User Study of a Network Debugger on FABRIC

Alexander Wolosewicz
Illinois Tech
Chicago, USA

Vinod Yegneswaran
SRI
Menlo Park, California

Ashish Gehani
SRI
Menlo Park, California

Nik Sultana
Illinois Tech
Chicago, USA

***Abstract*—Security tool evaluations and experiments are often hampered by ad-hoc setups, missing telemetry, and fragile infrastructure. We report on a controlled experiment using CRINKLE, a novel, interactive network packet provenance debugger that is deployed on the national-scale FABRIC testbed. Participants using CRINKLE consistently solved debugging tasks faster and with less effort: two verified correct solutions were completed in under six minutes with as little as one line of code, outperforming advanced non-CRINKLE users despite having far less FABRIC experience. Our integrated telemetry stream captured 364 records with negligible overhead and zero loss, enabling real-time detection of user and infrastructure issues and proactive remediation. Over 38 slices and 24 different FABRIC sites, we identified and resolved common platform problems (permissions gaps, SSH key failures, transient site outages). These findings demonstrate that pairing fine-grained telemetry with realistic, federated testbeds yields reproducible, performance-driven security tool evaluations and that CRINKLE can serve as both a high-performance debugger and a model for instrumented cybersecurity experimentation.**

***Index Terms*—Network Experiments; Network Debugging; User Study**

1. Introduction

Testbed environments are used to develop, debug, and evaluate research artifacts. Debugging in complex multi-tenant, testbed environments can be a vexing challenge as failures may arise from diverse causes including subtle configuration errors, transient infrastructure issues, interference, or unanticipated interactions between physical, emulated, and virtualized components. These problems could lead to pathologies such as unstable connectivity, variability in latencies, or unexplained packet loss–issues that are often time consuming and laborious to diagnose and reproduce.

Challenges like these are amplified in cybersecurity testbeds, where a simple misconfiguration or overlooked failure mode can pave the way for unintended vulnerabilities, undermine monitoring coverage, disrupt essential services, or otherwise compromise the validity of experiements. We posit that ensuring system reliability and diagnostic clarity is vital for maintaining effectiveness and integrity of cybersecurity experiments.

While traffic monitoring tools, such as `tcpdump` and `Wireshark`, provide raw network packet visibility, they place the burden of causal reasoning on the operator. Higher-level instrumentation systems, including provenance-based approaches, can capture the lineage of network events and highlight causal relationships between failures and underlying configurations. Prior research on distributed provenance [1]–[3] and distributed tracing [4]–[6] has shown that causality-aware analysis can accelerate root-cause identification. However, these systems are often evaluated in small-scale, controlled environments and lack integration with modern, federated testbeds or realistic multi-tenant infrastructure.

Large-scale experimental platforms such as Emulab [7], FABRIC [8] and SPHERE [9] have emerged to support reproducible networking and security research. FABRIC provides international, reconfigurable resources that can be programmatically instrumented, making it a promising venue for evaluating interactive debugging tools under realistic conditions.

Despite the importance of testbeds for networking and cybersecurity research, few studies have explored how such testbeds can be leveraged to run user studies for debugging and diagnostic tooling, especially with integrated telemetry that tracks user interactions, infrastructure state, and tool performance in real time.

This paper presents an approach to address that gap by evaluating CRINKLE, an interactive packet provenance debugger, in a controlled study on the FABRIC testbed. By combining provenance-driven debugging, comprehensive telemetry, and a federated experimental platform, we demonstrate a methodology for conducting reproducible, performance-focused evaluations of debugging tools and instrumentation approaches one that can generalize beyond CRINKLE to other operational and research contexts.

Contributions. The contributions of this experience report are threefold:

- We present the first large-scale, user-centered evaluation of CRINKLE, measuring task performance against a non-CRINKLE baseline.
- We design and deploy a fine-grained telemetry system that captures both user activity and infrastructure state with negligible overhead.
- We analyze FABRIC's suitability as a platform for con-

trolled experiments with debugging and instrumentation tools, identifying and categorizing infrastructure-related challenges encountered during the study.

1.1. Artifact Sharing

The open-source FABlib API [10] manages resources and experiments on the FABRIC testbed (Section 2). This API is used by FABRIC users for defining, reserving, and managing their experiments. FABlib is commonly used through Jupyter notebooks [11], which are also used on other testbeds, such as Chameleon. Notebooks provide both flexibility for designing experiments and easily reproducible artifacts [12], [13]. For our work, participants were given notebooks to complete (Section 5). A copy of the materials participants were given is distributed as a FABRIC artifact[1].

1.2. Ethical Concerns

This research involved human participation by FABRIC users. Before carrying out the user experiment, we submitted an IRB (Institutional Review Board) request to our institution and received an exemption. Participants used CRINKLE's evaluation platform on FABRIC to evaluate CRINKLE's network debugger and provide feedback. Participants later completed a survey to give feedback about the evaluation platform.

2. Background: FABRIC and FABlib

This section describes the FABRIC testbed, which provides the development and evaluation environment for this research. As a *federated* testbed, FABRIC is formed of a set of 33 *sites* that are located across a wide geographic area. There are FABRIC sites in universities and Internet exchange points (IXPs) across the USA, Asia, and Europe.

