations of small or large networks. Contiki is also used in the experiment but only used for its thin layer operating system to run Cooja. Cooja is a tool of Contiki as you can see in the file directory where the experiment is run. This is why Docker needs to include Contiki when building the environment.

3 EXAMPLE

3.1 Background

The experiment where this methodology is used on is based 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 within a predetermined range. Then using the algorithm [???], nodes continually move around the given terrain until a final cumulative visibility value has been computed. As a result, the output of the experiment is a *final cvis* 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 and output logs. The second parameter 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 the random seed parameter will be ignored. The next 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. The next two parameters are the transmitting range and interference range which are the same. Lastly, the terrain of the experiment can also be defined here as well.

3.2 Pipeline

Fig. 3 shows the pipeline for the experiment and below describes the functionality of each part of 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 without 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 without having to individually download dependencies.
- **Dockerfile**
  Retrieves the docker image with Contiki, Javam, Python, pip, java-random, pyyaml, and jinja2.
- **setup.sh**
  Builds a docker container with the environments, dependencies, and packages that are needed to run the pipeline.
- **run.sh**
  Runs *N* simulations from the simulations directory.
- **sim_config.yaml**
  Values of the parameters are configured by the user.

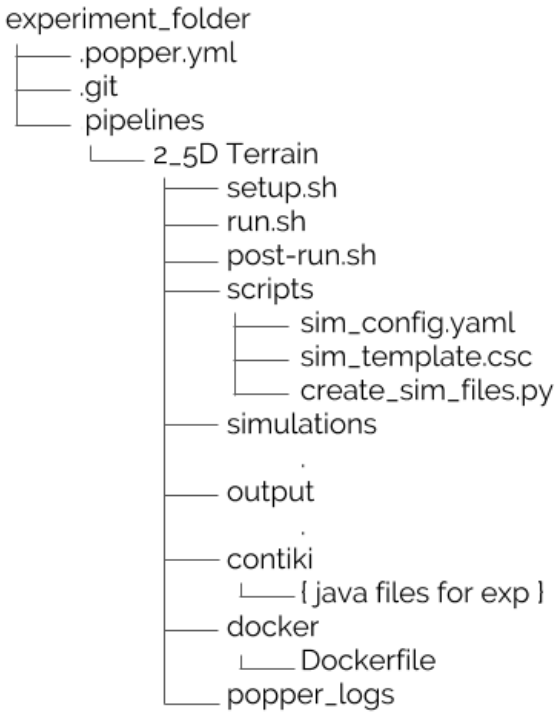

Figure 3: Experiment Pipeline

- **sim_template.csc**
  Simulation template where configured parameters are substituted per simulation.
- **create_sim_files.py**
  Creates *N* amount of simulations defined in the sim_config.yaml, merging the simulation configuration file and template file.
- **simulations directory**
  Contains all of the simulation files.
- **output directory**
  Contains all of the output logs for every simulation.
- **contiki/tools/cooja**
  Contains the main experiment code.

4 RESULTS

The final output obtained from the experiment is the final cumulative visibility value from the final placement of the nodes according to the specified terrain in the experiment.

In Fig. 4 and Fig. 5 we can see that the results are *not* exactly the same. Some of the reproduced results do not have all of the terrains as in the orginal results because not all of the terrains were available while reproducing the experiment. Furthermore, the values in Fig. 5 are higher than the values in Fig. 4. This is because the original paper was programmed in C++ and since then the author has translated the code into a Cooja environment, there is a difference in the end result.

The main take away from these results, despite some missing elements, is that the trend of both Fig. 4 and Fig. 5 are similar.

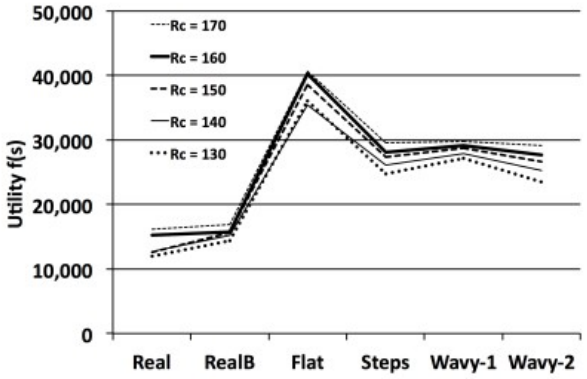

Figure 4: Original results from the 2.5D Terrain Experiment

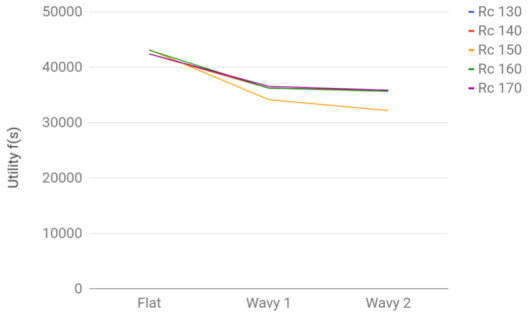

Figure 5: Reproduced results using Popper

5 CONCLUSION

In an ideal world, rerunning an experiment would not have to be a headache due to the numerous configurations, dependencies, and ambiguity of the original experiment. Using Popper and extra tools it was possible to obtain the results needed to achieve the initial goal.