Visualization of Network Emulation Enabled by Kathará

Marcel Großmann and Duy Thanh Le

Computer Networks Group, University of Bamberg
Bamberg, Germany
Email: marcel.grossmann@uni-bamberg.de | duy-thanh.le@uni-bamberg.de

***Abstract*—In network research, reproducibility of experiments is not always easy to achieve. Infrastructures are cumbersome to set up or are not available due to vendor-specific devices. Emulators try to overcome those issues to a given extent and are available in different service models. Unfortunately, the usability of emulators requires time-consuming efforts and a deep understanding of their functionality. At first, we analyze to which extent currently available open-source emulators support network configurations and how user-friendly they are. With these insights, we describe, how an ease-to-use emulator is implemented and may run as a Network Emulator as a Service (NEaaS). Therefore, virtualization plays a major role in order to deploy a NEaaS based on Kathará.**

***Keywords*—Network Emulator; Visualized Kathará; Containerization**

I. Introduction

It is a norm in the science community that experiments proposed by a pioneering paper must be reproduced and examined closely and thoroughly before considering to be valid and applied to the real world. For example, in medicinal chemistry or drug discovery fields, the literature must "provide adequate experimental detail, and Reviewers have a responsibility to carefully examine papers for adequacy of experimental detail and support for the conclusions" [1] or could result in legal issues because of fake or substandard results. Transparent, deploy-able, and reproducible papers are not yet prevalent in the network research community since they must provide access to all software, scripts, input data, and test-beds used in these experiments. The first three items are relatively easy to share, given the rise of public cloud-based hosting platforms for version control and collaboration (i.e., *git*) which offers access without restrictions. But a flexible, easy-to-deploy, and cost-effective test-bed is not trivial to obtain. A genuine test-bed, though having indisputable authenticity given by the usage of physical hardware like switches, routers, and sensors, also has a number of issues associated with it. It is costly and hard to scale as the expense increases relatively as the size of the test network expands, not to mention the cost of mistakes caused by implementation errors, accumulating to the lack of reproducibility. Network emulators, in contrast, offer lower-cost and high-effective alternatives.

There are several prototyping programmable emulators like GNS3[1], Mininet[2], Containernet[3], and Kathará[4], and they all have a fair share of pros and cons. For example, GNS3 offers network researchers various network experiments by utilizing multiple VMs to represent switches, routers, and hosts and interconnect them via virtual interfaces. That emulator seems appealing at first glance since any practical topologies can be easily replicated and evaluated, but VMs are too heavyweight due to the locally available resources limitation [2]. Mininet or Containernet, in contrast, embrace the concept of Container-based Emulation (CBE) by using lightweight, OS-level virtualization techniques to substitute VMs as emulated network elements, thus providing cheap experimenters with flexibility, and reproducibility [2, 3]. However, they all demand specific hardware, software, and OS and further system configurations in order to be installed and work properly, not to mention the steep learning curve to comprehend these tools. And lastly, all of the mentioned emulators are single-machine tools, thus restricting the testing topologies to a certain extent due to local resource confinement.

Therefore, we propose a vision of a cloud-based, multi-tenancy Network Emulator as a Service (NEaaS) platform by integrating current technologies into an open-source network emulator that allows users conveniently deploy and moreover, test a wide range of network designs. We demonstrate a first generation prototype of that platform to exhibit its capabilities compared to the current network emulators. Our prototype is made publicly available on Github under *uniba-ktr/KaaS*[5].

II. Foundation

A. Containerization

Hypervisor-based virtualization is highly regarded by cloud providers since it supports multiple VMs - each has different shares of CPU, memory, storage, and network interface, runs an isolated OS with many binaries (i.e., system libraries and application packages) and different software applications - to operate simultaneously in a single physical server. Though delivering several advantages (e.g., adaptability, portability, efficiency), it also has some drawbacks: heavyweight as each

[1] https://docs.gns3.com/docs/
[2] https://github.com/mininet/mininet
[3] https://github.com/containernet/containernet/
[4] https://github.com/KatharaFramework/Kathara
[5] https://github.com/uniba-ktr/KaaS

VM runs an independent OS that adds overhead to the storage footprint; slow startup time, since each VM has to proceed through all steps in a typical boot process of bare-metal server; inelasticity due to fixed resource distribution (e.g., v-cores, RAM, storage size). These shortcomings make hypervisor-based virtualization less favorable compared to the new approach, namely containerization when designing an effective network emulator [2, 4].

Containerization, on the contrary, is built on top of the two fundamental features of the Linux Kernel: namespaces and cgroups. The former allows the partition of its own Kernel resources, giving a process isolated resources such as: network stack, filesystem, process ID, and user ID. The latter allows user to limit, monitor, and isolate the amount of system resources such as CPU, memory, I/O rates of one or multiple processes. That results in a twofold increase in efficiency: high optimization of available resources, while minimizing overhead, along with minimal startup time. With these unique characteristics, containerization is considered by many as a compelling-based technique for a lightweight, cost-effective network emulator [2, 5].