Each FABRIC site hosts a rack of equipment that provides resources to the testbed. These resources include Tofino switches and several worker machines that host various types of GPUs, storage devices, and network cards (including Mellanox ConnectX NICs and Alveo FPGA NICs). For example, the site at the University of Hawai'i has 5 worker machines that (in aggregate) host 640 CPU cores, 108TB storage, 2.4TB RAM, 8 A30 GPUs, 16 P4510 NVMe devices, 6 single-user ConnectX NICs, a ConnectX NIC that can be shared among 635 users,[2] and 1 Alveo FPGA card.[3] Reservable resources in FABRIC sites are called *slivers*. Slivers can include nodes (Virtual Machines that are allocated storage, CPU cores, and RAM on worker machines), programmable switches, NICs and GPUs. Slivers are combined to create a *slice*—an example slice is shown in Fig. 1. A slice consists of the set of resources that researchers use to carry out a testbed experiment. This terminology is common across large testbeds, and it is not used exclusively by FABRIC. Slices can include any mix of slivers, and that mix can be changed at any time, provided that the resources are available.

FABRIC utilizes two sets of abstractions to define the network links within a slice [14]. These links interconnect NICs and switches in a slice. The first set consists of Layer 2 (L2) abstractions such as L2Bridge which links resources within a single FABRIC site, L2STS (L2-Site-to-Site) which links resources across sites, and L2P2P (L2-Point-to-Point) which links exactly two dedicated NICs between two sites. To the experimenter, these connections appear as if the resources are directly linked. The second set consists of Layer 3 (L3) abstractions, which trade some control for convenience. They utilize a service, called FabNet to connect nodes through the sites' L3-routed networks. These connections have a further External type that allows for the experimental data plane to connect to the wider Internet. L2 and L3 abstractions can be used with Facility Ports to connect FABRIC resources to external resources, such as Chameleon [15], which allows experimenters to use FABRIC's network flexibility with the dedicated resources of other testbeds.

2.1. Debugging needs on testbeds

An important network debugging challenge in federated network testbeds involves separating bugs in the researcher's network experiment from problems in the testbed's underlay network. Network problems that arise on a federated testbed network are not unlike problems on other networks. FABRIC racks are placed at professionally-managed facilities but they are not immune from unplanned downtime—including outages caused by fiber cuts [16], or pipe bursts leading to an electricity shutdown [17]. Moreover, FABRIC's network requires ongoing maintenance to address issues such as transceiver reseating, optics cleaning, and firmware and configuration updates.

Because of their role, federated network testbeds also face additional debugging challenges: **(1)** Experimental prototype network architectures and processors take months to develop, during which time they will require network debugging. **(2)** Shared testbeds can involve simultaneous experiments by different (tenant) researchers whose (virtualized) slice resources are not perfectly isolated, which can affect experiment results and their reproducibility. **(3)** Host and network virtualization is used to provide abstractions to different researchers, but abstractions occasionally leak. For example, virtual NICs do not receive all frames, even when set to run in promiscuous mode [18]—violating the "rule of least surprise" [19] and giving researchers a wide set of possible causes for this behavior. For another example, experiments might occasionally fail because they rely on sites that intermittently run out of resources because of heavy use—for example, IPv4 addresses might be depleted [20].

1. https://artifacts.fabric-testbed.net/artifacts/e190c5f4-d753-4521-a682-168d12460924

2. FABRIC uses SR-IOV and PCIe passthrough to provide access to host resources to its performance-sensitive users.

3. The resources of the FABRIC site at the University of Hawai'i are described at: https://portal.fabric-testbed.net/sites/HAWI

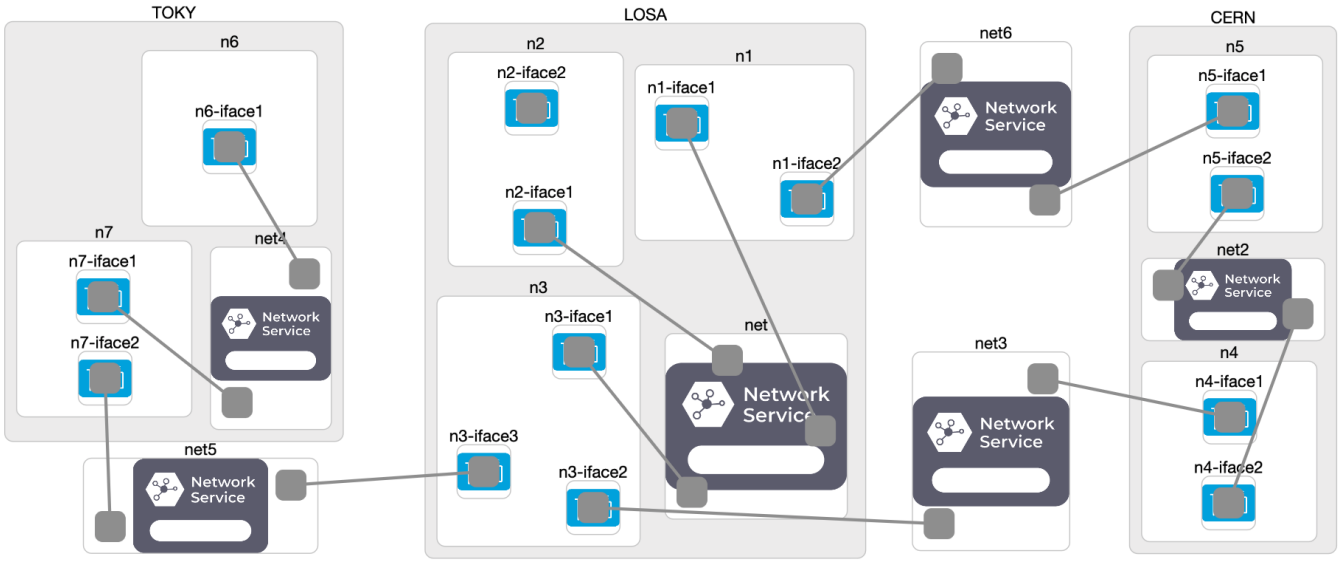


Figure 1: Example of a FABRIC *slice* (defined in Section 2). This visualization is produced by FABRIC and shows a multi-path network that consists of seven nodes (n1-n7) that reside in three FABRIC sites (at Tokyo, Los Angeles, and at CERN in Geneva). This network sets up a user-defined topology of intra-site and inter-site links (net1-net6).

