DEMO: P4 Replay (P4R): Reproducing Packet Traces and Stateful Connections at Line-Rate on Your P4-capable Hardware

Francisco Germano Vogt
Universidade Estadual de Campinas (Unicamp)

Fabricio Rodriguez
Universidade Estadual de Campinas (Unicamp)

Filipo Gabert Costa
Universidade Estadual de Campinas (Unicamp)

Christian Esteve Rothenberg
Universidade Estadual de Campinas (Unicamp)

Marcelo Caggiani Luizelli
Federal University of Pampa (Unipampa)

Gyanesh Patra
Gergely Pongrácz
Ericsson Research

Abstract

Network testers are essential for assessing network performance. They are used to generate and capture test packets to evaluate the network's correctness and efficiency. However, evolving demands, such as generating realistic, high-performance workloads, pose challenges for existing solutions. In this demonstration, we present P4 replay (P4R), a network tester based on a programmable switch ASIC capable of reproducing real packet traces and establishing high-performance stateful TCP connections. P4R can test external clients and servers with high throughput and accuracy as well as self-testing P4 applications running in parallel on the same device.

CCS Concepts

• **Networks** → **Network experimentation**; **Network performance analysis**; **Programmable networks**.

Keywords

P4, SDN, Network Testing, Traffic Generation

ACM Reference Format:
Francisco Germano Vogt, Fabricio Rodriguez, Filipo Gabert Costa, Christian Esteve Rothenberg, Marcelo Caggiani Luizelli, Gyanesh Patra, and Gergely Pongrácz. 2024. DEMO: P4 Replay (P4R): Reproducing Packet Traces and Stateful Connections at Line-Rate on Your P4-capable Hardware . In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM Posters and Demos '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3672202.3673743

1 Introduction

The complexity of modern networks requires controlled traffic generation to accurately evaluate the efficiency and scalability [3, 13]. With the emergence of programmable networks, there is a growing need for realistic traffic generation to simulate real-world scenarios [6] accurately. The traffic generation tools should deliver ever-increasing throughput performance, flexible and accurate control, and precise time-stamping to time verification (i.e., latency, jitter). Lately, switch-based approaches [2, 5, 14] have emerged to provide network testing at Tbps with precise control and time-stamping. However, current solutions still have several limitations, particularly in replicating real-world traffic patterns. While P4TG [5] and PIPO-TG [2] support scripted traffic patterns, they do not support reproducing real traffic captures and establishing stateful connections. HyperTester [14] only supports it in a limited way (e.g., without handshaking and flow control, just waiting for ACKs to send traffic), in addition to requiring auxiliary CPU support to generate traffic and not being open-source.

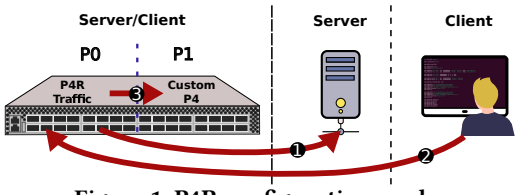

Figure 1: P4R configuration modes.

In this demo, we present P4 Replay (P4R) [12] as a high-end traffic generation tool that overcomes limitations from state-of-the-art Tofino-based traffic generators. P4R benefits from the Tofino traffic generation capabilities to replicate real-world traffic patterns while maintaining high performance and accuracy. The user/network tester can use P4R to reproduce pre-captured traces (i.e., PCAPs) and create stateful TCP connections at the Tofino line rate. P4R can be used in three configuration modes (see Fig. 1): client, server, or internal. In client mode ❶, P4R can reproduce PCAPs or establish TCP connections with a connected server; in server mode ❷, P4R responds to TCP connections from connected clients; and in internal mode ❸, P4R can send packet traces or TCP connections to test custom P4 code running in parallel, in another pipeline. Armed with these features, P4R emerges as a powerful tool for realistic network experimentation, implementing the most challenging features not present in state-of-the-art Tofino-based traffic generators [2, 5, 14].

2 Motivation & Challenges

The current generation of programmable switches and 100-to-400Gb network interface cards (NICs) [7, 8] are constantly reshaping the network testers' performance requirements. Generating traffic at hundreds of Gbps per port is supported only by specific hardware, solutions based on DPDK [4] and netmap [9], or solutions based on programmable switches. While solutions based on specific hardware and FPGA are expensive and inflexible, solutions based on DPDK, such as Trex [11], depend on the associated CPU and may require many cores to achieve the desired throughput.

