RO-Crate for Testbeds: Automated Packaging of Experimental Results

Eric Hauser
Technical University of Munich
Garching near Munich, Germany
hauser@net.in.tum.de

Sebastian Gallenmüller
Technical University of Munich
Garching near Munich, Germany
gallenmu@net.in.tum.de

Georg Carle
Technical University of Munich
Garching near Munich, Germany
carle@net.in.tum.de

***Abstract*—Result reproducibility, a cornerstone of open science, not only enhances transparency and trustworthiness in research findings but facilitates collaboration and accelerates scientific progress by enabling other researchers to validate and build upon existing results. In computer science, testbeds are widely used research infrastructures to create scientific results. Scientists create experiments in these testbeds typically as one-off setups to answer specific questions. The outcomes of this process are individualized experiments with a wide variety of approaches, structures, formats, or documentation. These properties hamper effective experiment validation or reuse.**

This paper proposes a concept to structure and document the outcome of testbed-hosted experiments. We describe our implementation that packages the results of a specific testbed framework, called plain orchestrating service (pos), in the standardized RO-Crate format and annotates metadata to the result files. The metadata includes information such as authorship, affiliations, software setup, hardware setup, energy consumption, and network topology. Furthermore, we demonstrate our implementation with an example measurement and publish the packaged RO-Crate to the open repository Zenodo. Our approach enables researchers to effectively organize, share, and reproduce experimental data, promoting transparency and collaboration.

***Index Terms*—Testbed, Network Experiments, Reproducibility, Result Data Management, RO-Crate**

I. Introduction

The successful validation of research results undoubtedly strengthens the trust in scientific findings. Therefore, researchers need to prioritize the reproducibility of experiments and the availability of results just as much as the results themselves. However, creating reproducible experiments is often seen as complex and time-consuming. In addition, the computer science community prefers novel over reproduced results for publication.

Fortunately, new concepts and methods have been proposed in recent years to improve the situation in the research community. The Association of Computing Machinery (ACM) came up with different badges [1] to honor the extra commitment of authors, ranging from providing experiment artifacts to explanations on how to reproduce them. Another concept in this area are the FAIR principles proposed in 2016 by Wilkinson et al. [2]. These four principles build the theoretical foundation for organizing and making data accessible to the community:
***F**indability*: Results should be assigned to a globally unique identifier and available to searchable resources.
***A**ccessibility*: Information should be retrievable using an open and free standard communications protocol. If the data becomes unavailable, its metadata should remain accessible.
***I**nteroperability*: Results should be presented in a formal, accessible, and widely recognized language for representation. Additionally, the data should reference other datasets.
***R**eusability*: A dataset should contain accurate and pertinent attributes, possess a license, and adhere to relevant community standards within its domain.
The principles also consider continuously growing data sets humans cannot analyze without machine help. Therefore, any implementation should provide a machine-readable interface.

Since the FAIR principles are high-level guidelines, they need to be converted into practical, domain-specific guidelines to become applicable within a community. This paper considers two domains: the testbed community and the data management community.

In the testbed community, the resources to run experiments, such as hardware, software stacks, or the availability of free frequency space, are essential. Specific resources enable the execution of experiments and significantly impact measurement results. Naturally, the testbed community created software frameworks to manage these resources. Testbeds, therefore, offer tooling to make resources *findable* and *accessible* [3]. There have also been efforts to make experiments *interoperable* between different testbeds [4] and *reusable* [5]. The experimental results, data formats, or structure are typically not considered by testbed frameworks.

The data management community focuses on aspects concerning the data and metadata. In testbed-driven research, we consider experimental results or derivatives thereof as data. Metadata includes additional information about the researchers and their affiliation, funding, publication, related work, etc. This data management community was one of the main contributors to creating the FAIR principles, which were implemented in different initiatives and projects in this field. A notable, recent initiative is RO-Crate [6], an approach to package research (meta-)data.