2.1.1. A typical FABRIC network experiment. A FABRIC experiment typically consists of generating, processing, and analyzing network traffic. As part of an experiment, researchers might install and run prototype software or hardware packet processors or network topologies. A slice can represent any type of network—including scaled-down HPC clusters, telecom backbones, and datacenter networks. Figure 1 shows a typical network (slice) on FABRIC. Slices can differ by the number of nodes and other types of slivers (such as smart NICs) that they include. Slices also differ by the software that is installed on nodes and the configuration of nodes, NICs, and other slivers.

2.1.2. Debugging needs for a network experiment on FABRIC. Consider the slice shown in Fig. 1. Network-related problems that might arise include:

- Congestion and faults in FABRIC's underlay, including fiber cuts, as described above. In Figure 1, this could result in traffic loss or delays at any node in the topology.
- Forwarding and routing: **(1)** for traffic flowing between n6 (in Tokyo) and n5 (at CERN), each step of the journey depends on correct configuration. For instance, n7 must be configured to forward packets between two interfaces. **(2)** n2 might be misconfigured to egress traffic on n2-iface2 instead of n2-iface1.
- Delays might be caused by congestion on intermediate nodes and their hosts. Fig. 1 shows a multi-path topology in which traffic from n6 to n5 can cross either net3 or net6 for load balancing, but if misconfigured this can lead to network bugs.
- Additional sources of network problems can arise from the use of smart NICs and FPGA cards because of their higher complexity to program and use.
- As a network topology grows, so do the sources of potential network problems—for instance, there are more elements that could fail or be misconfigured.
- Different runs of this experiment might behave differently because of background traffic. By default, testbeds do not provide per-experiment network telemetry that can be used to measure background traffic.

3. Network Debugging on FABRIC: State of the Art

As part of the research on CRINKLE, we surveyed FABRIC users to establish how they carry out network debugging on FABRIC, and how it compares to carrying out networking debugging on a non-testbed network. 22 people participated in this survey—which is part of a larger study described in Section 6.1. We learned that:

- All respondents use low-level tools and techniques (such as tcpdump and traceroute) for network debugging. Respondents tend to use the *same* network debugging tools and techniques on FABRIC as they do *outside* of FABRIC. The main difference between FABRIC and non-testbed network debugging consists of the use of Jupyter notebooks to run commands on FABRIC nodes and the use of vendor-specific tooling outside of FABRIC.
- 50% of surveyed researchers ran Jupyter notebooks authored by other FABRIC users, underscoring the reuse of research artifacts within the FABRIC community.
- 50% of surveyed researchers experienced infrastructure problems when running our notebook—including authentication, authorization, and availability problems related to their configuration. The experiment took place during a typical operation window on the testbed (it was not

undergoing maintenance). This characterizes the feedback that currently needs to be relayed between users, but that could be automatically sent to us as telemetry.

4. CRINKLE

This section introduces the CRINKLE system which was designed to meet the following **goal**: provide a debugging system that enables users, without requiring operator support, to debug issues within their slices of testbed experiments. For our implementation, this means that CRINKLE uses resources, permissions, and APIs that FABRIC already supports.

4.1. Overview

Fig. 2 outlines the workflow that is implemented in CRINKLE to enable researchers on FABRIC to debug their networks.FABRIC researchers ❶ usually use FABRIC through Jupyter notebooks that are hosted at a cloud server. To use CRINKLE, researchers use FABRIC as normal, since CRINKLE extends FABRIC's API with debugging features. The Jupyter notebook server controls FABRIC resources ❷. When using CRINKLE, FABRIC users augment their network with new monitoring capabilities to debug their network. They choose where to install monitors by bisecting links ❸. CRINKLE uses resources on FABRIC for monitoring the network, capturing data, and ❹ storing it. Debugging output and telemetry are streamed back ❺ to the Jupyter notebook. The debugging output is used by FABRIC users to fix network problems, and the telemetry is sent to a collector ❻ for remote analysis.

CRINKLE's telemetry stream consists of tuples that are sent to a collector. The tuples are emitted at different stages of an experiment. The tuple values are used to identify the experiment's stage, user, configuration, FABRIC site and other details that can help diagnose problems remotely. For our experiment, we monitored an endpoint which received HTTP POST requests containing these tuples, and reached out to participants via email when the telemetry indicated they might need assistance. Some circumstances prompting us to reach out included multiple slice setup failures or an excessive amount of time spent on one step.

4.2. Topology extension paradigm

As sketched in Fig. 2, CRINKLE transforms links to add *monitors*. This takes advantage of the flexibility of federated testbeds to add a dedicated monitoring node to the link. To the experiment, the topology is unchanged, but these monitored links provide the ability to observe and influence the network's data plane. CRINKLE requires no support from the user, and it is entirely hardware-agnostic as it can treat every node in the network as a black box.