These limitations have attracted attention to traffic generators based on programmable switches [1, 2, 5, 14]. While these generators can produce Tbps of traffic with high precision, they struggle

ACM SIGCOMM Posters and Demos '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0717-9/24/08
https://doi.org/10.1145/3672202.3673743

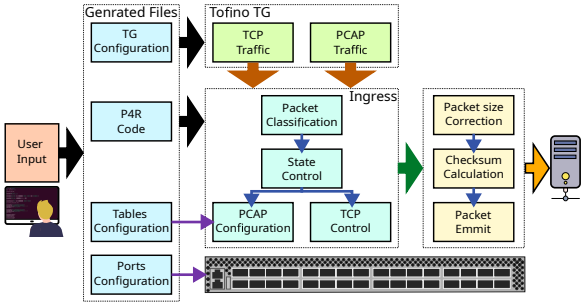

Figure 2: P4R Architecture.

with replicating realistic traffic patterns. P4TG and PIPO-TG [2, 5] only support the reproduction of simple traffic patterns, unable to reproduce packet traces and stateful connections. HyperTester [14] supports playback PCAPs with the limitation that they contain a single flow and the reproduction of TCP flows only in a stateless manner. More recently, Norma [1] proposed support for stateful connections on programmable switch ASICs; however, Norma does not support the reproduction of PCAP packet traces, works only in client mode, and does not support testing a P4 code internally, in addition to not providing open-source code.

Addressing the limitations of current network testers requires tackling several design and implementation challenges to make these features feasible in current programmable ASICs:

Limited memory. Current programmable ASICs have limited memory, typically only dozens of MB, to store state information. To replay PCAPs using just the ASIC (without an auxiliary CPU like HyperTester), we need to fit the PCAP packet information within this restricted memory. Similarly, all connection state information for stateful connections must be stored in the registers.

Pipeline constraints. To test a parallel P4 code with the generated traffic, we need to use just one pipeline for traffic generation and control. Unlike Norma, which utilizes multiple pipelines for generating and controlling network traffic, we have fewer recirculation ports, memory, and pipeline stages to implement our solution.

3 P4R Architecture & Implementation

P4R simplifies realistic and customized high-performance network tests by allowing users to configure the traffic generator using a user-friendly script. The script to define P4R traffic is Python-based, similar to Scapy and other Python-based packet manipulators. P4R auto-generates all necessary files from the defined traffic characteristics to configure the switch and start the traffic generation. The user inputs include the operation mode, traffic configurations, and the user's P4 code in the case of internal mode. P4R has the three different configuration modes described below:

Client mode. P4R can instantiate clients to send traffic (i.e., packet traces) or establish stateful TCP connections with external servers.

Server mode. P4R will act as a stateful TCP server (e.g., iperf) and establish connections with multiple clients.

Internal mode. The traffic generated by P4R is internally routed to a user's P4 code running in parallel in another pipeline.

These three P4R modes can be used individually to reproduce packet traces and establish stateful connections. Figure 2 illustrates the P4R architecture and workflow. For the PCAP reproduction, the user can only use the client and internal mode, and in addition to the desired PCAP file, the user can choose two playback modes: *timed* and *throughput*. In the timed mode, the packets will be sent in order concerning the timestamps in the file. In throughput mode, the packets will be sent in the PCAP order, respecting the throughput defined in the user input file. For the TCP stateful connections, the user can select any of the three configuration modes and even combine internal with client or server modes.

Table 1: P4R P4/TNA implementation approaches.

Task	Implementation approach
Packet Generation	Tofino's native traffic generation unit, periodic/one-shot trigger
Throughput Definition	Tofino port shaping or meters (user selection)
PCAP Reproduction	Registers, table entries, and random payload
TCP Stateful Connection	Registers to save connection states and template packets
Support for user P4 Code	Multi pipeline support and recirculations
Packet Size Correction	Tofino mirror extern to truncate the packets

P4R uses the input traffic patterns to generate four configuration files: (*i*) Tofino traffic generation, (*ii*) P4R P4 code, (*iii*) Tables, and (*iv*) Ports. These files are used to configure and start running the switch and Tofino's native traffic generator. Furthermore, we configured all the ingress pipeline tables and registers, which are responsible for storing and controlling the states of active connections. This design enables the swapping of pipelines through the traffic manager, allowing the generated traffic to be forwarded to the pipeline containing the user's P4 code. Table 1 summarizes the tasks and implementation approaches.

Challenges. P4R only captures the headers of packet traces, which are restricted to a maximum of 120 bytes and may not include all packet protocols. Additionally, we only establish stateful connections with iperf, and we do not have a comprehensive analysis of the true scalability of the number of connections supported by P4R.

