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, research experiments are often conducted using a variety of simulation and software development tools. Many of these simulation platforms, however, require their own operating system and specific setup, which makes replicating and validating research results a challenging task. There is a prominent need for a service that enables researchers to reproduce results with available tools in their own environment with no clashing dependencies. In this paper we use a convention client tool called Popper to make reproducibility of networking experiments less of a daunting task. In particular, we detail the steps taken to automate the execution and re-execution of experimental results presented in [**??** ,1]. Using Popper, we develop a workflow and test it on an existing experiment to compare the results from the original paper and reproduced outcome.

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, 4 pages. https://doi.org/10.1145/3184407.3184422

1 INTRODUCTION

The ability to reproduce previous experiments is one of the most important aspects in scientific research. However, as scientific discovery is rapidly advancing, researchers are pressured to rush publication of new findings and present breakthrough discoveries at conferences. Lately, there has been a growing concern in the research community about results that cannot be reproduced and thus cannot be verified. This is partially due to the lack of incentive for researchers to revoke experiments that were unable to be reproduced [2]. Moreover, replicating scientific experiments in the field of mobile and wireless networks is a challenging task. One of the biggest setbacks of reproducibility is the complexity that comes with rebuilding the same environment in which the original experiment was conducted [3]. Rerunning an original experiment can be strenuous since extensive configuration and software installation is 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 and understand conveyed ideas.

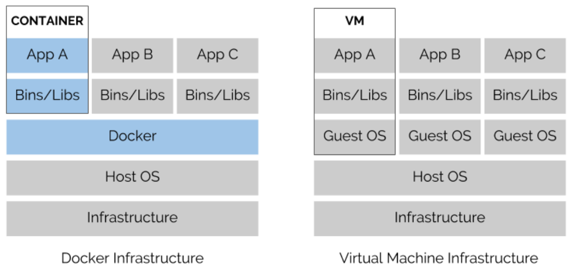


Figure 1: As you can see there is a difference between the virtual machine and the docker infrastructure. In a Docker container, it only contains the application itself and libraries for that experiment since the Docker layer takes care of the experiment environment. In a virtual machine an operating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

Next section (Section 2) reviews related work in the reproducibility field, then (Section 3) which goes over the main idea behind the experiments. (Section 4) goes in-depth of how each of experiments were conducted prior to using the reproducibility model and with the Popper convention followed by the results from the experiments (Section 5). Lastly, a brief discussion on challenges and takeaways from the whole process in (Section 6).

2 RELATED WORK

2.1 Before Reproducibility Tools

Network simulators were not designed with the concept of reproducibility which makes the automation and configuration of simulations more difficult. Previously, a tool that was commonly used to rerun experiments was a virtual machine. Recently, DevOp tools, such as Docker, became popular for packaging and recreating environments as one package. The base functionality between a virtual machine and 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.

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 with all the extra modules eliminates this ambiguity. Fig. 1 shows the infrastructure of these tools.

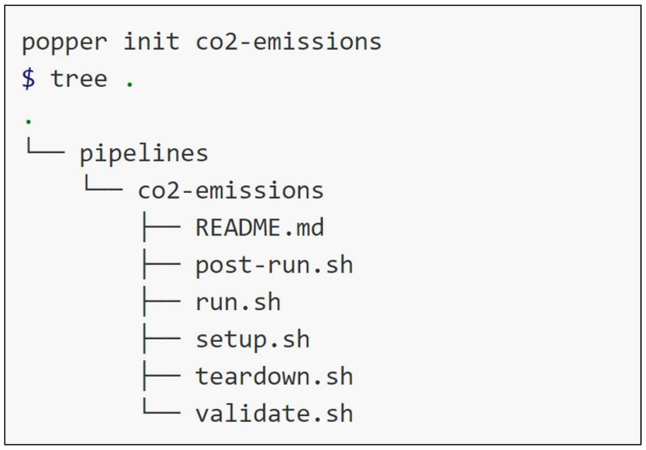
2.2 Existing Tools

Recently, researchers have started developing tools that aim to make the reproduction of scientific findings simple and undemanding. Sciunit [4] is one of such tools developed to encapsulate the author's workflow. It is a command-line tool that saves all dependencies related to executing a command in Linux. Consequently, the command can be later replicated in a different environment without having to obtain the same dependencies related to executing that command. ReproZip [5] is another tool that allows for easy reproduction of experiments using command-line executions. While executing the experiment in its original environment, ReproZip tracks operating system calls and packs necessary components of the experiment into a file. In the unpacking step, ReproZip can reproduce the experiment form that file in another environment.

2.3 Popper

Popper is a convention for creating reproducible scientific articles and experiments [6]. The convention is based on open source software (OSS) development model, using the DevOps approach to implement different stages of execution. The Popper Convention creates self-contained experiments that do not rely on libraries and dependencies other than what is already inside the "popperized" experiment. To achieve reproducibility, Popper uses pipelines that contains shell scripts that execute the original experiment. Fig. 2 shows an example Popper pipeline from one of the Popper lessons.