B. Container-based Emulation

Realizing the potential of OS-level virtualization, several notable network emulators are designed based on such technology, namely Mininet, Containernet, and Kathará. Mininet[2], "...a network emulation orchestration system...", can mimic many complicated network experiments with a high level of fidelity. A "host" in Mininet is a single user-space process sharing the same Kernel, process IDs, and filesystem with other Mininet hosts. Mininet also utilizes various Linux network tools (e.g., ip-netns, tc) in the form of Python classes for ease of development to create virtual networking interfaces and configurable links. Mininet is lightweight and agile, but there are some considerable downsides: limited options regarding resource constraints; no support of Docker containers as host type; steep learning curve since the hosts and their links are programmed in Python code. Consequently, Peuster et al.[3] develop Containernet, a container-enabled host version of Mininet. It allows users to employ Docker containers as emulated hosts and provides better options for restricting resource usages.

Kathará [6], successor of Netkit, also supports Docker containers as emulated hosts. However, Kathará obscures the actual implementation of network devices and links by using the concept of network scenarios. Network devices, links, and any required configurations are described in text files (e.g., lab.conf, device.startup). Therefore, both novice and expert users can build, run, and test any network experiments without spending much time and learning effort.

III. Architecture

A. The Architecture of the Prototype

As presented in Figure 1, the prototype consists of four services, where each one is a customized Docker container, and the links depict the dependencies between these services.

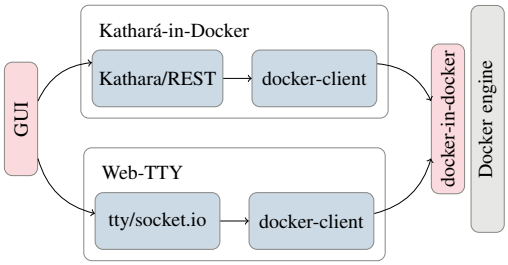

Figure 1. Single user prototype

As shown, the GUI service is a containerized web application allowing users to interact (e.g., creating devices, attaching devices to collision domains) with the API endpoints provided by the Kathará REST API service (i.e., emulator core). Moreover, users can remotely connect to any emulated hosts to run additional commands through the web sockets exposed by the Web-TTY container. And both Kathará REST and Web-TTY services depend on a "Docker-in-Docker"[6] (i.e., *DinD*) container serving as the Docker engine. Finally, the Kathará REST, Web-TTY, and *DinD* containers compose a single, completed network emulator. The prototype favors the microservices over the monolithic architecture due to several causes: allowing developers to implement and deploy new core functions without getting in the way of one another; it fits well with the cloud-native approach as a single micro-services emulator will be assigned on-demand to a single user; offering better security because each user has his dedicated *DinD* so that his emulated hosts are entirely isolated from others.

B. The System Workflow

Figure 2 describes the sequence diagram for creating and running a Kathará lab in the prototype. First, users add various networking devices on the GUI, then users make an HTTP POST request that holds information of these devices to the *create_lab* API endpoint. The emulator core checks the validity of the submitted JSON, then creates a Kathará lab and its configuration files within the *KinD* component itself. Thereafter, the emulator core responds with a *state_msg* containing the current state of the submitted lab. Next, if the lab is successfully created, users can start it by calling the *start_lab* endpoint. As a result, the emulator core tries to start the requested Kathará lab asynchronously. After that, when users check the current state of the lab by the *lab_info* endpoint and the lab was successfully started, they will receive the list of container IDs for the devices in that lab. Occasionally, if users want to run commands on any device, they can click on the icon of that device in GUI. The container ID of that device is sent to the Web-TTY service, and a terminal session via a websocket is initiated and displayed on the Kathará GUI as an HTML iframe.

C. Kathará REST API

As per documentation, Kathará does not support REST API service of any sort whatsoever, only a Python package with