Currently, there is a gap between testbeds and data management. Researchers must convert and add metadata information manually after the execution of experiments. This paper presents an approach to connect a reproducible testbed infras-

ISBN 978-3-903176-63-8

tructure with the aspects of data management. We demonstrate how RO-Crate is used to automatically package experiment results and include relevant metadata. The metadata includes detailed information about the used resources, e.g., hardware and network topology. Additionally, we automatically record the system configuration, including the used software, and provide optional energy consumption reports. We publish an example data set, describing the output of a testbed experiment as RO-Crate. During the process, we pay close attention to conforming and fulfilling the FAIR principles.

The paper is structured as follows: Section II elaborates more on FAIR, existing testbed infrastructures, and related initiatives. Thereafter, we explain our implementation in Section III and demonstrate the functionality of it in Section IV. Section V concludes the paper and mentions future ideas.

II. Background

In this section, we introduce current efforts to implement the FAIR concept with a particular focus on testbeds.

a) FAIR: The proposed FAIR principles do not suggest a specific implementation; they instead form guidelines. Jacobsen et al. [7] discuss how concrete implementations could look like. Furthermore, they discuss chances and risks, one of them ending up with multiple incompatible implementations. Lamprecht et al. [8] discuss implementing the FAIR principles in research software. The authors note that FAIR, for example, does not cover software quality. One of the main drivers of FAIR on a European level is the European Open Science Cloud [9] (EOSC). EOSC aims to integrate existing research infrastructures and services across Europe into a federated cloud environment, enabling researchers to seamlessly access and utilize a wide range of resources and tools. In general, the Horizon 2020 programme funded various projects fostering the FAIR principles in European research. Consequently, EOSC complies with the FAIR principles. Furthermore, the CERN-hosted Zenodo [10] platform allows researchers to upload and archive scientific data. Compared to EOSC, Zenodo is a specific repository for research data and publications. All uploads on the Zenodo platform are uniquely identifiable by a Digital Object Identifier (DOI). Zenodo also allows linking GitHub repositories. This feature is particularly valuable in the field of computer science, as Git is the de facto tool for version control. The EOSC initiative and the Zenodo platform primarily cover the *findability* and *accessibility* principles. Beyond that, researchers must pack and add metadata to their data sets before publishing results.

b) FAIRness of testbeds: There are various testbed systems available to researchers for conducting experiments, such as Cloudlab [11], Chameleon [12], or Grid'5000 [13]. These testbeds are publicly available to, typically academic, users. If used correctly, these testbeds can be used to create reproducible experiments, according to Nussbaum [14]. Compared to other testbed systems, the pos methodology, proposed by Gallenmüller et al. [15], ensures the reproducibility of experiments. Experiment nodes that are part of a pos-operated testbed boot only live images, ensuring a consistent initial state for experiments after a reboot. The *everything-must-be-scripted* paradigm requires users to automate the entire experiment workflow, including system configuration, experiment execution, and evaluation. Stubbe et al. [4] build a bridge between pos and the testbed infrastructures Cloudlab and Chameleon. Their implementation allows pos-orchestrated experiments in these testbeds. As a result, Stubbe et al. enable the interoperability of the pos controller with other testbeds and demonstrate the general compatibility and flexibility of the pos methodology. This work also strengthens pos' position as a universal testbed methodology and contributes to the *interoperability* demanded by FAIR. We see reproducibility as a necessary precondition to realize *reusability*, i.e., without data being reproducible, reuse is highly limited. The pos methodology provides a superior foundation to other testbeds for achieving the FAIR data principles.

c) SLICES: Moreover, pos is part of the *Scientific LargeScale Infrastructure for Computing/Communication Experimental Studies (SLICES)* project. The main goal of SLICES is to create a European testbed infrastructure for digital experiments. Demchenko et al. [16] explain the pos integration process. Hence, the authors list key aspects of how experiments should be designed to improve reproducibility. The paper walks through the complete process, from testbed infrastructures to experiment execution and results publishing. Moreover, they discuss how different technologies can be adopted in the SLICES infrastructure.

