Reproducible Computer Network Experiments: A Case Study Using 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@cs.ucsc.edu

Sam Mansfield
UC Santa Cruz
smansfie@ucsc.edu

Kerry Veenstra
UC Santa Cruz
veenstra@ucsc.edu

ABSTRACT

Computer network research experiments can be broadly grouped in two categories: simulated and real-world experiments. Simulation frameworks and experiment testbeds, respectively, are commonly used as the platforms on which these experiments are carried out. In many cases, given the nature of computer networks experiments, properly configuring these simulation and real-world platforms is a complex and time-consuming task, which makes replicating and validating research results quite challenging. This complexity can be reduced by leveraging 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 implement the experiments corresponding to two published research articles. Using Popper, we develop two workflows 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.

ACM Reference Format:
Andrea David, Mariette Souppe, Katia Obraczka, Ivo Jimenez, Sam Mansfield, and Kerry Veenstra. 2019. Reproducible Computer Network Experiments: A Case Study Using Popper. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 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 about the importance of being able to reproduce research results to better understand conveyed ideas and further improve upon them.

Replicating scientific experiments, however, 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. Many experiments in this field rely on expensive hardware and software. While simulation tools greatly facilitate conducting experiments when compared to real hardware testbed experimentation, rerunning an experiment from scratch can be strenuous. Network simulation experiments often come with the cost of extensive software configuration and package installation upon attempting to reproduce previous results. In addition, even with a correct setup there might still exist uncertainty whether results are reproduced correctly. In this paper, we make a case for a systematic approach to experimental reproducibility applied to network simulations. Furthermore, we use examples from our own experience, with an intent for it to serve as a guideline to researchers and students.

In addition to validating the credibility of scientific papers and their results, reproducing networking experiments has also been used as a hands-on way to teach both fundamental and advanced concepts in computer networking [2]. When teaching a new topic, educators want students to engage in the particular subject matter rather than the daunting task of setting up an environment. This educational aspect could be improved by using a tool that would enable students and educators to easily create and modify end-to-end workflows to make learning more accessible to students.

Another motivation behind making the case for experimental reproducibility in networking research is based on our own experience as members of an academic research lab. Often times junior students help and eventually may take over the work of more senior students who are soon graduating or have already left the university. Instead of reinventing the wheel, it is in the interest of the lab for the new students to improve and build on top of previous work while leveraging as much of it as possible. However, replicating someone else's work is challenging and often impossible. This is especially prevalent in computer networking experiments where most of the experiment setup is performed manually with little or no documentation. The current state of practice of setting up simulation experiments for researchers and practitioners involves,

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

among other things, obtaining and keeping track of large amounts of data from various datasets, installing required software packages and libraries, and setting up the appropriate environment. These steps are often left undocumented because they usually involve consulting multiple resources along with repeated trial and error attempts. Consequently, they become not only very tedious and time consuming, but also prone to errors. As such, having a systematic approach to creating an experimental pipeline that researchers could easily modify when conducting and reproducing experiments will be a significant step forward towards more rigorous scientific research.

A recently proposed reproducibility tool named Popper introduces a convention for creating experimentation pipelines which are easy to reproduce and validate [3]. In order to show the suitability of the Popper convention in the experimental networking domain, we document our experience of automating the execution and re-execution of two network simulation experiments presented in [4,5]. One of the main reasons we chose these two papers was because we had the help of the original authors available to us. As a result, we were able to obtain all their original scripts and notes for reproducing their existing experiments. Additionally, we met with the authors several times to understand how the scripts map to what is reported in the original papers. This paper details our experience with the goal of serving as a reference to other researchers seeking a way to make their experiments reproducible. The contributions of our work include:

- Applying Popper in the domain of computer networks, more specifically simulation experiments.
- A methodology template for others to create reproducible.
- Lessons learned.

The remainder of the paper is organized as follows, Section 2 gives a brief introduction to Popper, the tool that is used to help make networking experiments reproducible. In Section 3, we describe the networking experiments that we reproduce using Popper as well as the network simulation platform we use, while in Section 4, we describe how each experiment was conducted originally, i.e., prior to using Popper's reproducibility model. Section 5 presents experimental results under Popper and compares them with original results. Lastly, in Section 6, 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.