All traffic that crosses a monitored link will traverse that link's monitor node. On the monitor, this traffic is processed by a specialized DPDK application (Data Plane Development Kit [21]) through which CRINKLE can modify and record traffic. Monitors are connected using an out-of-band L3 network with an *analyzer*, a special higher-resourced node which maintains the database of packet provenance, which is described later in this section.

5. FABRIC User Study

This section outlines the user study carried out using CRINKLE.

Methodology. An IRB (Institutional Review Board) exemption was obtained for an experiment design involving human participants who would carry out network debugging exercises. This experiment was carried out in a novel manner: it recruited participants who use FABRIC, and who could solve the exercises by carrying out experiments on FABRIC. Participants would be given a Jupyter notebook by the experiment organizers. This notebook would create a network topology and simulate two network problems for participants to debug.

Participants were recruited by word-of-mouth and had a range of different experience levels—we deliberately avoided only recruiting students since they tend to have less experience. Participants were given a few days to solve the exercises and were allowed to install arbitrary software on their FABRIC nodes. After completing the exercises, participants were asked to complete a short survey.

Exercises. Fig. 3 shows the network topology that was used by both exercises that participants were asked to solve. The topology includes two hosts connected by multiple paths through nine routers. Each router ran a P4 program that was configured differently on each router. The P4 program was a simple IPv4 forwarding program that would take an input packet, use the destination IP to decide the outgoing ports, and then rewrite the MAC addresses for the outgoing link. The program additionally used a counter to decide which output port to forward on, which reset after 3 (so valid values of 0, 1, and 2), to allow for using multiple paths without using ECMP (since the flows were unchanging). The routers were not programmed to respond to ICMP or any other management protocol.

For Exercise 1, router r7 had a single-character misconfiguration in its forwarding rules that would send packets arriving from r8 back towards r8 instead of towards r6. For Exercise 2, the routers were configured to add GTP-U tunneling headers consisting of an outer IPv4 header, a UDP header, and an 8-byte GTP-U header [22], with the outer IPv4 headers using new IP addresses. Router r9 was given incorrect forwarding rules which caused it to drop tunnelled packets. For each exercise, participants were shown the results of 10 pings from h1 to h2, and were asked to identify which router was misconfigured, with the correct answers of r7 for Exercise 1 and r9 for Exercise 2.

Participants were informed immediately if their answer was correct or not, and could make multiple attempts.

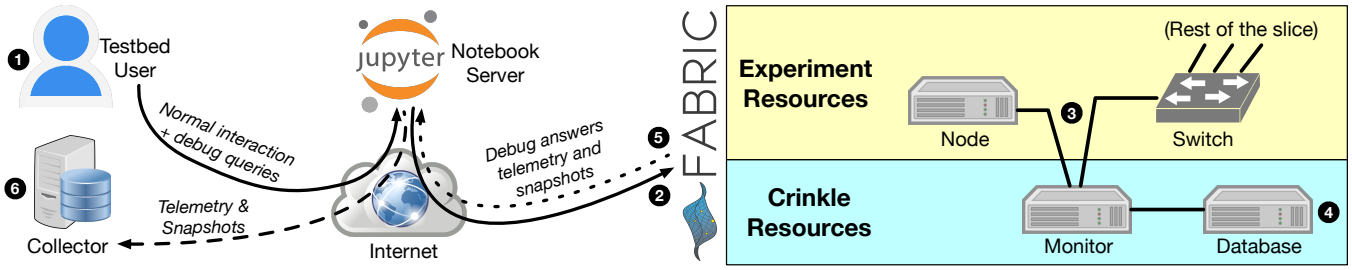


Figure 2: Overview of CRINKLE's workflow, detailed in Section 4.

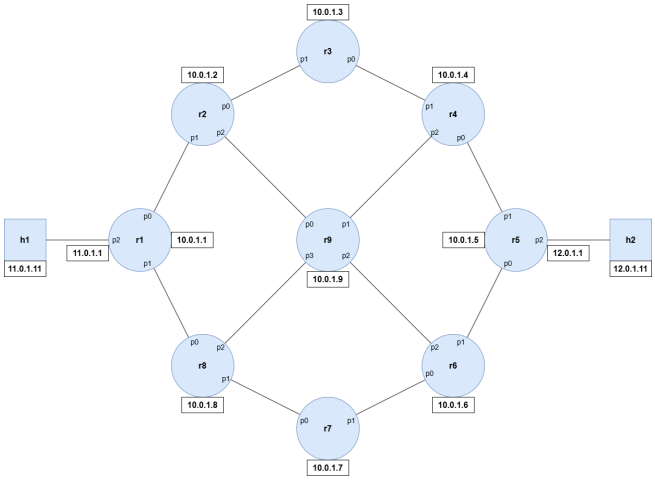
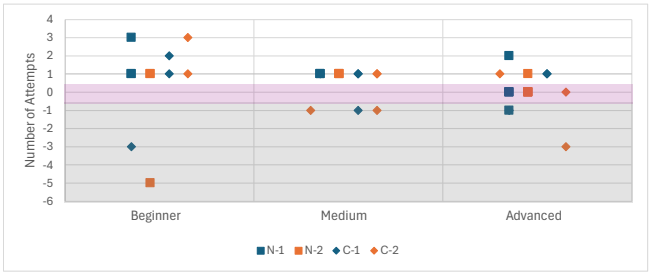
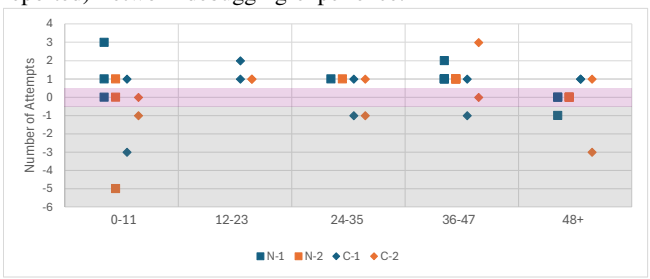


Figure 3: Topology used in the user study. This topology is described in Section 5.



