Reproducing Network Simulation 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 computer networks, research experiments are often conducted using a variety of network simulation platforms as well as software development tools. Many of these network simulators, however, require their own specific setup (e.g., operating system, libraries, software dependencies, etc), which makes replicating and validating research results quite challenging. There has been a growing need in experimental computer science research, including computer networks, for tools that enable experiment reproducibility. In this paper, we show how a recently proposed reproducibility tool called Popper facilitates reproducibility of networking experiments. In particular, we detail the steps taken to automate the execution and re-execution of experimental results presented in two research works. Using Popper, we develop a workflow and test it on existing experiments, comparing results from the original and the corresponding reproduced outcome. We also provide a list of lessons we learned throughout this process.

I. 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 breakthroughs. This is especially true in Computer Science and Engineering where knowledge and technology have been advancing overwhelmingly fast and the push to publish new results is even stronger.

Lately, however, there has been growing concern in the experimental computer science and engineering research community about results that cannot be reproduced and thus cannot be verified [1]. There is increasing consensus that it is quite important to be able to reproduce existing research paper results to better understand conveyed ideas and further improve upon them.

Replicating scientific experiments is a challenging task. In experimental computer science and engineering, more generally, and in computer networking, more specifically, one of the biggest setbacks of reproducibility is the complexity that comes with rebuilding the same environment in which the original experiment was conducted [2]. Rerunning an original experiment can be strenuous since extensive configuration and software installation is needed in order to reproduce any experiment. Our motivation behind this paper was our interest in reproducibility due to its challenging nature. We got in contact with the authors [3,4] and obtained all original scripts to reproduce their existing experiments. Additionally, we met with the authors several times to understand how the scripts

map to what is reported in the original paper. Then, we went ahead and started "popperizing" the experiments, going back to the authors as needed, as we had direct access to them.

The remainder of the paper is organized as follows, Section II, goes over the motivation in the reproducibility field. Section III goes over the tool that is used to help make these simulation experiments reproducible. In Section IV, we describe the networking experiments that we reproduce using Popper as well as the network simulation platform we use, while in Section V, we describe how each experiment was conducted originally, i.e., prior to using Popper's reproducibility model. Section VI presents experimental results under Popper and compares them with original results. Lastly, in Section VII, we reflect on our experience and provide a list of lessons learned that we hope will help other practitioners producing this type of networking experiments.

II. MOTIVATION

III. POPPER

Popper is a convention for creating reproducible scientific articles and experiments [5]. The convention is based on the 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. 1 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 stage. 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 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 our experiments, we needed to run an entire operating system. To achieve this Docker, a DevOps tool that packages applications and environments into containers, was used. This allowed us to create an image of the operating system that contained all the libraries and dependencies needed
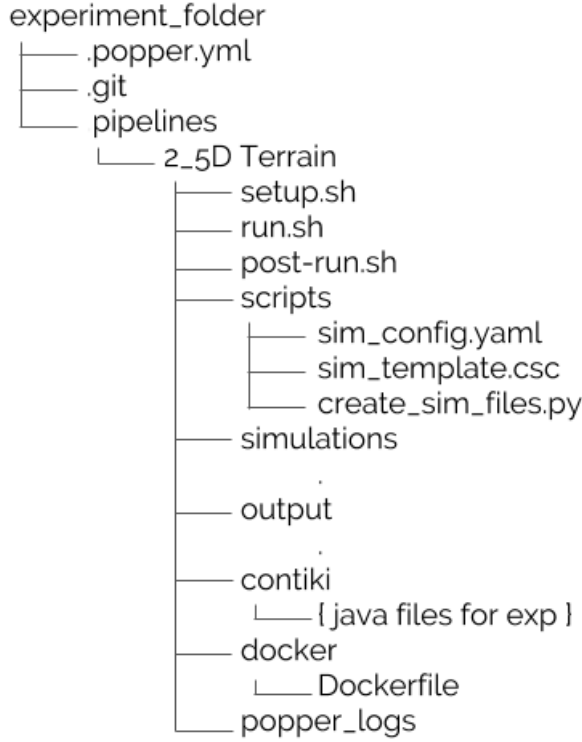
```
experiment_folder
├── .popper.yml
├── .git
└── pipelines
    └── 2_5D Terrain
        ├── setup.sh
        ├── run.sh
        ├── post-run.sh
        ├── scripts
        │   ├── sim_config.yaml
        │   ├── sim_template.csc
        │   └── create_sim_files.py
        ├── simulations
        │           .
        ├── output
        │           .
        ├── contiki
        │   └── { java files for exp }
        ├── docker
        │   └── Dockerfile
        └── popper_logs
```

Figure 1: Illustration how a general pipeline looks like using Popper

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.