d) RO-Crate and related approaches: Dublin Core [17] provides a set of metadata schemes to describe digital resources in a standardized and machine-readable way. Another metadata initiative is Schema.org [18], founded by Bing, Google, and Yahoo to make resources better indexable and thus easier to find. An important aspect of metadata is the cross-linking of information. One standard is JSON Linked Data [19] (JSON-LD), standardized by the World Wide Web Consortium (W3C). The involvement of the W3C highlights the importance of making data discoverable and accessible for the future of the Internet. In this context, RO-Crate [6] applies the concept of structured data to the scientific field. RO-Crate enables researchers to bundle research data alongside its metadata. This concept primarily covers the interoperability and reusability principles of FAIR. RO-Crate uses JSON-LD to describe scientific data sets. General descriptions, author information, and additional attributes can be specified per file or directory. Additionally, general parameters like related publications, funding, etc. can be put on record. Many libraries and tools for different programming languages are available.

e) FAIR experiment results: The Zenodo platform helps ensure the *findability* and *accessibility* of research data. The pos methodology provides the preconditions to support *interoperability* and *reusability*. RO-Crate can act as the missing link to create pos as a platform that provides FAIR experimental results. Currently, the pos result files and the executed scripts and their output, are collected in a result folder on the management host. This folder does not follow a well-known defined structure in the current implementation. Furthermore,

additional metadata like the researcher's information, affiliation, etc. are not automatically included in the folder.

III. Implementation

As described in Section II, we propose using RO-Crate to automatically organize experiment results. The pos controller already collects numerous metadata before, during, and after an experiment. However, the management of these experiment results does not follow any defined structure. Our prototype implementation uses the Python library ro-crate-py [20].

A. Experiment (Meta)-Data

The following subsections explain which and how we collect metadata for our implementation.

1) Author and Affiliation: Every testbed user requires an account to use the testbed. Currently, we authenticate users through an OpenID provider. This interface can retrieve additional information like full name, mail address, etc. Additionally, a configuration file, stored in the user's home directory, enables users to specify additional parameters such as their Open Researcher and Contributor ID (ORCID) [21]. Moreover, users can store their affiliation in the configuration file. As suggested by the RO-Crate specifications, we recommend users to directly cite their affiliation from the Research Organization Registry (ROR) [22]. The ROR makes research institutes unambiguously identifiable. If users have not specified their affiliation, we default to the location of the testbed.

2) Software Setup: By default, pos logs different information about the software setup of experiments. First, the pos controller saves all executed scripts during an experiment in the result folder. Concurrently, pos logs the `stdout` and `stderr` output and whether a script terminated successfully or not. In addition to the experiment scripts, pos supports variables (cf. [15]) to parameterize experiments. The pos controller records the configured variables. Moreover, since pos uses live images on the testbed nodes, the names of the booted live images are logged to the results. These images are built reproducibly in a pos experiment [15].

3) Hardware Setup: We use the pos methodology especially for network experiments. In this domain, the network topology that interconnects different nodes and the used network interface cards (NICs) are of particular interest. Therefore, we maintain a tool, named *jeppesen*, to detect the testbed nodes' hardware and the measurement network topology. We execute jeppesen as soon as the testbed topology or hardware has changed to keep this information up to date. Every jeppesen execution scans the entire testbed with all nodes and links. Finally, the scan results are stored in the testbed's internal database.

At the moment, these scans are used to simplify the planning of experiments based on the requirements of researchers. Users have three possibilities to access the hardware and topology information: Users can *(1)* directly query data from the testbed database; *(2)* use a CLI to print the hardware information; *(3)* access an interactive table on the testbed website to filter the experiment nodes by different hardware components and show a topology graph per node. Moreover, as part of our proposed implementation, we include the topology and hardware information as metadata for all experiments.

a) Hardware Information: Collecting hardware information is done individually per node. We use common tools to detect parameters like motherboard, CPU, GPU, RAM, mass storage, and NICs. The initial detection relies on the tool *lshw* [23] and includes *ethtool*, *lsblk*, and *dmidecode* for more detailed information. Due to our focus on network experiments, we provide extensive details on the built-in NICs, including the inserted transceiver modules. Finally, the detected hardware is stored in a JSON file.

