NEAT: Network Experiment Automation Tool

Andreas Schmidt, *IEEE Student Member* and Thorsten Herfet, *IEEE Senior Member*
Telecommunications Lab
Saarland Informatics Campus, D-66123 Saarbrücken, Germany
Email: {andreas.schmidt, herfet}@cs.uni-saarland.de

***Abstract*—Executing network experiments is crucial for validating results of networking research, but the process is often highly manual and seldomly reproducible. This slows down the innovation process, and peer reviews cannot be carried out appropriately, as it is hard to understand and reproduce the system-under-test. This paper presents an abstract view on network experiments and sketches the design of NEAT, a first prototype to automate network experiments in physical networks using state-of-the-art automation technologies. Using NEAT, experiments can be defined in a transparent human- and machine-readable manner allowing automated execution for higher efficiency, flexibility and reproducibility.**

***Index Terms*—SDN/NFV deployments in datacenter, ISP, industrial, and campus networks, SDN/NFV testbeds and emulation platforms**

I. Introduction

Research in network design and implementation comes with a significant portion of time spent on executing evaluations to prove viability of novel approaches or compare different solutions in different scenarios. The reviewing process, when done right and following approaches from [1], [2], requires to completely and precisely rebuild the scenario and validate the results. Even though networking is a systems rather than a computational research domain, the rules mentioned in [3] apply, and are going to be referred to in the following paper as §1 to §10. We especially deal with the rules "for every result, keep track of how it was produced" (§1), "archive the exact versions of all external programs used" (§3), "version control all custom scripts" (§4), and "record all intermediate results, when possible in standardized formats" (§5).

Establishing a physical testing environment is costly and time-consuming, so that many scientists use fully virtualized networks, such as GENI [4], or network simulators such as OMNnet [5] or Mininet [6]. This leads to either quick simulations that might lack precision, suffer from resource limitations and relation to physical setups, or extensive testbed setups that are costly to build and maintain. In this situation, we see a lot of potential in using recent advances in network softwarization and automation to run high fidelity experiments with tolerable effort.

The contribution of this paper is twofold:

- We identify an abstract structure of a network experiment, describing an experiment in a human- and machine-readable way, to foster reproducible research.
- We present NEAT, a prototype tool for executing network experiments in automated network environments.

II. Network Experiments

As in every other experimental domain, network experiments run in three phases, namely preparation, execution and post-processing. The preparation usually involves the following steps: Firstly, the network architecture is defined, which means that the topology is implemented by manually connecting devices or adding links in virtual environments. This includes a definition of link characteristics, including latency (delay, jitter), reliability (loss, reorder, corruption) and capacity (data rate). In a second step, the end-hosts are added and equipped with the services or applications used to carry out the experiment. This involves again some wiring effort, software installation, and additional system configurations.

When dealing with experiments in the domain of SDN, the next step is usually the deployment of a well-defined network controller. This step also includes initial communication between the SDN nodes and the controller instance, so that the setup is already done before the experiment.

After this preparation phase, the execution of the experiment starts by either triggering a function inside the network controller or starting one or more network applications running on end hosts. Following the experiment, all the execution data has to be gathered, which comes in various forms e.g. packet traces, log files, or tables. Following §5, this should be done using standardized formats, such as `.pcap` or `.csv`. The subsequent analysis should then follow rules §2, §6, §7, §8, §9 and §10, but this is out of scope for this paper.

III. Network Experiment Automation Tool

Now that we have defined an abstract pattern for network experiments, we want to automate this by exploiting different technologies for network automation.

A. Experiment Description

The first component we need to make experiments reproducible, is a human-readable format to describe the aspects that differ between experiments. Thereby, we can ensure that the origins of results are tracked (§1) and archived using version control (§4). We chose to use YAML [7], as it is also used in the solution we chose in Sec. III-D, but it could as well be done in other languages as JSON, which is less expressive, or XML, which is highly verbose and hard to write by hand. An example for a straightforward RTT measurement experiment is depicted in Fig. 1. Here we see how the involved hosts, controller as well as server and client application are specified, together with the links and their characteristics.

```
controller:
  minion: ctrl.uds.on
  image: registry.uds.on/LARN/Ryu:v1.0
  args: --relaying=True stp
links:
  - minion: n1.uds.on,
    interfaces:
      eth0:
        bandwidth: '10Mbps',
        delay: '20ms',
        ...
server:
  minion: h2.uds.on
  image: registry.uds.on/LARN/rtt:v0.7
  args: --server=True
  ip: 10.5.1.21/24,
  mac: 'AB:CD:EF:01:23:67',
  port: 8081
  result: /var/rtt.csv
client: ...
```

Figure 1. rtt.yml

B. Software-Defined Networking

Running network experiments comes with applying specific routing and forwarding policies. The best way to make this transparent and reproducible is by using an SDN controller that implements the desired behavior. Our experiment uses Ryu [8], which is a lightweight controller implementation in Python, but one could as well employ any other controller, as long as it comes in form of a Docker [9] container.

