Enhancements to P4TG: Protocols, Performance, and Automation

Fabian Ihle*, Etienne Zink*, Steffen Lindner*, Michael Menth*
*University of Tübingen, Chair of Communication Networks
Email: {fabian.ihle, etienne.zink, steffen.lindner, menth}@uni-tuebingen.de

***Abstract*—P4TG is a hardware-based traffic generator (TG) running on the Intel Tofino™ 1 ASIC and was programmed using the programming language P4. In its initial version, P4TG could generate up to $10\times100\,\mathrm{Gb/s}$ of traffic and directly measure rates, packet loss, and other metrics in the data plane. Many researchers and industrial partners requested new features to be incorporated into P4TG since its publication in 2023. With the recently added features, P4TG supports the generation of packets encapsulated with a customizable VLAN, QinQ, VxLAN, MPLS, and SRv6 header. Further, generation of IPv6 traffic is added and P4TG is ported to the Intel Tofino™ 2 platform enabling a generation capability of up to $10\times400\,\mathrm{Gb/s}$. The improvement in user experience focuses on ease of operation. Features like automated ARP replies, improved visualization, report generation, and automated testing based on the IMIX distribution and RFC 2544 are added. Future work on P4TG includes NDP to facilitate IPv6 traffic, and a NETCONF integration to further ease the configuration.**

***Index Terms*—Data Plane Programming, Network Testing, P4, Traffic Generator**

I. Introduction

Traffic generators (TGs) are used to test network devices or applications. Typically, TGs generate frames with different sizes, rates, traffic patterns, and packet headers to mirror the behavior of real-world traffic. For network testing, TGs further implement measurements of TX/RX rates, round trip times (RTTs), packet loss, and out-of-order packets. TGs can either be implemented in software or hardware. They come with several different trade-offs. Software-based TGs are cost-effective, customizable, and run on commodity hardware [1]. However, they are not as accurate and performant as hardware-accelerated TGs that use specialized hardware components [2], [3]. These components result in better performance and higher accuracy while being less flexible and more expensive.

Programmable switches facilitate the development of hardware-supported tools at affordable cost which was the initial motivation for the development of the P4-based traffic generator (P4TG) [4]. P4TG leverages the capabilities of the Intel Tofino™ switching ASIC for traffic generation in Ethernet/IP networks.

In its initially published version, P4TG supported customizable Ethernet and IPv4 headers. However, to effectively test productive networks, more diverse traffic patterns are needed. They include enhanced traffic generation capabilities such as new protocols and higher data rates. In this paper, we give an overview of the enhancements to P4TG, including VLAN, QinQ, VxLAN, SRv6 and MPLS encapsulation, IPv6 support, and up to $4\,\mathrm{Tb/s}$ of traffic generation. Further, the enhancements to P4TG improve the user experience by facilitating automated network testing through report generation and test profiles. This demonstrates another advantage of a self-programmed hardware-based TG, namely ease of extensibility.

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/3-1. This work was supported by the bwNET 2.0 project which is funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

II. The Traffic Generator P4TG

The traffic generator P4TG is a hardware-based TG implemented for the Intel Tofino™ switching ASIC [4]. The source code of P4TG is open-source on GitHub [5]. P4TG features custom traffic generation and measurement in the data plane. A generated packet in P4TG contains a UDP header, a P4TG header, and a customizable Ethernet and IPv4 header. The P4TG header contains a sequence number, transmission timestamp, and stream ID for measurements. P4TG measures rates, frame types, sizes, as well as lost and out-of-order packets in the data plane. Further, RTTs and inter-arrival times are sampled in the control plane. Various frame sizes ranging from $64\,\mathrm{B}$ (byte) to $9000\,\mathrm{B}$ are supported. The frames are padded with random data if the frame size exceeds the configured cumulative header size.

For traffic generation, the internal traffic generation ports of the Intel Tofino™ are leveraged [6]. Up to two additional loopback ports per physical port are required for traffic customization and measurement. Therefore, P4TG can use ten physical ports for traffic generation on a 32-port switch enabling a bandwidth of $1\,\mathrm{Tb/s}$. The generated traffic is either constant bit-rate (CBR) or Poisson (random) traffic. Up to seven CBR streams or one Poisson stream are configurable for traffic generation.