(a) The number of attempts by participants grouped by (self-reported) network debugging experience.



(b) The number of attempts made by participants, grouped by months of experience with FABRIC.

Figure 4: The number of attempts made by participants. A negative value indicates the number of attempts made without eventually reaching the correct answer. "N" is the group tasked with using any technique, "C" is the group tasked with using CRINKLE.

6. Evaluation

CRINKLE is evaluated in a first-of-its-kind user experiment (described in Section 6.1) to assess the **(1)** effectiveness of the debugger and its telemetry, and **(2)** usability by FABRIC users.

6.1. User study

This section describes the first user study that was carried out using FABRIC. In this study, FABRIC provided network and computational resources, and served as the platform on which physically-distributed study participants solved network debugging exercises.

6.1.1. Experiment setup. The user study was planned over 6 months. This section describes how the user experiment was run, and the sections that follow report the experiment's results.

Population. Participants were recruited as described in Section 5. The set of participants was split evenly into two: a set of participants who were given CRINKLE for debugging, and the other set were asked to use any network debugging approach they preferred. For fairness, the two sets were balanced for the number of participants and their relative experience. For both sets of participants, the notebooks streamed telemetry to the collector (Fig. 2). The final answers from both sets of participants were also uploaded to the collector for analysis.

Among participants, months of FABRIC experience and network debugging experience were weakly correlated. While participants who had Medium experience with network debugging also had middling FABRIC experience, Beginner and Advanced were fairly represented in both the high and low end of the range. The non-CRINKLE group had an average of 26.7 months of FABRIC experience, while the CRINKLE group had an average of 26.8 months of FABRIC experience.

Responses. 33 participants signed up to complete the exercises and were split into two sets of participants—of 17 and

TABLE 1: Attributes of the experiment participants in the CRINKLE group and the control group.

Attribute	Non-CRINKLE	CRINKLE
Network Debugging Experience		
Beginner	3	4
Medium	3	4
Advanced	5	3
Months of FABRIC Experience		
0-11	3	3
12-23	0	2
24-35	1	2
36-48	4	2
48+	3	2
Has P4 Experience		
Yes	3	7
No	6	3
No Answer	2	1

16 people: the first set would use CRINKLE for network debugging, and the second set is the control group who can use whichever technique they prefer. Ultimately, 11 people in each set completed the experiment, and their composition is shown in Table 1.

Problem set. The problems were designed to range across some of the needs described in Section 2.1.2. Participants were given two network debugging exercises that used the same 9-router, multi-path network topology—the topology and further details are provided in Section 5. Both exercises involve identifying a misconfigured router. **Exercise 1** involves an incorrect forwarding rule. **Exercise 2** involves a faulty program being applied by a router. In both exercises, the misconfiguration results in packet loss, and the participant must correctly identify the misconfigured router.

6.1.2. Effectiveness of the telemetry. The telemetry system was able to deliver all telemetry records to the collector with negligible overhead for users. During the experiment, the collector received 364 telemetry records. These included 85 records of attempts to carry out the exercises, and also revealed the problems that were encountered by participants—these are described in Section 6.1.3. Three participants encountered issues that were not caught by the telemetry, but the telemetry system can be patched to report these in the future. The uploads from participants were compared against the streamed telemetry, and this confirmed that no telemetry messages were lost.

6.1.3. Analysis of using FABRIC for a user experiment. During this study, 38 slices were created on FABRIC, and participants used 24 sites in aggregate. The telemetry revealed several instances of infrastructure-related problems:

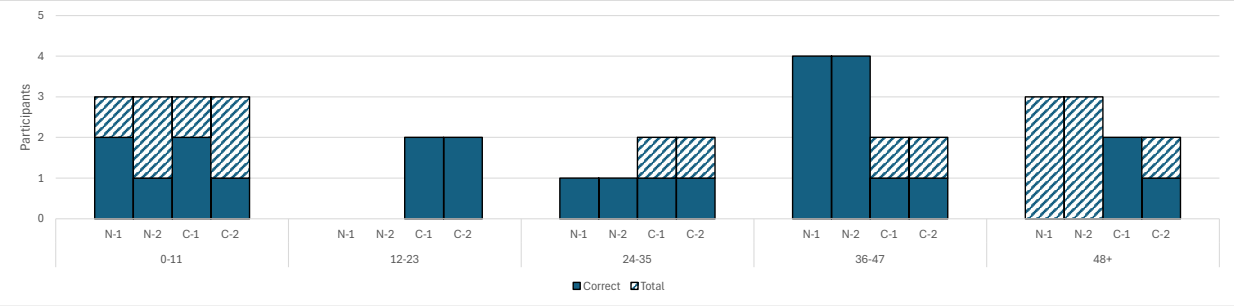
- 1 participant was missing permissions in their project and could not run the experiment notebook, and 2 participants were missing a FABRIC project. These 3 participants were added to the FABRIC project for this study.
- 1 participant was missing a configuration for their Jupyter environment.
- 4 participants had issues relating to missing or expired SSH keys for their FABRIC resources.
- 5 participants had a slice successfully reserve and come up, fail the baseline, then succeed on the next site. This was distributed among both groups—with 3 (N) and 2 (C) participants in each.
- 1 participant was unable to reserve resources due to a depletion at their FABRIC site. The participant's slice succeeded when attempted at a different site. There was also a FABRIC site with persistent problems throughout the study, but these were unrelated to CRINKLE.