IV. NETWORK SIMULATION EXPERIMENTS

A. Network Simulation Platforms

There are a variety of network simulation platforms such as NS3, MiniNet, and Cooja. NS3 [6] is an open source discrete event network simulator that is widely used for simulating environments for network research. Its goal is to provide scalability and ease of use for a variety of networks. Mininet [7] is also an open source simulation tool that provides a virtual network for interacting with Software-Defined Networking applications using OpenFlow. Cooja [8] is a widely used network simulation platform that is specialized in evaluating wireless sensor network applications. Cooja is a simulation tool for the Contiki open source operating system, which is used for building and connecting wireless systems for the Internet of Things [8]. Although each of these network simulators is a popular choice in the networking field, the experiments we are working with are conducted in Cooja, which allows for inclusion of simple radio propagation models.

B. TerrainLOS

One of the experiments that was reproduced in this paper is based on TerrainLOS [3]. 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 defines Average Cumulative Visibility (ACV) as a metric to characterize terrain. ACV denotes the average percentage of node locations that are visible in an area from all node locations on a map. For example, 100% ACV means that every location is visible from all locations, which further implies the presence of a flat terrain. In the experimental methodology, the authors of TerrainLOS define population as the percentage of nodes per location on a given map, e.g., a population of one means there is one node for every one hundred locations on the map. The ACV and the population metrics are used in evaluating network connectivity. Our experiments focus on automating the execution and re-execution of Experimental Connectivity in [3]. 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.

C. Sensor Network Deployment Over 2.5D Terrain

TerrainLOS has been used to evaluate the sensor placement algorithm proposed in [4] that aims at optimizing visual coverage in deployments over 2.5D terrain. The proposed algorithm works as follows: initially a set of nodes is placed on a given region. Then, each node executing the algorithm moves around the terrain to optimize the collective visibility of the network. In the original paper [4], each experiment involved the following steps: initialize a script with parameters such as number of nodes, terrain being used, node transmission range, run the script, analyze the results and repeat multiple times until desireable results have been obtained.

The old way the experiment [4] 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 range, then initialize a script, run the script, and wait for final results. For each simulation these manual steps had to performed.

V. EXPERIMENTS USING POPPER

A. TerrainLOS

To run TerrainLOS in Cooja, without using Popper, a researcher would have to go through several steps when attempting to replicate the results [3] . 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.

B. Sensor Network Deployment Over 2.5D Terrain

When first running the experiment [4], 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 Cumulative Visibility value is obtained. In the Popperized version of the experiment, 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. The automated workflow for this simulation is as follows; 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, from the pipeline created from the popper tool, and the fourth step creates N simulations that the user has defined.
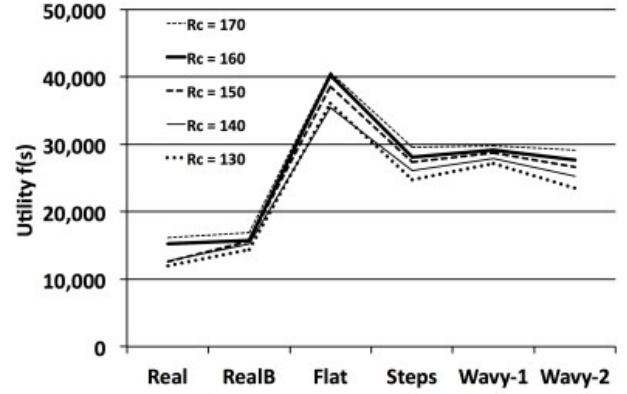

Figure 2: Original results from the 2.5D Terrain Experiment

Fifth, those N simulations are run and lastly the Cooja.testlog are outputted into the output folder to further evaluate the final result.