b) Network Topology: We are particularly interested in the local network topology of our testbed infrastructure. For this purpose, the Link Layer Discovery Protocol (LLDP) standardized in IEEE 802.1AB [24] enables topology detection in the local network. However, LLDP operates solely within a broadcast domain, which limits its applicability in wide area networks (WANs), as routers segment these broadcast domains. We use the software daemon *lldpd* [25]. Thereby, *lldpd* periodically sends LLDP packets locally forwarded to adjacent neighbors. Concurrently, *lldpd* collects the incoming LLDP packets from neighboring nodes and stores the received information in a local database. We execute *lldpd* for approximately $5\,\mathrm{min}$ to ensure that all nodes have announced their presence. After that, we collect and analyze the neighbor information from all nodes and parse it into a JSON file containing the complete testbed topology. In this step, we can distinguish different link types and special topology structures. Based on our requirements as a network experiment testbed, we distinguish between unidirectional and bidirectional as well as switched and split links. First of all, *lldpd* treats every network interface separately. Therefore, if we observe more than one neighboring device on an interface, we assume a switched link. Beyond that, if there is a bidirectional link (LLDP packets are exchanged in both directions) to a neighboring device plus a unidirectional link to a third device, we assume a splitter setup. The term splitter setup refers to an optical tap that distributes the received optical signal between multiple hosts. In our domain of computer networks research, optical taps are useful, for example, to neutrally timestamp the traffic between two nodes on a third node that does not influence the actual measurement. Figure 1 depicts how the differentiation of bidirectional, switched, and split links works. Finally, the detected and analyzed measurement topology is stored in a JSON file.

As an additional feature, to increase convenience for the testbed users, we create topology graphs from the JSON file for every node. To reduce complexity, we draw one individual graph per node showing its connected neighbors. As an example, Figure 2 illustrates the topology graph of a node named *nodeA*. Thereby, *nodeA* has two bidirectional direct connections with *nodeB* using a $2.5\,\mathrm{Gbit/s}$ twisted pair coppper cable. Additionally, *nodeC* is connected via a $100\,\mathrm{Gbit/s}$ direct attached copper (DAC) cable. Furthermore, we observe a unidirectional optical splitter setup on the