2 POPPER

Popper is a convention for creating reproducible scientific articles and experiments [6]. 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 Popper-compliant or "popperized" experiment. To achieve reproducibility, Popper uses pipelines containing shell scripts that execute the original experiment. An example of set of steps that an experimenter can follow to help achieve reproducibility are the following:

1. Experimental design and pipeline definition;
2. Selecting Software selection;
3. Creating of environment using Docker;

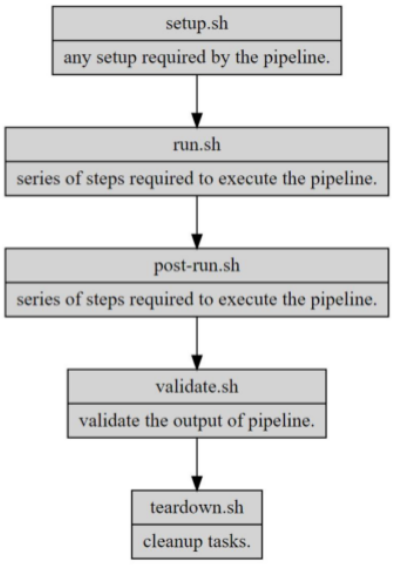

Figure 1: Automation workflow for an experiment.

4. Writing experiment scripts and parameter sweeps;
5. Writing analysis using python tools.

The Popper pipeline consists of five stages: setup, run, post-run, validate, and teardown. 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. Figure 1 shows the skeleton workflow for a Popper experiment using the "popper workflow" command. We note that each experiment may vary and that not all stages are needed for every experiment.

In our case, for example, the simulation experiments we reproduced were made to run in a virtual environment called Instant Contiki. For this reason, we needed a Popper pipeline that could run an entire operating system. To achieve this, we used Docker, a DevOps tool that packages applications and environments into containers. Docker allowed us to create an image of the Contiki operating system that contained all the libraries and dependencies needed to run it as just one package inside our pipeline. This feature of Popper that allows the use of DevOps tools makes it an advantageous convention, which we will demonstrate in the sections that follow.

3 NETWORK SIMULATION EXPERIMENTS

3.1 Network Simulation Platforms

There are a variety of network simulation platforms such as NS3, MiniNet, and Cooja. NS3 [7] is an open source discrete event network simulator that is widely used for simulation environments

for network research. Its goal is to provide scalability and ease of use for a variety of networks. Mininet [8] is also an open source simulation tool that provides a virtual network for interacting with Software-Defined Networking applications using OpenFlow. Cooja [9] 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 [9]. Although each of these network simulators is a popular choice in the networking field, the experiments we are working with are conducted in Cooja, as it allows for inclusion of simple radio propagation models.

3.2 TerrainLOS

The first experiment we have reproduced in this paper is based on TerrainLOS [4]. 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 nodes that are visible in an area from all nodes on a map. For example, 100% ACV means that every node is visible to all other nodes, which further implies the presence of a flat terrain. In their 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 in this paper focus on automating the execution and re-execution of Experimental Connectivity simulation in [4]. The purpose of this simulation is to experimentally evaluate the accuracy of connectivity results based on the models earlier presented by the authors in [4]. The connectivity results are plotted using the Average Cumulative Visibility metric and population size.

3.3 Sensor Network Deployment Over 2.5D Terrain

TerrainLOS has been used to evaluate the sensor placement algorithm proposed in [5] that aims at optimizing visual coverage in deployments over 2.5D terrain. 2.5D terrain is defined as using 2-dimensional rendering techniques such as the sensor placement algorithm and using controls in 3-dimensional space such as the terrains. It is named 2.5D terrain as it is not quite 3-dimensional but it is using features of 2-dimensions and 3-dimensions. 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, each new run of the experiment involved initializing a script with parameters such as number of nodes, intended transmission range of the nodes, and the desired terrain, then running the script, analyzing the results, and repeating these steps multiple times until the results are reasonable.

Additionally, the experiment in the paper required pre-installing an associated program containing a graphical user interface (GUI) that required familiarity with its features from the user. After extensive manual configuration and initialization of the parameters mentioned above, running the script and waiting for the final results was a repetitive and time-consuming task. Since each new experiment had to be configured and re-run a number of times for accurate results, the student or researcher had to be present in front of their computer throughout the duration of the process. Finding a way of automating this process and avoiding using a GUI was imperative.

4 RECONSTRUCTING EXPERIMENTS USING POPPER

4.1 TerrainLOS

TerrainLOS is intended to run in Cooja, the network simulator for the Contiki operating system. In order to run TerrainLOS, without using Popper, a researcher would have to go through several steps when attempting to replicate the results in [Sam's Paper]. First, they would have to download Instant Contiki, a development environment for the Contiki operating system, and install a virtual machine to run it. Once the user has logged in and started the Cooja simulator, they would have to download the necessary files, libraries, and dependencies needed to run the TerrainLOS propagation model. Lastly, they would have to create a jar file of TerrainLOS and load it into Cooja to run the simulations. 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. Similarly to our experience, the researchers or the reviewers of the project may find that after compilation there are a few necessary files or modules missing that were not part of the set-up instructions provided by the authors. However, opposed to our particular case, 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 generally cumbersome if not impossible.

