# P4R: Scaling Stateful Network Testing and Trace Replay with Nanosecond-level Accuracy

**Francisco Germano Vogt[1], Zhiheng Yang[3], Leonardo Henrique Guimarães[1], Fabricio Eduardo Rodriguez Cesen[4], Sergio Rossi Brito da Silva[1], Marcelo Caggiani Luizelli[2], Chrysa Papagianni[3], Christian Esteve Rothenberg[1]**

[1]Universidade Estadual de Campinas (UNICAMP), Campinas, SP, Brazil

[2]Universidade Federal do Pampa (UNIPAMPA), Alegrete, RS, Brazil

[3]University of Amsterdam (UvA), Amsterdam, Netherlands

[4]Telefonica Research, Barcelona, Spain

f234632@dac.unicamp.br, z.yang@uva.nl, l247225@dac.unicamp.br, fabricio.rodriguezcesen@telefonica.com, s217042@dac.unicamp.br, marceloluizelli@unipampa.edu.br, c.papagianni@uva.nl, chesteve@unicamp.br

***Abstract.*** *As network speeds transition to 100 Gbps and beyond, traditional software-based traffic generators face significant performance and accuracy bottlenecks due to CPU and PCIe limitations. While programmable switch-based testers have emerged to provide line-rate throughput, most current solutions are either limited to stateless traffic generation or rely on simplified stateful models, such as "stop-and-wait" mechanisms with static one-packet windows. This paper presents an evolved version of* ***P4R*** *(P4 Replay), an open-source framework designed for high-fidelity stateful network testing on programmable ASICs. P4R overcomes previous design constraints by introducing a dynamic window management system for TCP connections, enabling more realistic flow emulation compared to fixed-window architectures. Furthermore, P4R implements a dual-mode trace replay engine capable of reproducing pre-captured PCAP files with nanosecond-level timing accuracy or user-defined throughput. Supporting client, server, and a unique internal self-testing mode, P4R allows for the evaluation of complex P4 applications and external network functions at line-rate. We demonstrate that P4R bridges the gap between synthetic traffic and realistic network experimentation, providing the community with a versatile and precise tool for modern networking research.*

## 1. Introduction

High-performance network testing is fundamental for the development and validation of modern cloud-scale infrastructures. As network speeds transition from 100 Gbps to 400 Gbps and beyond, traditional software-based testers struggle to maintain accuracy due to CPU bottlenecks and unpredictable jitter. While hardware-based testers provide the required performance, they often lack the flexibility to support customized protocols or the emerging need for "in-switch" self-testing in programmable data planes.

Recently, programmable switch ASICs have emerged as a promising platform for line-rate traffic generation[Chen et al. 2023, Costa et al. 2024, Lindner et al. 2023, Vogt et al. 2025, Bosshart et al. 2014]. Tools like HyperTester and Norma have demonstrated the feasibility of generating Tbps-level traffic with stateful capabilities. However, existing switch-based solutions still face significant limitations in fidelity and realism. Most stateful testers rely on a simplified "stop-and-wait" mechanism, sending a single packet and waiting for its acknowledgment, which fails to emulate the complex dynamics of real-world TCP congestion and flow control. Furthermore, reproducing real-world packet traces (PCAPs) at line-rate while maintaining strict timing accuracy remains a challenge for current ASIC-based architectures.

In this work, we present an evolved version of P4R, an open-source network testing framework designed to bridge the gap between synthetic traffic generation and realistic network emulation. P4R extends the capabilities of programmable ASICs to support high-fidelity trace replay and advanced stateful connection management. Unlike its predecessors, P4R introduces a dynamic window management system for TCP connections, moving beyond fixed-window abstractions to allow for a more realistic "in-flight" packet volume. Additionally, P4R implements a dual-mode PCAP replay engine: a throughput mode for maximum stress testing and a timestamp mode that enforces nanosecond-level inter-packet gaps derived from real-world captures. To ensure practical usability, P4R integrates a real-time performance monitoring interface and supports multiple operational modes, including a unique "internal mode" for self-testing P4 applications within the same device. We demonstrate that P4R can scale to Tbps throughput while maintaining the precision required for fine-grained latency and jitter analysis. Our key contributions are:

- **Nanosecond-level Trace Replay:** A multi-mode engine capable of reproducing complex packet traces with strict timing or throughput-driven constraints.
- **Dynamic Window Stateful TCP:** A stateful responder that supports configurable dynamic windows, enabling more realistic emulation of session dynamics compared to previous stop-and-wait models.
- **Integrated Monitoring & Usability:** A comprehensive framework that combines high-performance data plane implementation with a user-friendly monitoring interface and automated configuration scripts.
- **Open-Source Versatility:** A fully open-source tool that supports client, server, and internal testing modes for programmable hardware.