III. Enhancements to P4TG

Since its publication in 2023, P4TG has been extended with many features requested by academic and industrial partners. This section provides an overview of the new features, explains them in detail, and divides them into two areas: enhanced traffic generation capabilities and improved user experience.

TABLE I
P4TG FEATURE OVERVIEW.

Existing	New		Experimental	Planned
• Ethernet	• IPv6	• VLAN	• Test profiles	• NDP
• IPv4	• QinQ	• VxLAN	• File reporting	• NETCONF
• Statistics	• MPLS	• SRv6	• Auto. testing	
• Tofino 1	• ARP replies	• Tofino 2	• Localization	
• Web frontend	• Rust backend	• Dark mode		
• Python backend				

A. Overview

Table I provides a non-exhaustive overview of existing, new, experimental, and planned features of P4TG. The following sections introduce the new and experimental features in more detail. New features are implemented and merged into the stable branch of P4TG. Experimental features are implemented but need more testing until they are merged. Planned features are future work that will be included later on.

Adding features to P4TG is a threefold process. First, new packet formats require changes to packet parsing and processing in the P4 data plane. Second, the controller must be extended including configuration of the new data plane functionality and modification of the REST API. Third, the new features must be made available in the frontend by adding forms to configure them using the REST API.

With Intel's decision to open-source the Intel Tofino SDE, P4TG can now be compiled and used by everyone who has an Intel Tofino™ ASIC [7]. We provide a GitHub Continuous Integration/Continuous Deployment (CI/CD) workflow that automatically compiles P4TG for the Intel Tofino™ 1/2 on updates to the P4TG GitHub repository.

B. Enhanced Traffic Generation Capabilities

This section describes the traffic generation capabilities added to P4TG.

1) Traffic Customization: In the initial published version, P4TG supported the generation of Ethernet and IPv4 traffic with source and destination address randomization. The randomization allows the generation of multiple IP streams emulating real network traffic. P4TG is extended with *IPv6* traffic generation capabilities. For IPv6, up to 48 bits can be randomized in the source and destination addresses.

Various traffic encapsulation protocols are necessary to model the network conditions in real-world environments. *VLAN* is an encapsulation that adds a VLAN tag to an Ethernet frame. The use cases for VLAN tags are manifold including segmentation of physical networks into multiple logical domains, traffic engineering, QoS, and traffic prioritization. Additionally, with *QinQ* two VLAN headers are added to an Ethernet frame allowing for more sophisticated traffic engineering using Layer 2 (L2) tunneling, and a larger VLAN ID space. Further, *VxLAN* encapsulates Ethernet frames within a UDP datagram to provide a L2 overlay in a L3 network. *MPLS* is a broadly established protocol in wide-area networking that enables high-performance forwarding. Stacked MPLS labels enable tunneling, or encoding explicit paths leveraging the source routing paradigm in SR-MPLS. With the recent advance to segment routing (SR), technologies such as SR-MPLS and SRv6 have become more important. For SRv6, a routing extension header is added to the IPv6 base header. This routing extension header encodes an explicit path and contains a list of segments, i.e., IPv6 addresses that must be traversed.

P4TG supports the encapsulation of generated traffic with VLAN and QinQ headers, up to 15 MPLS label stack entries (LSEs), and up to three segments in SRv6[1]. Additionally, the already encapsulated traffic can be further encapsulated by a VxLAN header[2]. The header fields of the encapsulation protocols are fully customizable by the control plane during runtime. By customizing the content of MPLS LSEs or SRv6 SIDs, traffic with protocol extensions such as MPLS network actions (MNA) [8] and SID compression [9] can be generated. The VLAN encapsulation of P4TG was used for an evaluation in [10] and the MPLS encapsulation in [11].