Popper provides a significantly more effortless way to reproduce someone's experiment without the need of having the original author explain the steps needed for the procedure. Usually, the author would tailor their code in a way that follows the Popper convention from the start. However, making an experiment Popper compliant in retrospect is possible as well. We want to show this by detailing the steps taken to make Experimental Connectivity simulation of TerrainLOS Popper compliant.

First, in the implementation of the Popper pipeline, two stages were generated – the run stage and the post-run stage. Although in this particular experiment the setup, validate, and teardown stages were not used, the workflow for other experiments may differ. In our pipeline, the run stage takes care of setting up the Instant Contiki and Cooja environment. Since Instant Contiki requires a virtual machine to run and Cooja is usually used with a GUI, the setup of the two was accomplished with the help of Docker containers. Docker creates an image of the Contiki operating system including the Cooja simulator. Once the virtualization of the Contiki system

is finished, the main task of the run stage is to execute the author's script that takes ACV and population size as inputs. The original simulation experiment was run using population sizes of one, ten, thirty, and eighty, and ACVs ranging from one to hundred percent with increments of ten. The same input arguments are used for the reproduced experiment as well. After the script has been executed, the output of these runs is saved in log files, which are read in the post-run stage with another script written by the author. The results are then graphed and saved in an image file as output. 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 Sensor Network Deployment Over 2.5D Terrain

When first running the experiment [5], 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. Fifth, those N simulations are run and lastly the Cooja.testlog are outputted into the output folder to further evaluate the final result. Listing 1 shows an example Popper pipeline for this experiment.

Listing 1 Sample contents of a Popper repository.

```
paper-repo
| README.md
| .popper.yml
| pipelines
|   |-- myexp
|   |   |-- setup.sh
|   |   |-- run.sh
|   |   |-- post-run.sh
|   |   |-- scripts/
|   |       |-- sim_config.yaml
|   |       |-- sim_template.csc
|   |       |-- create_sim_files.csc
|   |   |-- simulations/
|   |   |-- output/
|   |    -- contiki/
|   |       |-- {java files for exp}
| paper
|   |-- build.sh
|   |-- figures/
|   |-- paper.md
|   |-- paper.pdf
|    -- references.bib
```

5 RESULTS

5.1 TerrainLOS

The simulation experiment titled Experimental Connectivity in [4] outputs a graph depicting the percentage of connected networks based on Average Cumulative Visibility and population size. This graph can be seen in Figure 2. Intuitively, population size of 80 has the highest percentage of connected networks from ACV ranging from zero to hundred percent. The authors of [4] explain that this is because a larger population can bypass obstacles in the terrain (e.g., mountain) more likely than a smaller population. For this reason, the percentage of connected networks drop as the populations size decreases.

In our reproduced experiment output, depicted in Figure 3, a similar graph is seen. The reproduced experiment is not an exact copy of the original. This is because the experimental simulation outputs for Experimental Connectivity are intended to be probabilistic and vary across multiple runs. It is possible to generate the exact graph using the original simulation logs from the author, but we wanted to showcase the re-execution of the pipeline from the start of the experiment. We still observe the general trend in the reproduced results. Population size of 80 produces the highest percentage of connected networks. Furthermore, as population size decreases, the percentage of connected networks decrease as well. This trend indicates a successful reproduction of the experiment.

5.2 2.5D Deployment on TerrainLOS

Similar to Experimental Connectivity, the results of [5] are obtained in a form of a graph. The output of the original paper can be seen in Figure 4, while the output of our reproduced experiment is shown in Figure 5. Figure 4 shows results for every data point calculated for the average of ten nodes in random starting positions on specified terrain [5]. Furthermore, the graph illustrates each communication radius from 130 to 170 with increments of ten for the given terrains.

In the graph in Figure 5, we can see that the outputs 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 Figure 5 are higher than the values in Figure 4. This difference is because the original paper used a custom, synchronous simulator that was programmed in C++. Since then, the author of

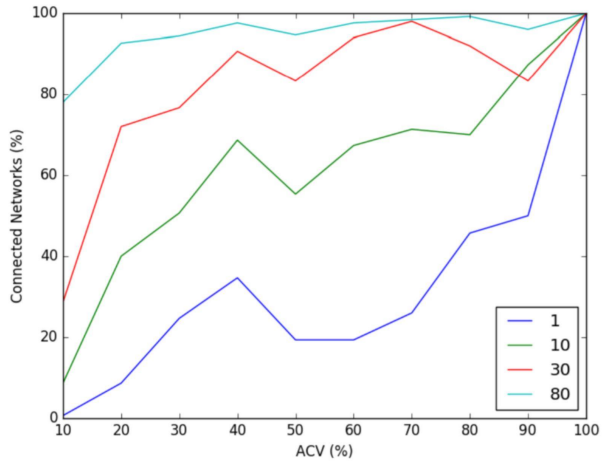

Figure 2: Original results from the Experimental Connectivity experiment in [4].