## 2. Motivation and Related Work

### 2.1. Motivation

The evolution of cloud-scale networking toward 400 Gbps and beyond has rendered traditional software-based traffic generators increasingly inadequate. Software testers, such as MoonGen [Emmerich et al. 2015] and TRex [TRex ], are fundamentally constrained by the PCIe bandwidth and CPU processing bottlenecks. Evaluations show that these solutions cannot exceed 300 Gbps, and for small packet workloads, performance can drop to as low as 40 Gbps even when utilizing multiple CPU cores. Beyond throughput, accuracy remains a significant concern: MoonGen relies on NIC-level hardware meters that lack fine-grained precision for all target rates, while TRex uses software-based timestamps, leading to unstable inter-packet gaps and jitter.

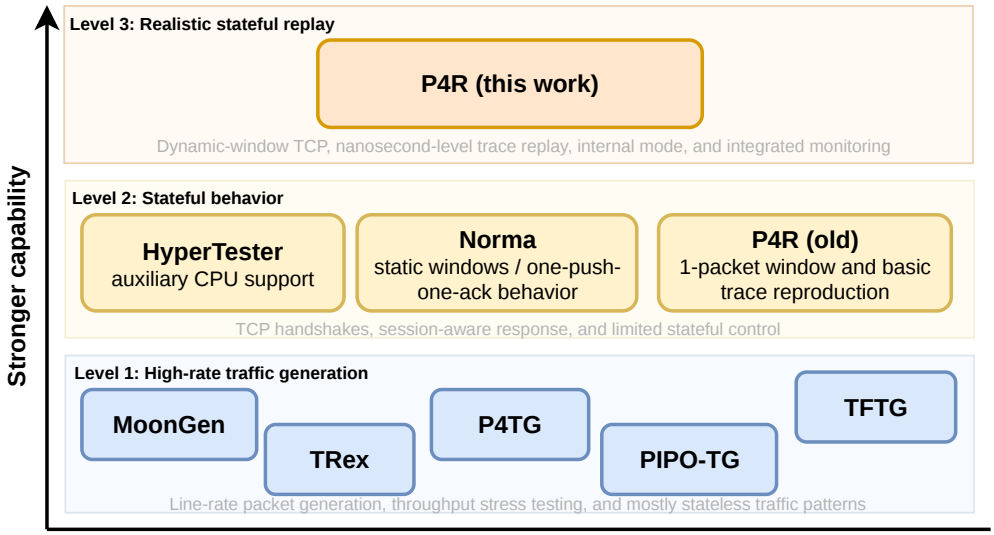


**Figura 1. Capability gap addressed by P4R.**

To overcome these limitations, the research community has turned to programmable switch ASICs. Solutions like P4TG [Lindner et al. 2023], PIPO-TG [Costa et al. 2024], and TFTG [Vogt et al. 2025] leverage the Tofino architecture to achieve Tbps-level traffic generation with high precision. However, these tools are primarily designed for stateless traffic patterns, lacking the flexibility to maintain session-aware protocols. While HyperTester [Zhou et al. 2019] and Norma [Chen et al. 2023] introduced stateful capabilities, they still face critical trade-offs. HyperTester requires auxiliary CPU support to manage connection states, whereas Norma implements a rigid stateful responder based on "one-push-one-ack" sequences. This approach prevents the implementation of sophisticated congestion control or dynamic window management, as it only allows for a single packet to be in-flight at a time. Furthermore, to the best of our knowledge at the time of writing, none of these leading stateful testers are available as open-source projects.

P4R [Vogt et al. 2024] was introduced to address these gaps by enabling both stateful connection establishment and realistic trace replay at scale. Although the initial prototype was recognized at the ACM SIGCOMM 2024 Student Research Competition, it was limited by a 1-packet window and basic trace reproduction features. This work extends the P4R framework to support nanosecond-level accuracy in trace replay and truly dynamic window management. Figure 1 summarizes this capability gap and highlights the position of P4R in the design space of high-speed traffic generation and replay systems.

### 2.2. Related Work

**Software-based Testers:** MoonGen [Emmerich et al. 2015] and TRex [TRex ] utilize DPDK to bypass the kernel and generate high-performance traffic. These tools are highly flexible, supporting a wide range of protocols and even certain forms of stateful traffic generation. However, these solutions are fundamentally constrained by CPU and PCIe bottlenecks, which makes it difficult to sustain very high rates under realistic workloads.