Having the telemetry stream enabled us to reach out to struggling participants proactively. In each case, participants were helped to overcome these issues since they did not relate to the debugger's evaluation. Some participants faced multiple issues, and the total number of participants who faced infrastructure-related problems were 5—which indicates that FABRIC is a reasonable platform on which to carry out such user studies.

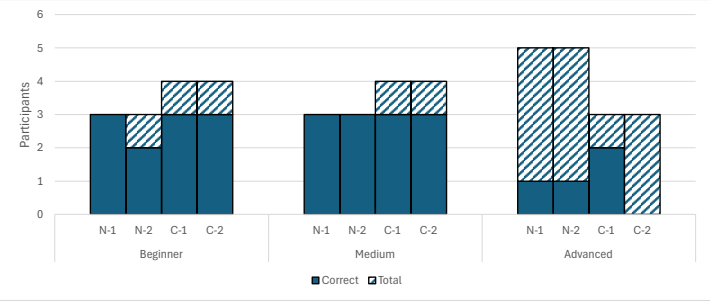
6.1.4. Effectiveness of the debugger. The results described below support the following conclusions:

- For users with some experience with FABRIC and network debugging, by using CRINKLE they could *debug on average up to 37.5% faster*, with one participant correctly solving an exercise with a single line of code. For other users, CRINKLE is at least as effective (in terms of number of correct answers and time taken) as existing debugging techniques.
- For highly-advanced users, the lack of familiar tools (like traceroute) appears to be a barrier. We believe that advanced users have established methods of debugging network issues, and might be less receptive to new methods.
- The debugger utilizes a linear number of monitoring resources on FABRIC, resulting in larger slices when compared to the original FABRIC experiment, but this does not contribute to instability.
- Some CRINKLE participants, mostly those with Beginner network debugging experience, struggled to understand the provenance graph output from CRINKLE. Improved teaching methods such as hands-on tutorials could help in this case, as for the experiment CRINKLE participants were given only a 15-minute demonstration video and API documentation.

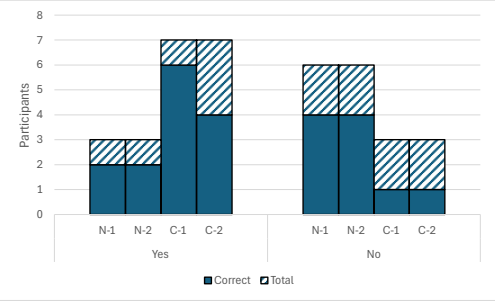
Of the 11 participants in each group, 6 correctly solved both exercises while an additional 1 non-CRINKLE and 2 CRINKLE participants solved just the first exercise. The non-CRINKLE group that correctly solved both exercises consisted of 2 Beginner, 3 Medium, and 1 Advanced user while the CRINKLE group was 3 Beginner and 3 Medium users (Figure 5b). By network debugging experience, most of the participants with incorrect or incomplete submissions belonged to the Advanced group (Figure 5b), and by FABRIC experience they were generally at either the low or high ends (Figure 5a). There were 6 incomplete runs, which were



(a) Users who provided correct answers to an exercise, grouped by months of FABRIC experience.



(b) Users who provided a correct answer, grouped by network debugging experience.



(c) Users who provided a correct answer, grouped by having P4 experience.

Figure 5: "N" is the group tasked with using any technique, "C" is the group tasked with using CRINKLE.

solely Advanced participants, and 5 responded with reasons for their incompletion with most indicating a combination of reasons. Of these 5: 3 indicated they ran out of time; 3 that the instructions were unclear; 4 that the environment was too unfamiliar (as in, could not install wanted tools or were confused about CRINKLE); 1 that FABRIC technical issues were a barrier (in this case, causing them to run out of time). Most participants that had knowledge of P4 were able to solve one or both exercises (Figure 5c). Members of the Beginner group or those with less FABRIC experience generally took more attempts to reach the correct answer than their more experienced counterparts (Figure 4).

For the part of each group that correctly solved both exercises, Beginner members of the CRINKLE group were generally equal to their non-CRINKLE peers for Exercise 1, but they did not experience a speed-up the CRINKLE members did for Exercise 2.

For Medium-experience members, CRINKLE participants were on average 37.5% faster than their non-CRINKLE counterparts for both exercises. Notably, while one non-CRINKLE participant managed to guess and submit the correct answer without verification in just over a minute—a valid technique used by other participants, though with verification—two of the CRINKLE participants submitted correct answers with verification in under 6 minutes. The three CRINKLE participants respectively executed 1, 6, and 7 lines of code, indicating that when familiar with the tool they can efficiently use it.

One Advanced CRINKLE group member was over twice as fast as the successful non-CRINKLE member on Exercise 1 while having less FABRIC experience (0-11 months vs 36-47 months).

7. Findings, Recommendations, and Implications for Cybersecurity Experimentation

This section discusses the findings of this research and offers recommendations to improve the breadth and accuracy of experiments that are carried out on state-of-the-art federated testbeds.