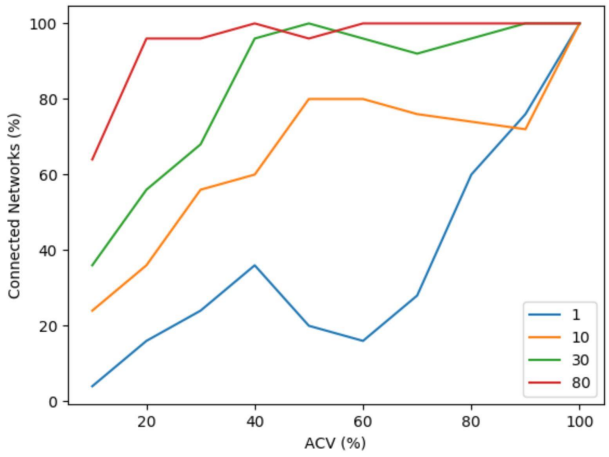

Figure 3: Reproduced network connectivity results using Popper.

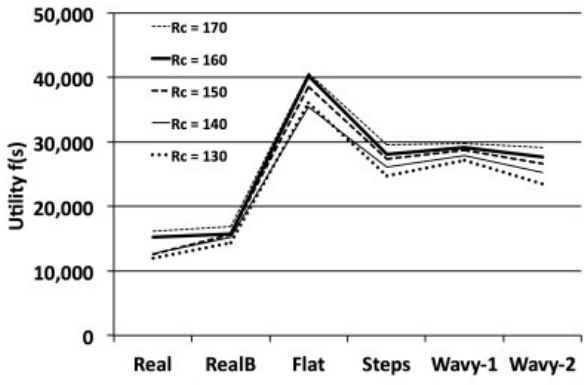

Figure 4: Original results from the 2.5D Terrain Experiment.

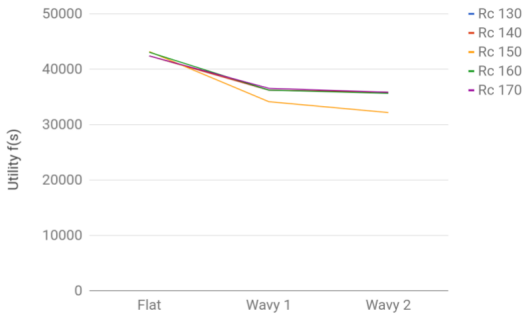

Figure 5: Reproduced results using Popper.

the experiment decided to switch environments. For this reason, the experiment has been translated into a Cooja environment as a new Java model in the event-driven simulator. Despite missing elements, due to the author's decision, the trend in both Figure 4 and Figure 5 is uniform.

6 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 network experiments. However, having access to the original authors is quite uncommon. Even with the opportunity of consulting with the authors, reproducing their experiment was an extensive task as they have made a few changes to their work since publication. This further shows how 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.

As we strived for portability across different hardware and operating systems, we encountered some limitations. For example, our experiments were conducted in the Cooja network simulator, 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 cannot run in a command-line environment. A command-line interface not only helps the process of reproducibility but is required by many reproducibility tools. Furthermore, in the process of automating the experiments, we encountered issues using Docker on a Windows system and needed to switch to 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.

7 CONCLUSION

Experimental reproducibility is an essential component of scientific research. However, unlike other disciplines in the sciences, reproducing experimental results in the field of computer science and

engineering has not been part of common practice for a number of reasons. This includes the fact that it is a fast evolving field and re-creating the original experimental environment from the ground up is often too complex and sometimes impossible. In this paper, we reported our experience using a recently proposed tool called Popper which employs a systematic approach to automating the experimental process, including experimental setup, (re-)execution, data analysis, and visualization. We showcase how Popper can be used to facilitate experimental reproducibility in the experimental computer networking domain. We hope our work will provide a workflow template to guide network researchers and practitioners towards making experimental reproducibility part of the best practices in the field.

REFERENCES

[1] J. Kurose, "Dear colleague letter: Encouraging reproducibility in computing and communications research," *National Science Foundation*, Oct. 2016.

[2] L. Yan and N. McKeown, "Learning networking by reproducing research results," *ACM SIGCOMM Computer Communication Review*, vol. 47, 2017, pp. 19–26.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] "Ns-3," *ns3 RSS*.

[8] "Mininet," *Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet*.

[9] "Instant contiki and cooja," *Get Started with Contiki, Instant Contiki and Cooja*.