[6]https://hub.docker.com/_/docker

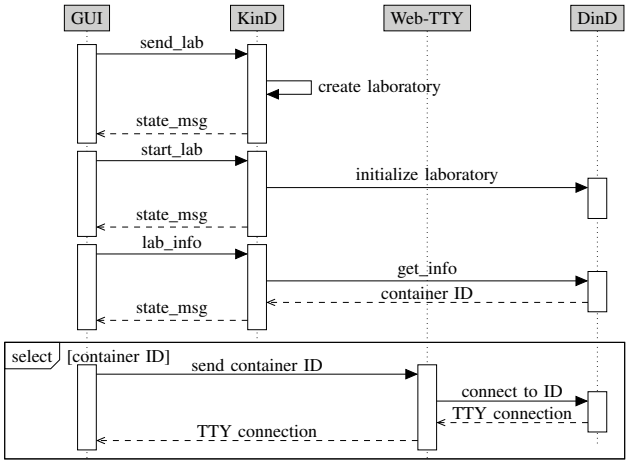

Figure 2. Micro-service interactions

various APIs that allows developers to interact with it within Python apps[7], thus severely restricting the capabilities of the network emulator. Therefore, a new RESTful API interface is introduced and implemented to allow remote control of the Kathará emulator from another service, particularly in this design, the Kathará UI. The fastAPI framework[8] is chosen due to its robustness, agility, and support of better libraries such as Pydantic[9] or type hints. Moreover, additional Python wrapper classes (e.g., LabWrapper, LabController, KatharaWrapper) are introduced to mask the complexity when interacting with the core libraries. Consequently, the new Kathará REST API provides an API endpoint for each command of Kathará. For example, Listing 1 shows the implementation of the API endpoint to create a lab:

Listing 1. Lab creation API endpoint

```
@app.post("/lcreate", response_model=Message)
async def create_lab(info: Laboratory):
    log.info("Create Lab {}".format(info.json()))
    result = LabController().gen_lab(info)
    return result
```

D. Kathará-in-Docker

Furthermore, both the Kathará emulator and the new REST API framework are incorporated in a single Docker image, thus creating a concept of *Kathará-in-Docker* (i.e., *KinD*). This novel capability highly contributes to the NEaaS vision, as an emulator sandboxed environment is easy to be deployed and scaled on-demand using available container orchestrators like Docker Swarm or Kubernetes.

IV. Evaluation

The need for reproducibility of dissertations in the research community, in general, and in networking research groups, in particular, has been resonating for a long time. Others already lay out the features which a network emulator should have so it can fulfill such demand. The experience gathered after using those emulators is summurized in the following list:

- **Scalability**: the platform should facilitate a wide range of emulation of network topologies from several to hundreds or thousands of network devices without altering the system architecture.
- **Flexibility**: It should be effortless to create any network experiments in a convenient manner.
- **Extensibility**: New capabilities or types of network devices should be integrated into the platform with ease, and should not require any modifications in the design of the platform.
- **QoE**: The time and effort a user, regardless of his ability, spent to use the tool should be minimal. A rich and productive graphical user interface is recommended.
- **Realism**: The system and its components should show the same functions found in the physical hardware. Similarly, the traffic generated by the platform should be as genuine as possible to the actual traffic.
- **Cost**: The overall costs (e.g., capital cost, operating cost) should be as minimum as possible.

Table I
Feature comparison of network emluators

	Mininet	Containernet	GNS3	Kathará	Kathará -UI
Scalability	+	+	-	+	+
Flexibility	-	-	+	+	+
Extensibility	-	-	(limited)	(limited)	+
QoE	-	-	+	(limited)	+
Realism	+	+	+	+	+
Cost	+	+	+	+	+

As a result Table I represents the achievements of the mentioned network emulators, as well as, our prototype regarding the mentioned features.

V. Conclusion & Future Works

In this paper, we demonstrate a new approach to enhance network emulator tools with modern technologies to unlock new capabilities. The main contribution is that this platform also allows network researchers regardless of their proficiency to easily reproduce any network experiments with a high level of fidelity, as well as universities or other institutes to provide better network emulators for students to learn.

However, this prototype is far from finished. A new user management module is required to provide more functions for users, such as registration, authentication, creating, and saving multiple Kathará labs. The infrastructure consumed by the NEaaS platform should also be provisioned and managed by additional services like Proxmox[10] and Ansible[11]. Then on top of that layer, a container orchestrator, presumably Kubernetes, is needed to schedule and allocate multiple containers demanded by numerous users with ease and efficiency.

[7]https://github.com/KatharaFramework/Kathara/wiki/Kathara-Python-API
[8]https://fastapi.tiangolo.com/lo/
[9]https://docs.pydantic.dev/latest/
[10]https://www.proxmox.com/en/
[11]https://github.com/ansible/ansible

References

[1] R. G. Bergman and R. L. Danheiser, "Reproducibility in chemical research," *Angewandte Chemie International Edition*, vol. 55, no. 41, pp. 12 548–12 549, 2016. DOI: https://doi.org/10.1002/anie.201606591.

[2] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12, Association for Computing Machinery, 2012, 253–264, ISBN: 9781450317757. DOI: 10.1145/2413176.2413206.

[3] M. Peuster, H. Karl, and S. van Rossem, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016, pp. 148–153. DOI: 10.1109/NFV-SDN.2016.7919490.

[4] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12, Association for Computing Machinery, 2012, 253–264, ISBN: 9781450317757. DOI: 10.1145/2413176.2413206.

[5] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys '07, Association for Computing Machinery, 2007, 275–287, ISBN: 9781595936363. DOI: 10.1145/1272996.1273025.

[6] M. Scazzariello, L. Ariemma, and T. Caiazzi, "Kathará: A lightweight network emulation system," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–2. DOI: 10.1109/NOMS47738.2020.9110351.