Debugging Heterogeneous Hardware Research Experiments on a Large-Scale Network Testbed

Alexander Wolosewicz*, Ashish Gehani†, Vinod Yegneswaran†, Komal Thareja#, Mert Cevik#, Nik Sultana*
*Illinois Institute of Technology, †SRI, #RENCI

Abstract

FABRIC is a state-of-the-art, international research testbed which last year hosted 25,000 experiments. It provides researchers with programmable and reconfigurable hardware such as Xilinx U280 FPGA NICs and Tofino switches. Researchers can control the configuration of testbed resources, but this control does not spare researchers from laborious and long debugging workflows.

This work-in-progress paper describes our development of Crinkle, an API that extends FABRIC's testbed management to enable simpler and more intuitive debugging of testbed experiments involving heterogeneous and programmable hardware. Our goal is to provide a system that provides a network environment (especially one that uses reconfigurable hardware) with the comfort and power expected from software debuggers. We outline the current design and future plans for how these resources can be made more usable by testbed users.

1 Introduction and Background

Modern testbeds such as FABRIC [3] provide a variety of hardware for large-scale research experiments. However, when running an experiment on a testbed, there is often a lengthy process to debug the experiment before one can realize actual research results. This process is complicated by the lack of a network-wide debugger, especially one general enough for research work.

From our own experience of using this equipment on FABRIC [4, 10], from on-boarding others to use this equipment [9, 12], and from conversations within the community, we see that the most commonly used technique for network debugging involves packet capture. Capturing traffic at points in the network allows researchers to observe the effects of devices between them. The captured traffic also reveals how a packet was transformed or why it appeared (or did not appear) at that point. For small-scale tests, this is often sufficient, and can be paired with logs or counters from the network hardware (when available) for greater insight.

However, obtaining and parsing this information at scale is time-consuming and laborious. Usually, very little code can be reused from previous experiments, and a researcher usually has to write their own code debugging. Further, while captures allow for some debugging, they are passive, and read-only—analyzing them then building and deploying a fix can be very time-consuming.

A previous LATTE paper by Jelvani et al. [6] outlined methods for in-situ debugging of hardware to improve the development process by allowing developers to avoid the costly compilation time for FPGAs. Independently, we are building an API that interfaces with network testbed APIs—specifically FABRIC's—to accomplish a complementary goal. As outlined by Jelvani et al., an in-situ debugger should possess the ability to pause and restart execution, examine and modify state, and observe the effects, and our work is translating these to network equivalents.

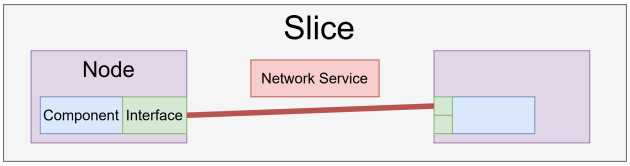

Figure 1: Main elements in a FABRIC experiment.

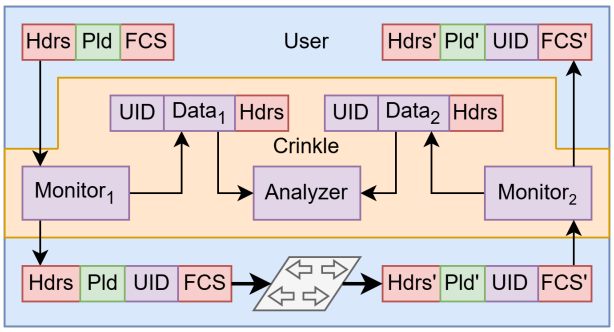

Figure 2: Crinkle provides `Monitors` (which send packet histories and receive configurations) and an `Analyzer` (which processes these histories into a database).

2 Background on FABRIC

Our current prototype is built for the FABRIC testbed, using a fork [11] of the Fablib API [1] for requesting and managing networks for FABRIC experiments.

FABRIC experiments are built as `Slices`, and can be built in whatever configuration a user wishes—provided that the relevant hardware is available. The hierarchy of a `Slice` is shown in Fig. 1. A `Slice` is a collection of `Nodes` representing virtual machines and servers. A `Node` can hold `Components`, which includes (smart) NICs, GPUs, and FPGAs. These components have `Interfaces` which are the endpoints of `Network Services`, which are abstracted Layer 2 and 3 networks that allow `Nodes` to appear directly connected even if physically located across a continent. A `Slice` can span multiple FABRIC sites, the physical datacenters that house the `Nodes` and are present across the US and internationally. `Nodes` and `Network Services` can be removed or added to a `Slice` at any time, provided that the hardware is available.