7.1. Findings: Using FABRIC for User Experiments

As analyzed in Section 6.1.3, we found that FABRIC is a viable platform for carrying out a distributed, asynchronous user study. The downside of carrying out a study in this format was that the experiment conditions were less controlled than if participants were physically constrained to a location—one participant said they had to "complete the work while making a lot of context switches among other things." The upside is that we were able to recruit a wider set of participants—particularly when compared to a classroom experiment, which tends to produce more homogeneous and less-experienced participants.

Using FABRIC greatly simplified the process of providing participants with equal access to a well-resourced experiment platform, but Section 6.1.3 identifies two types challenges that participants encountered—and that are therefore likely to be encountered by general FABRIC users.

First, there were occasional inconsistencies and faults in the distributed system. Second, there was a long start-up time for the experiment—which ate into the time that participants could spend on the experiment. To mitigate this, we added a CRINKLE-specific VM image on FABRIC to speed-up the start of nodes in its topology. Even then, we observed that the average start-up time for the CRINKLE cohort was 46.68 minutes (σ 5.58), whereas for the non-CRINKLE cohort the average was 27.23 minutes (σ 22.38). The next section makes recommendations to overcome both these difficulties.

7.2. Feature Recommendations

Based on the observations made during the CRINKLE user study, we offer these recommendations for testbed features: **(1)** Since slices can take long to start, we suggest that testbeds support a callback action that is invoked once a slice is started, and that the slice start-up can complete in the background. Such an action could consist of sending an email or running a Python script. Moreover, when starting a slice, users should be able to run fine-grained status query. **(2)** For closer collaboration between testbed users—such as the collaboration through which we got feedback from our study participants—it would be useful to be able to start slices and then hand them over to other testbed users. This would have enabled us to reduce the burden on study participants who had to navigate a long setup process. **(3)** To help distinguish experiment-level problems from FABRIC-level problems, we suggest the adoption of a "Power-On Self-Test" best practice for testbed experiments. This was carried out as part of our study, and provided an early indication of underlying problems that would have undermined the rest of the experiment being carried out by a remote participant.

7.3. Implications: Testbed Experimentation

CRINKLE provides an important support for running experiments on testbeds through: (1) providing real-time monitoring of experiments, (2) providing precisely-measurable artifact behavior, (3) obtaining telemetry from other testbed users who are running an instance of the experiment, (4) obtaining feedback from other testbed users that is inlined with telemetry. This combination of features is essential for research, and user input helps scale evaluation results. This is also particularly when evaluating cybersecurity research through user experiments. The research described in this paper underscores the viability of carrying out asynchronous and distributed user experiments on testbeds, which widens the pool of participants considerably when carrying out user experiments. Overall, the study demonstrates that with proper instrumentation using tools like CRINKLE and attention to user experience, platforms like FABRIC can serve as powerful, scalable foundations for interactive cybersecurity research and evaluation.

8. Related Work

To our knowledge, this paper describes the first user study of a network debugger. This study targeted CRINKLE, the first telemetry system for a shared testbed and the first network debugger for a testbed. Past work on testbed debugging focused on wireless problems [23], [24].

When compared to related work in non-testbed networks, CRINKLE provides novel primitives for both network debugging and telemetry without requiring programmable network elements. For telemetry, CRINKLE streams experimenter-selected information at experimenter-selected points to a collector, and cross-user, cross-experiment, cross-site analysis can be carried out on that telemetry. Compared to telemetry that is used for performance diagnosis for datacenters [25], the Internet [5], [26], and in programmable networks [27], this telemetry information is integrated with rich debugging state for remote analysis of network and experiment problems.

Recent work on network debugging leveraged Software-Defined Networking to increase the debuggability and verifiability of networks or uses abstractions of network configuration. This is one of the key differences of CRINKLE over related work: CRINKLE works on actual traffic and non-programmable, third-party hardware.

CRINKLE is primarily designed for a high-performance, federated testbed environment, and leverages the testbed's ability to reconfigure itself. This environment produces both technical *opportunities* and *challenges* when compared to the ISP and datacenter networks that related work is designed for. Opportunities include more transparency and agency for network users (who are given more visibility and control when compared to users of other networks), and challenges include: inconsistent network load from other users, workload diversity and the presence of prototype protocols, and—when compared to debugging techniques devised for network operators—a lack of total control of the network. CRINKLE is a runtime tool that operates on network traffic—not on abstractions of the network's configuration—and it is hardware and protocol agnostic.

9. Conclusion

Despite the importance of testbeds for networking and cybersecurity research, very little research has been directed at how to use testbeds to run user studies involving research artifacts running on those testbeds. This paper presents such a user study for CRINKLE—a novel, interactive packet provenance debugging and diagnostic tool, intended to support the research of testbed users. CRINKLE telemetry tracks user interactions, infrastructure state, and tool performance in real time. To our knowledge, this paper was the first to present a controlled user study on the FABRIC testbed. Future work includes adapting the techniques used in this user study to enable external participants to evaluate and provide feedback on cybersecurity research prototypes that run on FABRIC and other testbeds, to support testbed researchers in carrying out user studies for their systems.

Acknowledgement

We thank the anonymous reviewers for their feedback. We thank the participants for their time and engagement with our user experiment. We thank Nishanth Shyamkumar at IIT for technical assistance, and Komal Thareja and Mert Cevik from RENCI for help with FABRIC. This work was supported by the National Science Foundation (NSF) under award 2346499. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of funders.