The Popper pipeline consists of a few stages: the setup, run, post-run, and optional validate and teardown stages with the experiment name co2-emissions. In the setup stage, a user would usually download all the necessary files to run the project. These files are, for example, data files, libraries, and other dependencies. The run stage executes the script that is used to run the original experiment. The post-run stage is where a user would display the results obtained from in the run stage. This stage could be used to open a log file that shows the results of the experiment or run a script that graphs and displays the results. In the case of my experiment, I needed to run an entire operating system on my machine. To achieve this, I used Docker, a DevOps tool that packages applications and environments into containers. This allowed me to create an image of the operating system that contained all the libraries and dependencies needed to run it as just one package. This feature of Popper that allows the use of DevOps tools makes it a very useful convention, as it is demonstrated in the next sections.



```
popper init co2-emissions
$ tree .
.
└── pipelines
    └── co2-emissions
        ├── README.md
        ├── post-run.sh
        ├── run.sh
        ├── setup.sh
        ├── teardown.sh
        └── validate.sh
```

Figure 2: Illustration how a general pipeline looks like using Popper

3 NETWORK SIMULATION

3.1 Background

In many network simulations, especially wireless sensor network simulations, Cooja is used as the main tool to conduct experiments. Cooja is a simulation tool for the Contiki open source operating system, which allows simulations of small and large networks.

3.2 TerrainLOS

One of the experiments that was reproduced in this paper is based on TerrainLOS [???]. TerrainLOS is an outdoor terrain propagation model that aims to create a more accurate simulation of outdoor sensor network communication. Most simulation platforms either assume a completely flat terrain or tend to use very simplistic channel propagation models that do not represent realistic outdoor terrain conditions. To present a more accurate outdoor simulation model, TerrainLOS uses common geographical height maps, called Digital Elevation Models (DEMs). These data files are used in experimental evaluations to investigate communication between nodes under realistic conditions. TerrainLOS describes the Average Cumulative Visibility (ACV) as a metric to characterize network connectivity over the terrain classified in the DEMs. ACV denotes the percentage of sensor nodes that are visible in an area from all locations on a given map. For example, 100% ACV means that every node is visible from all locations, which further implies the presence of a flat terrain. In the experimental methodology, the authors of TerrainLOS refer to the average number of nodes in an area as the average population, and the average number of nodes that a given node can communicate with is referred to as the average node density. These two metrics are used in determining network connectivity. The experiment focuses on automating the execution and re-execution of one of the experimental simulations performed in the paper, called Experimental Connectivity. This experiment focuses on experimentally evaluating the accuracy of connectivity results based on the models presented in the paper. The connectivity results are plotted using the average cumulative visibility metric and population size.

3.3 2.5D Deployment on TerrainLOS

The experiment where this methodology is used on is based on the paper Guiding Sensor-Node Deployment Over 2.5D Terrain [1]. This experiment [1] is a layer that is used on top of the TerrainLOS experiment which uses a 2D dimension for the placement of the nodes using x and y coordinates, then adds a layer of a terrain to give that 2.5D dimension. The idea of this experiment is to initially place nodes on a specified terrain map within a predetermined range. Then using the algorithm from the paper, 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 cumulative visibility value where of all of the nodes are placed in a way that maximizes the coverage of a given terrain.

The old way the experiment [1] was conducted was that the experiment would have to start with a GUI, configure a few parameters such as the number of nodes, terrain, and transmitting range, then initialize a script, run the script, and wait for final results. For each simulation these manual steps had to done.

4 POPPERIZATION

4.1 TerrainLOS

To run TerrainLOS in Cooja, without using Popper, a reviewer would have to go through several steps when attempting to replicate the results [???] . First, they would have to download Instant Contiki and install a Virtual Machine to start Contiki. Then, they would have to download all the necessary files, libraries, and dependencies needed to run TerrainLOS. Lastly, they would have to create a jar file of TerrainLOS, run the simulator, and load it into Cooja. This is a very time-consuming task, not to mention the very likely possibility of encountering errors upon attempting to run the project the first time. Additionally, reviewers rarely have a chance to contact the original author of the experiment and receive step by step instructions or solutions to the encountered errors. For this reason, interpreting error messages is often times cumbersome if not impossible. Using Popper is a much more effortless way to re-execute someone's experiment without the need to have the original author explain the steps needed to execute the experiment.

In the implementation of the Popper pipeline, two stages were generated – the run stage and the post-run stage. The run stage also takes care of setting up the Contiki Cooja environment with the help of Docker containers. Docker creates an image of the Contiki operating system including the Cooja simulator. The main task of the run stage is to execute the author's python script that takes ACV and population size as inputs. The original experiment was run using population sizes of one, ten, thirty, and eighty, and ACVs ranging from one to hundred with increments of ten. The same arguments are used for the reproduced experiment as well. The output of these runs is saved in log files, which are read in the post-runs tage with another python script written by the author. The results are then graphed and saved in an image file. As a result, the original experiment is "popperized" and can be run by just simply executing the "popper check" command inside the experiment pipeline.