**Stateless Switch Testers:** P4TG [Lindner et al. 2023] and PIPO-TG [Costa et al. 2024]

provide high-performance 1 Tbps generation for Ethernet/IP networks but do not support stateful handshaking or trace reproduction. TFTG [Vogt et al. 2025] focuses on high time-fidelity for traffic patterns on Tofino hardware, but even providing nanosecond-level accuracy, it does not support any stateful protocol.

**Stateful Switch Testers:** HyperTester [Zhou et al. 2019] was a pioneer in using Tofino for stateful tests but relies on non-ASIC components for connection control, which compromise the performance. Norma [Chen et al. 2023] provides a robust stateful responder using an EFSM abstraction but is limited by static windows, supporting just a transmit-and-wait version of the stateful protocols (sending one packet and waiting for one ack), in addition to being a closed-source implementation.

**Packet Replay and P4R:** The initial P4R [Vogt et al. 2024] proposed a novel method for reproducing traces and stateful connections using only the ASIC's capabilities. This demo proved the feasibility of "internal mode" testing, where a single switch can test its own P4 applications in parallel pipelines. However, in the first version, P4R also just supported a transmit-and-wait version of TCP. In this work, we extend these capabilities allowing dynamic windows, and also improving the capabilities of the PCAP reproduction feature.

## 3. P4 Replay Overview

P4R is designed as a comprehensive framework for high-fidelity network testing, moving beyond simple packet generation to realistic protocol emulation. The system leverages the Tofino architecture to maintain stateful consistency and timing accuracy at line rate.

### 3.1. Architecture overview

The P4R architecture, illustrated in Figure 2, is divided into a Python-based control plane and a high-performance P4 data plane. The workflow begins with a user-friendly script, similar to Scapy, where the user defines traffic characteristics, operational modes, and trace files.

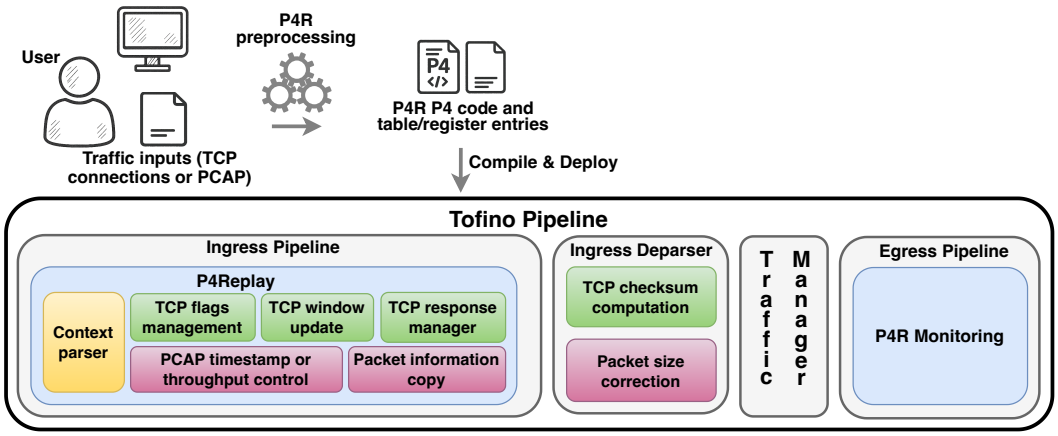


**Figura 2. P4R architecture and workflow.**

Based on this input, P4R auto-generates four critical configuration components:

- **Tofino TG Configuration:** Parameters for the native traffic generation unit.

- **P4R Data Plane Code:** The P4 logic responsible for state control and packet modification.
- **Table Configurations:** Match-action entries for flow classification and session management.
- **Port Configurations:** Front-panel and loopback port setups.

The data plane utilizes the Ingress pipeline for core logic, including packet classification (identifying if it is a pcap packet or TCP), state control (behaving according to each TCP flag and window size), and checksum calculation. The PCAP reproduction filter also controls the exact timestamp that the packet should be forwarded (if timestamp mode is used), or the desired throughput (if throughput mode is used). For Internal Mode, P4R uses the traffic manager to route generated traffic between pipelines, allowing parallel testing of custom P4 applications on the same device.

### 3.2. Operational Modes

P4R is designed to support multiple testing workflows. Rather than exposing a single fixed deployment model, it can be configured to operate in three complementary modes, depending on whether the switch is used to initiate traffic, respond to external traffic, or test a P4 program internally on the same device. Table 1 summarizes the operational role of P4R in each case.