References

[1] P. Groth and L. Moreau, "Representing distributed systems using the open provenance model," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 757–765, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X10001949

[2] A. Gehani and D. Tariq, "SPADE: Support for Provenance Auditing in Distributed Environments," *13th ACM/IFIP/USENIX International Middleware Conference*, 2012. [Online]. Available: http://www.csl.sri.com/users/gehani/papers/MW-2012.SPADE.pdf

[3] H. Irshad, G. Ciocarlie, A. Gehani, V. Yegneswaran, K. H. Lee, J. Patel, S. Jha, Y. Kwon, D. Xu, and X. Zhang, "TRACE: Enterprise-Wide Provenance Tracking For Real-Time APT Detection," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 16, 2021. [Online]. Available: http://www.csl.sri.com/users/gehani/papers/TIFS-2021.TRACE.pdf

[4] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, "Magpie: Online modelling and performance-aware systems," in *9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, 2003.

[5] R. Fonseca, G. Porter, R. H. Katz, and S. Shenker, "X-Trace: A pervasive network tracing framework," in *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*. Cambridge, MA: USENIX Association, Apr. 2007. [Online]. Available: https://www.usenix.org/conference/nsdi-07/x-trace-pervasive-network-tracing-framework

[6] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper, a large-scale distributed systems tracing infrastructure," 2010.

[7] F. Hermenier and R. Ricci, "How to Build a Better Testbed: Lessons from a Decade of Network Experiments on Emulab," in *Testbeds and Research Infrastructure. Development of Networks and Communities*, T. Korakis, M. Zink, and M. Ott, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 287–304.

[8] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, "FABRIC: A national-scale programmable experimental network infrastructure," *IEEE Internet Computing*, vol. 23, no. 6, pp. 38–47, 2019.

[9] J. Mirkovic, D. Balenson, and B. Kocoloski, "Enabling Reproducibility through the SPHERE Research Infrastructure," *USENIX ;login*, December 2024.

[10] "Experiment Management APIs (FABLib)," https://learn.fabric-testbed.net/article-categories/fablib-api/.

[11] "Project Jupyter." [Online]. Available: https://jupyter.org/

[12] "FABRIC Artifact Manager," https://artifacts.fabric-testbed.net/artifacts/.

[13] M. Powers, "Sharing Experiments with Trovi," https://www.chameleoncloud.org/blog/2022/06/01/sharing-experiments-with-trovi/, May 2022.

[14] "Network Services in FABRIC." [Online]. Available: https://learn.fabric-testbed.net/knowledge-base/network-services-in-fabric/

[15] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs, "Lessons learned from the chameleon testbed," in *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020.

[16] "FABRIC Forums: FABRIC TACC and FABRIC INDI dataplane links are down," Nov. 2024. [Online]. Available: https://learn.fabric-testbed.net/forums/topic/fabric-tacc-and-fabric-indi-dataplane-links-are-down/#post-7840

[17] "FABRIC Forums: STAR site power loss, connectivity losses," Sep. 2023. [Online]. Available: https://learn.fabric-testbed.net/forums/topic/star-site-power-loss-connectivity-losses/

[18] "L2Bridge without MAC learning?" Jan. 2023. [Online]. Available: https://learn.fabric-testbed.net/forums/topic/l2bridge-without-mac-learning/

[19] E. S. Raymond, "Applying the Rule of Least Surprise," Jun. 2008. [Online]. Available: http://www.catb.org/~esr/writings/taoup/html/ch11s01.html

[20] "FABRIC Forums: failed lease update all units failed in priming," Feb. 2024. [Online]. Available: https://learn.fabric-testbed.net/forums/topic/failed-lease-update-all-units-failed-in-priming/

[21] "Data Plane Development Kit." [Online]. Available: https://www.dpdk.org/

[22] Y. Zhang, T. Wan, Y. Yang, H. Duan, Y. Wang, J. Chen, Z. Wei, and X. Li, "Invade the Walled Garden: Evaluating GTP Security in Cellular Networks," in *2025 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 1159–1177. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00028

[23] G. Mujica, J. Portilla, and T. Riesgo, "Testbed architecture and framework for debugging Wireless Sensor Networks," in *2015 Conference on Design of Circuits and Integrated Systems (DCIS)*, 2015, pp. 1–6.

[24] K. Yi, R. Feng, N. Yu, and P. Chen, "PARED: A testbed with parallel reprogramming and multi-channel debugging for WSNs," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, 2013, pp. 4630–4635.

[25] V. Harsh, W. Zhou, S. Ashok, R. N. Mysore, B. Godfrey, and S. Banerjee, "Murphy: Performance Diagnosis of Distributed Cloud Applications," in *Proceedings of the ACM SIGCOMM 2023 Conference*, ser. ACM SIGCOMM '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 438–451. [Online]. Available: https://doi.org/10.1145/3603269.3604877

[26] W. Sussman, E. Marx, V. Arun, A. Narayan, M. Alizadeh, H. Balakrishnan, A. Panda, and S. Shenker, "The case for an Internet primitive for Fault Localization," in *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, ser. HotNets '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 160–166. [Online]. Available: https://doi.org/10.1145/3563766.3564105

[27] B. Wang, H. Chen, P. Chen, Z. He, and G. Yu, "MARS: Fault Localization in Programmable Networking Systems with Low-cost In-Band Network Telemetry," in *Proceedings of the 52nd International Conference on Parallel Processing*, ser. ICPP '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 347–357. [Online]. Available: https://doi.org/10.1145/3605573.3605622