VI. RESULTS

A. TerrainLOS

The graph of the Experimental Connectivity experiment depicts the percentage of connected networks based on Average Cumulative Visibility and population size [3].

Original results from the Experimental Connectivity experiment in [3].

Reproduced network connectivity results using Popper

Fig. **??** shows the results of the original experiment. The reproduced experiment does not match the original results exactly, as seen in Fig. **??**. This is due to the fact that the experimental outputs for Experimental Connectivity are probabilistic and vary across multiple runs. Additionally, the original experiment results were obtained from a combination of five runs, which resulted in a smoother graph. The reproduced results, on the other hand, were obtained from only one run. Although no results are going to produce the exact outcome from the original experiment, we still observe the general trend in the reproduced experiment.

B. 2.5D Deployment on TerrainLOS

The resulting graphs both show results for every data point is calculated for the average of ten nodes in random starting positions on specified terrain [4]. Furthermore, the graph illustrates each communication radius from 130 to 170 with increments of ten for the given terrains.

In Fig. 2 and Fig. 3 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 original results because not all of the terrains were available while reproducing the experiment. Furthermore, the values in Fig. 3 are higher than the values in Fig. 2. This difference likely is because the original paper used a custom, synchronous simulator that was programmed in C++. Since then the experiment has been translated into a Cooja environment as a new Java model in the event-driven
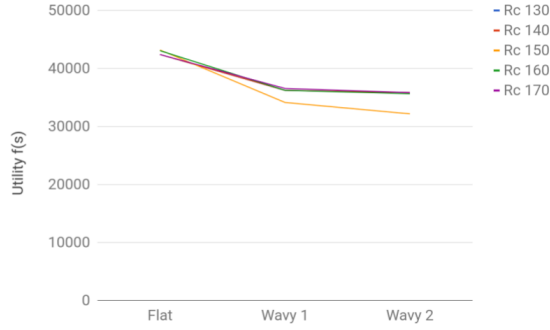
Figure 3: Reproduced results using Popper

simulator by the choice of the author [4]. The main take away from these results, despite some missing elements and due to the change in environment, there is a common trend of both Fig. 2 and Fig. 3 for corresponding terrains.

VII. LESSONS LEARNED

Throughout our work using Popper to reproduce the experiments mentioned in this paper, one of the main takeaways that we learned is the difficulty involved in automating an experiment that was not implemented with reproducibility in mind. In our case, we had the opportunity to closely work with the original authors of the experiment, however having access to the authors of the original experiments is quite unlikely. Focusing on reproducibility from the start (e.g., using the Popper convention or other reproducibility tools) makes it easier to obtain a versioned, automated, and portable pipeline that others can easily re-execute.

Furthermore, our experiments were conducted in Cooja, which has the option to run without a GUI. However, this is not the case for other GUI-based network simulation tools. Experiments implemented using platforms that are exclusively GUI-based are much harder to automate, since they can't be ran in a command-line environment. A command-line interface not only helps the process of reproducibility but is required by many reproducibility tools.

As we strived for portability across different hardware and operating systems, we encountered some limitations. For example, in the process of automating the experiments, we used Docker to execute the dependencies our experiments relied on. Throughout our experience we encountered issues using Docker on our Windows system which resulted in having to use a Linux based operating system. Further, when creating the Docker files, we had to make sure that the images we were using were up to date and maintained, otherwise our environment would not be fully functional.

VIII. 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 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] 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.

[2] 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.

[3] S. Mansfield, K. Veenstra, and K. Obraczka, "TerrainLOS: An outdoor propagation model for realistic sensor network simulation," *Modeling, analysis and simulation of computer and telecommunication systems (mascots), 2016 ieee 24th international symposium on*, IEEE, 2016, pp. 463–468.

[4] 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.

[5] 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.

[6] "Ns-3," *ns3 RSS*. Available at: https://www.nsnam.org/.

[7] "Mininet," *Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet*. Available at: http://mininet.org/.

[8] "Instant contiki and cooja," *Get Started with Contiki, Instant Contiki and Cooja*. Available at: http://www.contiki-os.org/index.html#why.