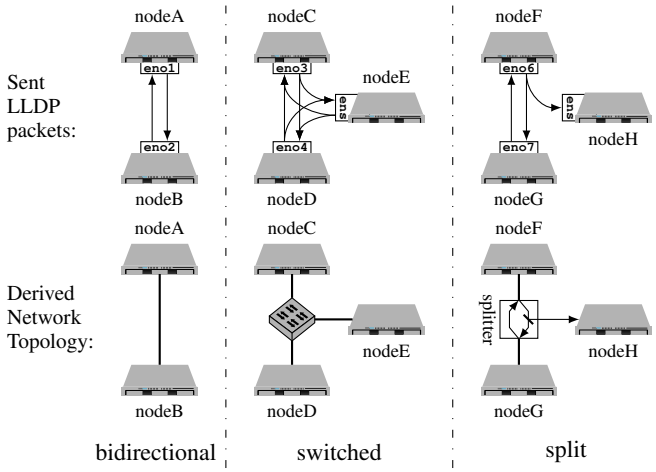

Fig. 1. Detection of bidirectional, switched, and split links

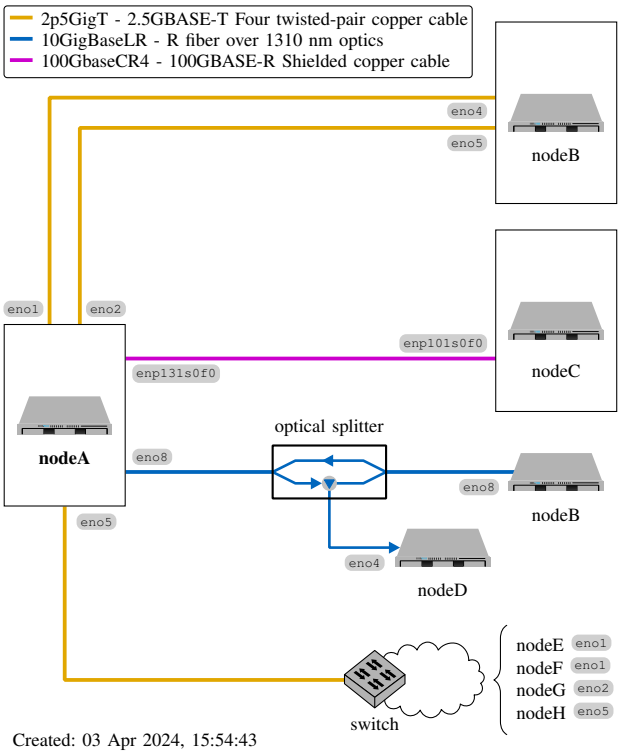

Fig. 2. Topology graph of a testbed node including an optical splitter and a switched network

10 Gbit/s bidirectional fiber link between *nodeA* and *nodeB*. In this setup, *nodeD* unidirectionally sniffs the traffic in the direction from *nodeA* to *nodeB*. Lastly, we have a switched network between *nodeA*, *nodeE*, *nodeF*, *nodeG*, and *nodeH*.

4) Energy: The pos controller supports switching power outlets that include a power meter. If a testbed node is connected through such a metered power outlet, we can measure the voltage in V, current in mA, power in W, and energy in W h once per second. During experiments, the energy measurement can be started and stopped individually per node. When enabled, the pos controller stores these values together with a timestamp in a CSV-formatted file. In our testbed, we use the *Expert Power Control 8226-1* [26] metered power outlet from *GUDE*.

B. Structure of the Result Folder

As part of this work, we propose a new structure for the result folder. The directory tree in Figure 3 lists the new structure. According to the RO-Crate documentation, the root folder's name is not defined. By default, pos uses the date and the exact time plus the microseconds of the allocation. Within the root folder, we store the `ro-crate-metadata.json` file. This file is mandatory according to the RO-Crate specification and specifies the folder's content. Furthermore, it contains meta information like author, affiliation, etc. In our proposed structure, the root folder contains four folders: *scripts*, *files*, *energy (optional)*, and *config*.

a) scripts: This folder contains the scripts executed on the allocated testbed nodes. We create a subfolder for every testbed node used in the experiment. The example structure in Figure 3 lists a script *setup* executed on *nodeA* saved as `setup.file`. The stdout and stderr outputs are stored in the `setup.stdout` and `setup.stderr` files, respectively. Finally, pos includes a `setup.status` file indicating whether the script terminated successfully or failed.

b) files: Files uploaded from a testbed node to the testbed management host are stored here. These files refer to measurement and evaluation results generated during the experiment. Again, we have subfolders for every testbed node.

c) energy (optional): If used during an experiment, this folder contains the energy consumption reports. For every testbed node that has available energy measurement results, we create a subfolder. Every individual energy measurement that is started and afterwards stopped, generates a separate CSV report.

d) config: This folder contains additional metadata about the experiment setup. As described in Section III-A3, we provide hardware and network topology information about setups. Again, we have subfolders for every node. The hardware configuration of a node is stored in this folder. Alongside the hardware configuration, the network topology is also accessible in this folder as a machine-readable JSON and a human-readable PDF.