**Tabela 1. Operational Modes Supported by P4R**

| Mode | Role of P4R | Experimental Goal | Typical Use Case |
|---|---|---|---|
| Client | Traffic initiator | Benchmark an external Server Under Test (SUT) | Sends TCP sessions or replay traces to an external target |
| Server | Stateful responder | Emulate a stateful endpoint | Handles handshakes and data exchange with external clients |
| Internal | Self-testing engine | Validate P4 logic in-switch | Generates traffic in one pipeline and forwards it to another |

### 3.3. Main Features

#### 3.3.1. Nanosecond-level Trace Replay.

Unlike existing switch-based generators that struggle with timing fidelity, P4R introduces two distinct playback modes for PCAP traces:

**Timestamp Mode:** Strictly follows the original inter-packet gaps (IPGs) recorded in the trace file with nanosecond-level precision. This is achieved by leveraging Tofino's high-precision clock to schedule packet emission.

**Throughput Mode:** Replays the trace packets in their original order but scales the timing to meet a user-defined target throughput, ideal for stress-testing network buffers.

#### 3.3.2. Window-Aware Stateful TCP

A major contribution of this work is the transition from static "stop-and-wait"models to Dynamic Window Management. While previous state-of-the-art systems like Norma

were limited to a fixed sequence of one PUSH followed by one ACK, P4R utilizes ASIC registers to track the number of in-flight packets.

**Dynamic Flow Control:** P4R manages a configurable window size, allowing it to burst multiple packets before requiring an acknowledgment.

**Stateful Handshaking:** The system fully handles the TCP three-way handshake and session teardown at line-rate.

#### 3.3.3. Integrated Monitoring and Usability

To improve the visibility of testing tasks, P4R now includes an Integrated Performance Monitoring interface. This module tracks real-time metrics such as throughput ($Gbps$), packet rate ($Mpps$), and active session counts directly from the data plane counters. Additionally, improvements in documentation and automated scripts ensure that the tool is accessible for rapid prototyping in both academic and industrial environments.

### 3.4. Workflow and Pre-processing

The operational workflow of P4R is divided into a proactive pre-processing phase and a real-time execution phase (see Figure 2). This separation ensures that complex computations, such as parsing multi-gigabyte traces, do not interfere with the nanosecond-precision required during replay.

- **Input Parsing and Analysis:** The Python control plane analyzes the provided PCAP files or TCP connection requirements. For traces, it extracts inter-packet gaps (IPGs) and header fields. For TCP, it maps the desired connections into stateful table structures.
- **Resource Mapping:** Each packet in the trace is converted into a specific table entry or register value. P4R maps the connection states (e.g., sequence numbers, window sizes) into the ASIC's register arrays to allow for dynamic updates during the handshake and data transfer phases.
- **Deployment and Triggering:** Once the tables are populated via gRPC or Thrift, the Tofino traffic generation unit is configured to trigger the initial packets. The data plane then takes over, maintaining the state machine and timing without further control-plane intervention.

## 4. Implementation Details

The core of P4R's high-fidelity replay lies in its efficient use of the RMT (Reconfigurable Match-Action Table) pipeline resources.

- **Dynamic Window Engine:** To move beyond the "stop-and-wait"limitation, P4R utilizes a set of register arrays to track the state of active sessions. For each flow, the system maintains the *last_seq_sent* and *highest_ack_received*. The "window-aware" logic calculates the current in-flight volume in real-time; new packets are only emitted if the difference remains below the user-defined window size. The recovery mode if a packet is lost (and therefore its ack does not arrive) is a go-back-N approach, where we retransmit all packets from the last ack received.

- **Nanosecond Timing Control:** For the *Timestamp Mode*, P4R leverages the Tofino's intrinsic metadata timestamps. By comparing the egress global clock with the scheduled departure time stored in registers, the deparser can delay or truncate packets to maintain sub-microsecond accuracy in inter-packet gaps.

### 4.1. Constraints and Limitations

While P4R provides a significant leap over previous stateful testers, it operates within the hardware constraints of current programmable ASICs:

- **Header-only Reproduction:** Due to PHV (Packet Header Vector) limitations and parser constraints, P4R currently reproduces only the packet headers from PCAP traces, with a maximum depth of 120 bytes. While sufficient for most L4-L7 testing, full payload replay is not yet supported.
- **Memory Bounds:** The maximum number of packets in a single trace replay is currently capped at 100,000. This limit is imposed by the available SRAM allocated to the packet information copy registers to ensure the application coexists with other P4 logic.
- **Congestion Control:** The stateful TCP implementation currently relies on manual window sizing. The integration of dynamic congestion control algorithms (e.g., Cubic or BBR) is computationally intensive for the RMT pipeline and remains part of our future work.