4.2 2.5D Deployment on TerrainLOS

When first running the experiment [1], there were a few tools that had to be downloaded before getting the experiment to work. Java and Contiki had to be installed since those are the environments where the experiment runs. Once the environment was set up, the code for the experiment would run in Cooja. Then for every experiment to be run, a simulation file had to be configured per experiment manually. This part of the process can be very lengthy since each simulation contains numerous different parameters. After each simulation script has been configured, each script could be run within the simulator, then after a certain amount of time the final cvis value is obtained which the part of the experiment results.

With the Popper version, there are two stages in the pipeline - the setup stage and run stage. The setup stage builds a Docker container which creates the necessary environment for the experiment to run. Additionally, the setup stage creates simulation scripts for every experiment the user would like to run. In the run stage, each of the scripts that have been made from the setup stage are now run in the Cooja simulator. Furthermore, in the Popper version the user only has to configure one file for multiple simulations where popper will run each simulation individually and then output the final results. Workflow of automation of experiment: 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.

5 RESULTS

5.1 TerrainLOS

5.2 2.5D Deployment on TerrainLOS

As a result, 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.

The resulting graphs both show results for distributed simulated annealing where every data point is calculated for the average of ten nodes in random starting positions on specified terrain [1]. Furthermore, the graph illustrates each communication radius from 130 to 170 with increments of ten for the given terrains.

In Fig. 3 and Fig. 4 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. 4 are higher than the values in Fig. 3. This is because the original paper was programmed in C++ and since then the

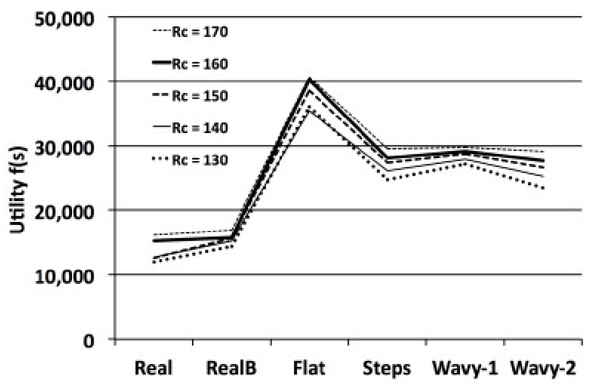


Figure 3: Original results from the 2.5D Terrain Experiment

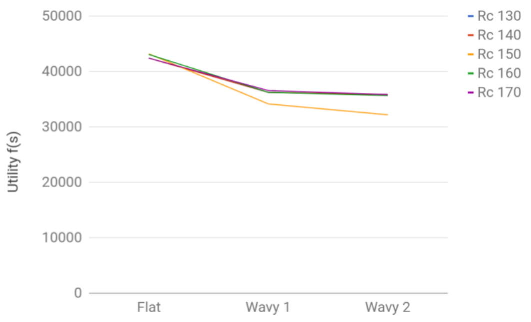


Figure 4: Reproduced results using Popper

experiment has been translated into a Cooja environment, which results a difference in the newer result.

The main take away from these results, despite some missing elements and due to the change in environment, is that the trend of both Fig. 3 and Fig. 4 are similar.

6 CONCLUSION

Reproducibility of published experiments is one of the most essential components of scientific research. However, reproducing experimental results is difficult, mainly due to the complexity involved with setting up the original environment in which the experiment was conducted. Researchers rarely have the opportunity to reach out to the author of the experiment when rebuilding their work in the environment the original project was built in. This process can be time-consuming and often impossible. The Popper convention offers a very straightforward way of implementing reproducibility for scientific research articles. Throughout our work with reproducing the experiments mentioned in this paper, we had the opportunity to closely work with the original authors of the experiments. This made the process of executing and re-executing their experiments simpler, as we were able to double check if the experiments were running the intended way and interpret error messages.

It is harder to try to make a fully finished experiment reproducible, especially if the person trying to reproduce the experiment is not the original author of that experiment. Reproducibility should be kept in mind at the start of an experiment. Reproducing small sections of the code is easier than trying to reproduce code that has already been completed.

REFERENCES

[1] K. Veenstra and K. Obraczka, "Guiding sensor-node deployment over 2.5 d terrain," *Communications (icc), 2015 ieee international conference on*, IEEE, 2015, pp. 6719–6725.

[2] J. Kurose, "Dear colleague letter: Encouraging reproducibility in computing and communications research," *National Science Foundation*, Oct. 2016. Available at: https://www.nsf.gov/pubs/2017/nsf17022/nsf17022.jsp.

[3] I. Jimenez, A. Arpaci-Dusseau, R. Arpaci-Dusseau, J. Lofstead, C. Maltzahn, K. Mohror, and R. Ricci, "PopperCI: Automated reproducibility validation," *Computer communications workshops (infocom wkshps), 2017 ieee conference on*, IEEE, 2017, pp. 450–455.

[4] "Sciunit," *Sciunit*. Available at: https://sciunit.run/.

[5] "ReproZip," *ReproZip*. Available at: https://www.reprozip.org/.

[6] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "The popper convention: Making reproducible systems evaluation practical," *Parallel and distributed processing symposium workshops (ipdpsw), 2017 ieee international*, IEEE, 2017, pp. 1561–1570.