IV. Demonstration

To demonstrate our implementation, we create a simple measurement. For this publication, we focus on the management of result files rather than the results themselves. Therefore, we reproduced an experiment of the original pos publication [15]. This experiment measures the forwarding throughput of the Linux router. In multiple individual measurement runs, we investigate two different packet sizes and increasing packet rates. In every measurement, we generate text-based throughput files. In the evaluation phase, we combine these throughput files in one final plot as PDF. Finally, all results files (text and PDF) are packaged by our implementation described in Section III and uploaded to Zenodo.

```
2024-01-01_12-30-00_123456/
├── ro-crate-metadata.json ..... RO-Crate Metadata File
├── scripts/ ..... scripts executed on experiment nodes
│   ├── nodeA/
│   │   ├── setup.file ..... script file
│   │   ├── setup.status ..... states if script exited successfully
│   │   ├── setup.stdout ..... stdout of script
│   │   └── setup.stderr ..... stderr of script
│   └── .../
├── files/ ..... result files uploaded by experiment nodes
│   ├── nodeA/
│   │   ├── throughput.txt ..... result file uploaded by nodeA
│   │   └── throughput.pdf ..... result file uploaded by nodeA
│   └── .../
├── energy/ ..... consumed energy of experiment nodes (optional)
│   ├── nodeA/
│   │   └── energy.csv ..... consumed energy of nodeA as CSV (optional)
│   └── .../
└── config/ ..... software, hardware, and topology of experiment nodes
    ├── allocation.json .. general configuration like variables, images, etc.
    ├── nodeA/
    │   ├── hardware.json ..... hardware of nodeA as PDF
    │   ├── topology.pdf ..... topology of nodeA as PDF
    │   └── topology.json ..... topology of nodeA as JSON
    └── .../
```

Fig. 3. Proposed structure of the result folder

Fig. 4. Measurement setup

A. Measurement Setup

For the measurement setup, we use two off-the-shelf servers equipped with a *Supermicro X10SDV-TP8F* motherboard, *Intel Xeon D-1518* CPU (4 cores, 2.2 GHz), 32 GB RAM, and a dual-port *Intel X552* NIC (10 Gbit/s per port). The servers run Debian Buster (*Linux* kernel v4.19). Both servers are directly connected over two 10 Gbit/s fiber links. One server acts as Load Generator (LoadGen) and the other as Device under Test (DuT). Figure 4 shows the measurement setup. The complete experiment is orchestrated by pos [15]. As described in Section II, pos enforces reproducibility by fully scripted experiments.

B. Measurement Execution

We use MoonGen [27] as a packet generator and throughput measurement tool. The LoadGen sends packets to the DuT while the DuT forwards the packets back to the LoadGen. On the DuT, we configured the Linux router to forward incoming packets from the LoadGen on the first link back to the LoadGen on the second link. We use two different packet sizes, minimum-sized 64 B packets and 1500 B-sized packets. As the second parameter, we increase the packet rate from 0.1 Mpps to 2 Mpps in 0.1 Mpps steps. Considering all permutations of packet sizes and packet rates, this experiment consists of 40 individual measurement runs. A single measurement run requires approximately 45 s. After a measurement run, MoonGen logs the measured throughput as a text file. Subsequently, the throughput files are uploaded to the testbed management host.

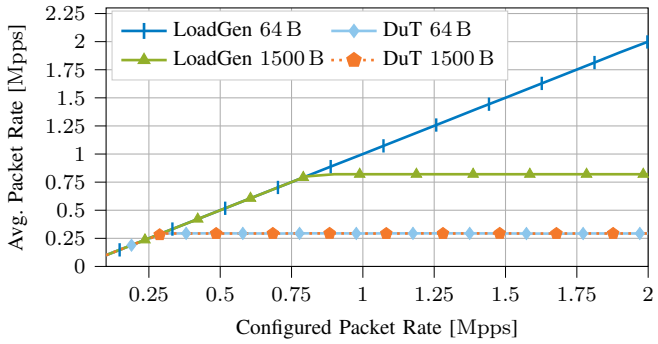

Fig. 5. Throughput of the Linux router

C. Measurement Evaluation

We run the evaluation directly after the measurement phase. The evaluation script combines all throughput files and creates a final line plot. This plot is displayed in Figure 5 and depicts the throughput of the Linux router for different packet sizes and packet rates, respectively. The x-axis depicts the configured packet rate, while the y-axis shows the achieved average packet rate. As this paper focuses on the demonstration of result data management, we describe measurement results only briefly. The two LoadGen lines state how many packets were sent to the DuT. The 1500 B line has a cut-off at approximately 0.82 Mpps because we reach the line rate of the 10 Gbit/s link. In comparison, the 64 B line continuously grows because a higher packet rate is required to saturate the link. The DuT lines show the performance of the Linux router. For both packet sizes, we observe a cut-off at approximately 0.29 Mpps. Even for 1500 B packets with 0.29 Mpps resulting in 3560 Mbit/s, the throughput is significantly below line rate and demonstrates the limits of the Linux router.