3 Current Work and Discussion

Flexibility is a key feature of research testbeds like FABRIC—nodes can be effectively removed or inserted at any part of the network

topology. Our system Crinkle exploits this feature to add monitoring at any point in a topology. The user specifies a Layer 2 or Layer 3 link to be monitored, and Crinkle abstracts the installation of a special Node (a `Monitor`) to that link which all traffic crosses. Crinkle then connects these `Monitors` to an `Analyzer` node on an out-of-band IPv6 network, as shown in Fig. 2.

The default functioning of the `Monitors` in-band is to take a packet in, append a trailer containing a unique identifier (if it does not already exist), and then push the packet to the destination. The trailer is appended similarly to the Redundancy Control Trailer of the Parallel Redundancy Protocol [2]; that is, right before the L2 trailer (such as Ethernet's FCS) and without increasing the length fields of other headers, so that to most devices it appears as padding. This enables Crinkle to maintain a history of individual frames without requiring the rest of the network to explicitly support the protocol, though certain network-wide variables such as maximum packet size must account for the extra trailer length. We can thus monitor any device, including physical switches or FPGAs which may not support packet capturing. Out-of-band, the `Monitors` send a short packet containing the `Monitor`'s ID, the packet UID, a timestamp of its arrival, and its header stack to the `Analyzer`, which maintains a database of this information. Preliminary measurements indicate this introduces an overhead of, on average, 2.75 µs for processing and 30 µs including network overhead.

With this packet history, we supply the ability to retrieve and query a history of network traffic from a central source without any explicit support by the user's experiment. The user can query by time, location, header contents, or packet UID to see how specific packets traversed the network, which can reveal the source of network failures.

A research question involves *where* to place these `Monitors` and *what* to have them monitor, and whether this decision-making can be automated for researchers. While working on a network testbed means inserting these monitors requires little work, the finite nature of available resources means a user might wish to maximize the obtainable insight from a fixed resource budget. An ideal system would thus take this budget and information about the network, such as topology information from FABRIC and potential traffic patterns from the user using a query language, and place `Monitors` to maximize coverage within the budget.

These abstract monitoring resources do not have to be fully passive. We can enable the inserted nodes to modify the traffic crossing them. For example, if a researcher notices a network node is misconfigured, they can direct a monitoring node to compensate for that and emit correct traffic. Besides this state modification, Crinkle currently allows users to emit packets from `Monitors`, returning the history of that packet, which makes the process of getting probe packets to a network core and understanding how it is processed significantly easier.

With the ability to monitor and emit packets, Crinkle could be used to restart the execution of a network from a paused point, allowing us to explore "resuming from a breakpoint" in network debugging. FABRIC provides highly accurate time synchronization, so the timestamps of packet arrivals across nodes can be used to control emitting packets during a restart. Further, packet UIDs mean that redundant parts of history following a breakpoint can be removed, grabbing only the history directly before it.

3.1 Integration with Reconfigurable Hardware

To provide better debugging support for network experiments that involve black-box, reconfigurable hardware devices, we are designing a device-abstraction layer that interfaces with vendor-specific toolchains. In turn, this interface leverages vendor-provided CLI and API interfaces through which Crinkle can query and set device-specific values. In turn, these values will be used to service experiment-wide queries or changes, to provide testbed users with an integrated and simplified debugging experience. Crinkle's API will be extended to support debugging multiple FPGAs and ASICs in the live testbed (non-simulated) network environment.

3.2 Benefit to Testbed Operators

With a `Slice` there can be issues users encounter due to their own misconfigurations, or that can result from the FABRIC back-end. While traffic could be dropped on a connection due to a user-introduced issue like incorrect MAC addressing, it would appear identical to if there was an issue on the back-end side that meant the link was not correctly provisioned or forwarding traffic. To the user, in both cases it would appear traffic leaves one node, and does not appear in packet captures on the other. Crinkle would provide insight that allows the user to rule out their own misconfigurations, so that they know they are dealing with a back-end issue and can reach out to the testbed operators. This cuts down on noise; testbed operators are a limited resource and enabling users to resolve more issues on their own allows them to focus more on the issues they are uniquely able to resolve.