## 5. Documentation, Code, and Demonstration

The P4R framework is fully open-source, aiming to foster the development of realistic, high-performance network testing within the programmable networks community. All source code, including the P4 data plane implementations, Python control plane scripts, and comprehensive documentation, is available at `https://github.com/intrig-unicamp/P4R`. The repository includes a detailed manual for installation and usage, ensuring that the primary features presented in this work, such as nanosecond-precision trace replay and dynamic window-aware stateful TCP, can be fully reproduced. Additionally, a video tutorial demonstrating the tool's configuration and its performance under line-rate conditions is also available at the repository. We actively encourage community engagement through contributions, issue reporting, and expanding the tool's support for new stateful protocols.

## 6. Final Remarks

In this paper, we present the evolution of **P4R**, an open-source network testing framework designed for high-fidelity stateful traffic generation and trace replay at line rate. By leveraging the capabilities of programmable switch ASICs, P4R overcomes the performance and accuracy limitations inherent in software-based solutions. Our implementation introduces a dynamic window management system for stateful TCP connections, effectively moving beyond the rigid "stop-and-wait" models found in previous state-of-the-art hardware testers. Additionally, the dual-mode replay engine provides the community with a unique tool capable of reproducing complex PCAP traces with nanosecond-level timing accuracy. The versatility of P4R is further highlighted by its "internal mode,"which enables researchers to perform self-testing of P4 applications in parallel pipelines within the

same device. This capability, combined with a user-friendly configuration interface and an open-source codebase, positions P4R as a practical and accessible alternative for both academic research and industrial performance validation.

As for future work, we plan to expand the framework's protocol support to include stateful protocols such as QUIC and RDMA. We also intend to formalize the scalability limits regarding the maximum number of simultaneous connections supported by the ASIC's memory. Finally, we explore the integration of P4R with network emulators like P7 [Rodriguez et al. 2022] to provide a unified environment for traffic generation and topology emulation on a single hardware platform.

## Acknowledgments

This work was supported by Ericsson Telecomunicações Ltda., and by the Sao Paulo Research Foundation (FAPESP), grant 2021/00199-8 (CPE SMARTNESS), 2024/16207-8, 2020/05183-0, and 2023/00794-9. Also, it was supported by Foundation for Research of the State of Rio Grande do Sul (24/2551-0001394-6), CAPES, Brazil-Finance Code 001, and CNPq (405809/2025-5). Finally, this work was also supported by the Dutch National Growth Fund through the 6G flagship project "Future Network Services".

## Referências

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

Chen, Y., Tian, B., Tian, C., Dai, L., Zhou, Y., Ma, M., Tang, M., Zheng, H., Yang, Z., Chen, G., et al. (2023). Norma: Towards practical network load testing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1733–1749.

Costa, F. G., Vogt, F. G., Cesen, F. R., de Castro, A. G., Luizelli, M. C., and Rothenberg, C. E. (2024). Pipo-tg: Parameterizable high-performance traffic generation. In *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, pages 1–9. IEEE.

Emmerich, P., Gallenmüller, S., Raumer, D., Wohlfart, F., and Carle, G. (2015). Moongen: A scriptable high-speed packet generator. In *Proceedings of the 2015 Internet Measurement Conference*, pages 275–287.

Lindner, S., Häberle, M., and Menth, M. (2023). P4tg: 1 tb/s traffic generation for ethernet/ip networks. *IEEE Access*, 11:17525–17535.

Rodriguez, F., Vogt, F. G., De Castro, A. G., Schwarz, M. F., and Rothenberg, C. (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*, SIGCOMM '22, page 4–6.

TRex. Cisco TRex Traffic Generator. `https://trex-tgn.cisco.com/`. Online; accessed February 2026.

Vogt, F. G., Da Silva, S. R. B., Cesen, F. E. R., Costa, F. G., Luizelli, M. C., and Rothenberg, C. E. (2025). Tftg: Time fidelity traffic generation through p4/tofino programmable hardware. *IEEE Network*.

Vogt, F. G., Rodriguez, F., Costa, F. G., Luielli, M. C., Rotenberg, C. E., Patra, G., and Pongracz, G. (2024). P4 replay (p4r): Reproducing packet traces and stateful connections at line-rate on your p4-capable hardware. In *Proceedings of the ACM SIGCOMM 2024 Conference: Posters and Demos*, pages 122–124.

Zhou, Y., Xi, Z., Zhang, D., Wang, Y., Wang, J., Xu, M., and Wu, J. (2019). Hypertester: high-performance network testing driven by programmable switches. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 30–43.