D. Publication of results

As described in Section III, our implementation generates a result folder compliant with the RO-Crate specification. To show the usability of our implementation, we upload this result folder to an open repository. We have decided to use the Zenodo [10] platform. One of the key benefits of Zenodo is the ability to generate unique Digital Object Identifiers (DOIs). Additionally, Zenodo supports versioning and generates separate DOIs for different versions, which is particularly useful when combined with the platform's feature of linking GitHub repositories. The results of this demonstration measurement are available under the following DOI https://doi.org/10.5281/zenodo.10966199.

V. Conclusion

The abstract FAIR principles must be transferred to specific measures to become applicable to scientific research. The particular requirements between different research fields necessitate the adoption of FAIR in a domain-specific form. In this paper, we developed an approach to apply the FAIR principles to testbed-driven experiments for the domain of computer science.

Testbed-driven experiments in computer science are defined by the hardware and software environment in which they were executed. Our paper demonstrates tools and approaches to ensure the conservation of the experimental environments, such as the pos framework to create and execute reproducible experiments, and additional tools to collect the hardware specifications and network topology of the experiment. We further demonstrate how experimental results can be packaged and shared in a structured way using the RO-Crate specification. In detail, we have restructured the result folder of experiments and added new metadata, such as information about the authors and their affiliations. After that, we showed our implementation's functionality in a demonstrative experiment. The demonstration substantiates the satisfaction of the FAIR principles. First, the publication on Zenodo ensures the *findability* and the *accessibility* of results. Second, the used toolchain and the packaged results themselves fulfill the *interoperability*, and *reusability* principles.

The presented approach is an initial step towards FAIRness and we plan to extend our approach in future work. Therefore, we aim to add additional metadata, such as funding or general information about the related research project. Moreover, we plan to link single datasets from different experiments together to further increase the *interoperability*. As the LLDP-based approach works well for our application scenario, wireless nodes, especially mobile ones, require a more sophisticated way for topology detection. Finally, we note that the SLICES project is still in its early stages and will continue developing schemas to describe their experiments [28]. With pos, as a part of SLICES, we plan to accompany this process and adapt and integrate these schemas to reflect the needs of SLICES' users.

ACKNOWLEDGMENT

This work is partially funded by the European Union's Horizon 2020 research and innovation programme (grant agreement no. SLICES-PP 101079774, SLICES-SC 101008468, and GreenDIGIT 101131207). The German Federal Ministry of Education and Research (BMBF) supported our work under the projects 6G-life (16KISK002) and 6G-ANNA (16KISK107) as well as the German Research Foundation (DFG) as part of the HyperNIC (CA595/13-1) project. Additional funding was received by the Bavarian Ministry of Economic Affairs, Regional Development and Energy within the project 6G Future Lab Bavaria.

REFERENCES

[1] ACM. (2020) Artifact Review and Badging Version 1.1. Accessed: 2024-03-24. [Online]. Available: https://www.acm.org/publications/policies/artifact-review-and-badging-current

[2] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, 2016.

[3] geni-lib, "Welcome to geni-lib's documentation!" [Online]. Available: https://geni-lib.readthedocs.io/en/latest/

[4] H. Stubbe, S. Gallenmüller, and G. Carle, "The pos experiment controller: Reproducible & portable network experiments," in *19th Wireless On-Demand Network Systems and Services Conference, WONS 2024, Chamonix, France, January 29-31, 2024*. IEEE, 2024. [Online]. Available: https://doi.org/10.23919/WONS60642.2024.10449532

[5] Chameleon Cloud Developers, "Trovi." [Online]. Available: https://chameleoncloud.org/experiment/share/

[6] S. Soiland-Reyes, P. Sefton, M. Crosas *et al.*, "Packaging research artefacts with ro-crate," *Data Sci.*, vol. 5, no. 2, 2022. [Online]. Available: https://doi.org/10.3233/ds-210053