4 Conclusions and Future Work

This demo shows how P4R contributes to the ecosystem of realistic high-performance network testing. Using a programmable switch ASIC, P4R can reproduce advanced traffic patterns, establish high-throughput TCP connections, and be the first strategy to reproduce packet traces using the ASIC capabilities. As an open-source, user-friendly traffic generator, P4R provides a valuable tool for network testing and advanced programmable networking research, enabling self-testing P4-based solutions with our internal mode.

Future work. First, we plan to formalize the performance and scalability limits of the traffic generated by P4R. We also want to consider the implementation of P4R in the most recently programmable ASIC models like Tofino 2 and evaluate how this extends P4R capabilities. Furthermore, we plan to include other features in P4R to support different types of stress tests, such as SYN flood and microbursts, and even analyze the feasibility of including the QUIC protocol for stateful connections. Finally, we consider integrating our traffic generator with the ASIC-based P7 network emulator [10], enabling traffic generation, emulation of network topologies, and testing P4 codes on a single switch.

5 Acknowledgments

This work was supported by Ericsson Telecomunicações Ltda., and by the Sao Paulo Research Foundation (FAPESP), grant 2021/00199-8 (CPE SMARTNESS), 2020/05115-4, and 2020/05183-0. This study was also partially funded by CAPES, Brazil-Finance Code 001, and CNPq (404027/2021-0).

References

[1] Yanqing Chen, Bingchuan Tian, Chen Tian, Li Dai, Yu Zhou, Mengjing Ma, Ming Tang, Hao Zheng, Zhewen Yang, Guihai Chen, et al. 2023. Norma: Towards Practical Network Load Testing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1733–1749.

[2] Filipo Gabert Costa, Francisco Vogt, Fabricio Rodriguez, Ariel Góes de Castro, Marcelo Caggiani Luizelli, and Christian Esteve Rothenberg. 2024. PIPO-TG: Parameterizable High-Performance Traffic Generation. In *To appear in IEEE/IFIP Network Operations and Management Symposium (NOMS) 2024*.

[3] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. Moongen: A scriptable high-speed packet generator. In *Proceedings of the 2015 Internet Measurement Conference*. 275–287.

[4] Linux Foundation. 2015. Data Plane Development Kit (DPDK). http://www.dpdk.org

[5] Steffen Lindner, Marco Häberle, and Michael Menth. 2023. P4TG: 1 Tb/s Traffic Generation for Ethernet/IP Networks. *IEEE Access* 11 (2023), 17525–17535.

[6] Wai-xi Liu, Cong Liang, Yong Cui, Jun Cai, and Jun-ming Luo. 2022. Programmable data plane intelligence: advances, opportunities, and challenges. *IEEE Network* (2022).

[7] Mellanox. 2020. 200Gb/s ConnectX-6 Ethernet Single/Dual-Port Adapter IC. Online. https://www.mellanox.com/products/ethernet-adapter-ic/connectx-6-en-ic

[8] NVIDIA. 2021. NVIDIA BlueField-2. https://resources.nvidia.com/en-us-accelerated-networking-resource-library/bluefield-2-dpu-datasheet [Access: Feb 16, 2024].

[9] Luigi Rizzo. 2012. Netmap: a novel framework for fast packet I/O. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference* (Boston, MA) *(USENIX ATC'12)*. USENIX Association, USA, 9.

[10] Fabricio Rodriguez, Francisco Germano Vogt, Ariel Góes De Castro, Marcos Felipe Schwarz, and Christian Rothenberg. 2022. P4 programmable patch panel (p7) an instant 100g emulated network on your tofino-based pizza box. In *Proceedings of the SIGCOMM'22 Poster and Demo Sessions*. 4–6.

[11] Team TRex. 2023. TRex Realistic Traffic Generator. https://trex-tgn.cisco.com/ Available: https://trex-tgn.cisco.com/ [Acces: March 20, 2024].

[12] Francisco Germano Vogt, Fabricio Rodriguez, Filipo Gabert Costa, Marcelo Caggiani Luizelli, Christian Esteve Rothenberg, Gyanesh Patra, and Gergely Pongracz. 2024. P4 Replay (P4R). https://github.com/intrig-unicamp/P4R.

[13] Yu Zhou, Jun Bi, Yunsenxiao Lin, Yangyang Wang, Dai Zhang, Zhaowei Xi, Jiamin Cao, and Chen Sun. 2019. P4Tester: Efficient runtime rule fault detection for programmable data planes. In *Proceedings of the International Symposium on Quality of Service*. 1–10.

[14] Yu Zhou, Zhaowei Xi, Dai Zhang, Yangyang Wang, Jinqiu Wang, Mingwei Xu, and Jianping Wu. 2019. Hypertester: high-performance network testing driven by programmable switches. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 30–43.