4 Related Work

Static analyzers like HSA [7] are very powerful at finding misconfigurations, which are more likely to occur in testbed contexts where the network is commonly built from scratch. However, strong as these methods are, a static analysis cannot reveal the cause of runtime bugs, such as from congestion or hardware or software faults. For that reason, our work is more similar to runtime debuggers such as Hydra [8] and ndb [5]. These generate or monitor data while traffic is flowing and allow for live or post-hoc debugging. The techniques used by those approaches capture runtime bugs, and used properly can reveal misconfigurations like the static analyzers. However, a significant weakness of these published methods is that none are hardware-agnostic; indeed, they all rely on P4 or other programmable hardware. For a research testbed, we believe that generality and allowing for any combination of hardware is essential, as it allows researchers to explore unorthodox and novel techniques. Crinkle addresses a similar problem space in isolating bugs and rapidly prototyping solutions while being hardware- and protocol-agnostic via the use of known machines that can serve as proxies to unknown or black-box interfaces.

5 Conclusion

There is much work to be done to improve the usability of reconfigurable hardware, and this can be seen on testbeds like FABRIC where the complexity and time-scales of using reconfigurable hardware toolchains meets large scale research deployments. We believe that targeting FABRIC is a fruitful and productive endeavour to incubate ideas that can improve the tooling for researchers, and in doing so develop tooling that can be used in more general settings.

Acknowledgments

We thank the reviewers for their feedback. This material is based on work supported by the National Science Foundation (NSF) under Grant CNS-2346499. Any opinions, findings, conclusions, or recommendations in this material are those of the authors and do not necessarily reflect the views of NSF.

References

[1] [n. d.]. Fablib. https://github.com/fabric-testbed/fabrictestbed-extensions

[2] 2021. *IEC 62439-3: Industrial communication networks - High availability automation networks - Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR).*

[3] Ilya Baldin, Anita Nikolich, James Griffioen, Indermohan Inder S Monga, Kuang-Ching Wang, Tom Lehman, and Paul Ruth. 2019. FABRIC: A national-scale programmable experimental network infrastructure. *IEEE Internet Computing* 23, 6 (2019), 38–47.

[4] Hyunsuk Bang, Chris Neely, and Nik Sultana. 2024. In-Network Remote Attestation for ScienceDMZ. https://sc24.supercomputing.org/scinet/network-research-exhibition/accepted-nre-demos/.

[5] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Maziéres, and Nick McKeown. 2012. Where is the debugger for my software-defined network?. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks* (Helsinki, Finland) *(HotSDN '12)*. Association for Computing Machinery, New York, NY, USA, 55âĂŞ60. https://doi.org/10.1145/2342441.2342453

[6] Alborz Jelvani, Richard P. Martin, and Santosh Nagarakatte. 2024. A Case for In-situ Hardware Development. https://capra.cs.cornell.edu/latte24/paper/16.pdf

[7] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header space analysis: static checking for networks. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (San Jose, CA) *(NSDI'12)*. USENIX Association, USA, 9.

[8] Sundararajan Renganathan, Benny Rubin, Hyojoon Kim, Pier Luigi Ventre, Carmelo Cascone, Daniele Moro, Charles Chan, Nick McKeown, and Nate Foster. 2023. Hydra: Effective Runtime Network Verification. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 182âĂŞ194. https://doi.org/10.1145/3603269.3604856

[9] Nik Sultana. 2024. Getting Started with the ESnet SmartNIC framework on FABRIC. http://packetfilters.cs.iit.edu/esnet-smartnic-tutorial/.

[10] Nik Sultana, Yatish Kumar, Chin Guok, James B. Kowalkowski, and Michael H. L. S. Wang. 2024. Shape-shifting Elephants: Multi-modal Transport for Integrated Research Infrastructure. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks, HOTNETS 2024, Irvine, CA, USA, November 18-19, 2024*. ACM, 308–317. https://doi.org/10.1145/3696348.3696855

[11] Alexander Woloseicz. [n. d.]. Crinkle Fablib Fork. https://github.com/awolosewicz/fabrictestbed-extensions/tree/crinkle-dev

[12] Alexander Wolosewicz, Prajwal Somendyapanahalli Venkateshmurthy, and Nik Sultana. 2025. Experience Report: Using the FABRIC Testbed to teach a Graduate Computer Networking course. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE TS 2025, Pittsburgh, PA, USA, 26 February 2025 - 1 March 2025*, Jeffrey A. Stone, Timothy T. Yuen, Libby Shoop, Samuel A. Rebelsky, and James Prather (Eds.). ACM, 1246–1252. https://doi.org/10.1145/3641554.3701923