[7] A. Jacobsen, R. de Miranda Azevedo, N. S. Juty *et al.*, "FAIR principles: Interpretations and implementation considerations," *Data Intell.*, vol. 2, no. 1-2, 2020. [Online]. Available: https://doi.org/10.1162/dint_r_00024

[8] A. Lamprecht, L. J. García, M. Kuzak *et al.*, "Towards FAIR principles for research software," *Data Sci.*, vol. 3, no. 1, 2020. [Online]. Available: https://doi.org/10.3233/ds-190026

[9] The European Open Science Cloud. https://eosc.eu/. Accessed: 2024-04-06.

[10] European Organization For Nuclear Research and OpenAIRE, "Zenodo," 2013. [Online]. Available: https://www.zenodo.org/

[11] D. Duplyakin, R. Ricci, A. Maricq *et al.*, "The design and operation of cloudlab," in *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, 2019. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/duplyakin

[12] K. Keahey, J. Anderson, Z. Zhen *et al.*, "Lessons learned from the chameleon testbed," in *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*. USENIX Association, 2020. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/keahey

[13] D. Balouek, A. Carpen-Amarie, G. Charrier *et al.*, "Adding virtualization capabilities to the grid'5000 testbed," in *Cloud Computing and Services Science - Second International Conference, CLOSER 2012, Porto, Portugal, April 18-21, 2012. Revised Selected Papers*, ser. Communications in Computer and Information Science, vol. 367. Springer, 2012. [Online]. Available: https://doi.org/10.1007/978-3-319-04519-1_1

[14] L. Nussbaum, "Testbeds Support for Reproducible Research," in *Proceedings of the Reproducibility Workshop*, 2017.

[15] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle, "The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments," in *CoNEXT '21: The 17th International Conference on emerging Networking EXperiments and Technologies, Virtual Event, Munich, Germany, December 7 - 10, 2021*. ACM, 2021. [Online]. Available: https://doi.org/10.1145/3485983.3494841

[16] Y. Demchenko, S. Gallenmüller, S. Fdida, P. Andreou, C. Crettaz, and M. Kirkeng, "Experimental research reproducibility and experiment workflow management," in *15th International Conference on COMmunication Systems & NETworkS, COMSNETS 2023, Bangalore, India, January 3-8, 2023*. IEEE, 2023. [Online]. Available: https://doi.org/10.1109/COMSNETS56262.2023.10041378

[17] Dublin Core. https://www.dublincore.org/. Accessed: 2024-04-07.

[18] Schema.org. https://schema.org/. Accessed: 2024-04-07.

[19] JSON-LD 1.1: A json-based serialization for linked data. https://www.w3.org/TR/json-ld/. Accessed: 2024-03-25.

[20] P. De Geest, B. Droesbeke, I. Eguinoa *et al.*, "Researchobject/ro-crate-py: ro-crate-py 0.9.0," Oct. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.10017862

[21] ORCID: Open Researcher and Contributor ID. https://orcid.org/. Accessed: 2024-03-07.

[22] ROR: Research Organization Registry. https://ror.org/. Accessed: 2024-03-07.

[23] L. Vincent, "lshw: HardWare LiSter for Linux," https://github.com/lyonel/lshw, 2024.

[24] "IEEE Standard for Local and metropolitan area networks - Station and Media Access Control Connectivity Discovery," *IEEE Std 802.1AB-2016 (Revision of IEEE Std 802.1AB-2009)*, 2016.

[25] V. Bernat, "lldpd: Implementation of IEEE 802.1ab (LLDP)," https://github.com/lldpd/lldpd, 2024.

[26] GUDE: Expert Power Control 8226-1. https://web.archive.org/web/20230928065702/https://gude-systems.com/en/products/expert-power-control-8226/. Accessed: 2024-04-09.

[27] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*. ACM, 2015. [Online]. Available: https://doi.org/10.1145/2815675.2815692

[28] P. Andreou. (2022) D4.1: Data Management Plan. Accessed: 2024-04-10. [Online]. Available: https://doi.org/10.5281/zenodo.5869390