C. Network Function Virtualization

For the applications on the end hosts and intermediate network functions, we are again using Docker. Thereby all applications and libraries are bundled together so they can be installed in a single action. This ensures §3, as the version of any software that was involved in the experiment is archived. In contrast to other virtualization techniques, such as virtual machines, containers provide a lightweight and fast alternative, which integrates nicely with softswitch solutions.

D. Configuration Management (CM)

While SDN enables us to reconfigure our network operation easily via packet-manipulation and dynamic forwarding, it does not allow us to orchestrate the systems that are part of the network. For data centers, this is an additional crucial part of an automation strategy, which is also true for experiment automation. The relevant tasks are for instance moving files to and from a system, starting services and applications, as well as running arbitrary shell commands. We particularly use the modules involved with deploying and integrating Docker containers, executing commands to set link parameters and retrieving the experiment results. There are a lot of open source and proprietary solutions for CM available, amongst the most popular ones are SaltStack, Puppet, Chef and Ansible. We picked SaltStack [10], because it is open-source, is implemented and extensible via Python and integrates smoothly with the rest of our tools.

E. Version Control (VC) and Continuous Integration (CI)

Finally, a fully automated solution also includes the network software development process. This can be done using version control and continuous integration [11] tools, where the software is stored, build and archived for future inspection. This ensures §4 and §3 so that we can fully reconstruct an executed experiment from the used artifacts. In our lab, we are intensively using a self-hosted version of GitLab CI [12] to track the state of our experiment components.

IV. Conclusion

This paper presented NEAT, a prototype implementation used to enable reproducible networking research within automatized environments. While we are entering an age of fully automated networks within data center and ICT environments, we should also make use of these technologies for network experiments. Finally, we also adhere to §10, and will publish NEAT[1], which is part of the *OpenNetworking project at Saarland Informatics Campus*[2].

Acknowledgments. This work was supported by the German Research Foundation (DFG) as part of the priority programme SPP 1914 "Cyber-Physical Networking" under grant HE 2584/4-1.

References

[1] ACM, "Result and artifact review and badging." https://www.acm.org/publications/policies/artifact-review-badging. Accessed: 2017-07-04.

[2] "Reproducing network research | network systems experiments made accessible, runnable, and reproducible.." https://reproducingnetworkresearch.wordpress.com/. Accessed: 2017-07-04.

[3] G. K. Sandve, A. Nekrutenko, J. Taylor, and E. Hovig, "Ten Simple Rules for Reproducible Computational Research," *PLoS Computational Biology*, vol. 9, no. 10, pp. 1–4, 2013.

[4] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments," *Elseview Journal on Computer Networks*, vol. 61, pp. 5–23, 2014.

[5] "OMNeT++ Discrete Event Simulator." https://omnetpp.org/. Accessed: 2017-07-24.

[6] "Mininet: An Instant Virtual Network on your Laptop (or other PC)." http://mininet.org/. Accessed: 2017-07-24.

[7] "YAML Ain't Markup Language." http://www.yaml.org/start.html. Accessed: 2017-07-24.

[8] "Ryu SDN Framework." https://osrg.github.io/ryu/. Accessed: 2017-07-04.

[9] "Docker - Build, Ship, and Run Any App, Anywhere." https://docker.io/. Accessed: 2017-07-04.

[10] "SaltStack intelligent orchestration for the software-defined data center." https://saltstack.com/. Accessed: 2017-07-04.

[11] M. Fowler and M. Foemmel, "Continuous integration," *Thought-Works http://www.thoughtworks.com/ContinuousIntegration.pdf*, 2006.

[12] "Code, test, and deploy together with GitLab open source git repo management software." https://about.gitlab.com/. Accessed: 2017-07-04.

[1] http://neat.larn.systems

[2] https://www.on.uni-saarland.de