2) Tofino™ 2 Support: P4TG was initially implemented for the Intel Tofino™ 1 switching ASIC. Since the publication of P4TG, the implementation has been extended to support the more powerful Intel Tofino™ 2 hardware. P4TG running on Intel Tofino™ 2 hardware supports traffic generation of up to $4\,\mathrm{Tb/s}$ with $400\,\mathrm{Gb/s}$ per port. Furthermore, the extended pipeline size of the Intel Tofino™ 2 compared to the Intel Tofino™ 1 enables more sophisticated traffic encapsulation, such as SRv6. The generation accuracy is evaluated in Section IV.

3) Improved Control Plane: The control plane is the middleware between the data plane of P4TG and a web-based user interface for configuration. Using the data plane API, the control plane of P4TG configures the Intel Tofino™ hardware, aggregates measurement data, and exposes configuration and statistic endpoints through a REST API. Initially, the control plane was implemented in Python. To make the control plane more robust, it is entirely rewritten in the Rust programming language. Rust is an up-to-date, fast, and memory-efficient programming language. During the control plane rewrite of P4TG, the `rbfrt` library was developed [12]. The `rbfrt` library provides bindings for the gRPC interface of the Intel Tofino™ switching ASIC to configure the data plane using the Rust language.

C. Improved User Experience

This section describes features improving the user experience of P4TG.

1) Frontend: P4TG provides a web-based frontend written in ReactJS to facilitate the monitoring and configuration of measured and generated traffic. The frontend is extended with a live visualization of generated and received traffic rates, packet loss and RTT measurement, and frame size distribution. The visualized data over time can be exported using the REST API for better usability. Additionally, the configuration of packet encapsulation as described in Section III-B1 is available

[1]SRv6 is only supported on the Intel Tofino™ 2.

[2]Not in combination with SRv6 encapsulation.

in the frontend. Further, a dark mode and localization are added for better accessibility. After measurement, a report file about the analyzed traffic can be exported in PDF or CSV format.

2) Automated ARP Replies: In a typical network, the ARP table of a device under test (DUT) is filled by ARP requests and replies. Traffic is only forwarded if the respective ARP table entry is available. However, P4TG did not implement the handling of ARP requests, and the ARP table of a DUT had to be filled manually. To facilitate this, P4TG automatically responds with a pre-configured MAC address to ARP requests. This feature can be configured on a per-port basis.

3) Automated Testing: The P4TG REST API provides endpoints to automate network testing by configuring, starting, and stopping the traffic generation. When running multiple individual network tests in sequence, e.g., multiple tests with different stream descriptions, the developer must script those tests by explicitly calling the `start` and `stop` REST API endpoints for each test. To facilitate automated network testing, the P4TG REST API is extended to accept a configuration object that defines multiple tests. Each test consists of a stream description and a test duration. As a result, the developer can predefine multiple tests with different stream descriptions and test durations and send them all at once in a single REST API call to P4TG. The P4TG controller automatically executes the tests and provides statistics for each test in the `statistics` REST API endpoint.

4) Test Profiles: Traffic profiles such as the Internet mix (IMIX) and benchmarking suites such as defined in RFC 2544 [13] are standards for network testing. IMIX is a traffic profile that contains a distribution of different frame sizes, reflecting real-world conditions. RFC 2544 provides a benchmarking suite for network testing with various metrics including throughput and latency benchmarking.

Frame size distributions such as IMIX, or tests for the RFC 2544 benchmarking suite can be applied by configuring P4TG accordingly. However, the IMIX distribution and the individual RFC 2544 tests must be configured manually by the developer. Therefore, P4TG is extended with a set of predefined test profiles. Currently, the test profiles contain an IMIX and an RFC 2544 profile. The IMIX profile comprises a predefined stream description based on the IMIX frame size distribution. The RFC 2544 profile supports throughput and latency benchmarking, as well as frame loss rate and reset time measurements with different frame sizes. A developer can select a predefined test profile in the UI to test a DUT, e.g., to automatically apply the RFC 2544 benchmarking suite. Additional profiles may be added in the future.

D. Future Work

In its current state, traffic generation is configured and statistics are retrieved using the HTTP REST API of P4TG. However, network operators typically configure their devices using the state-of-the-art NETCONF protocol for network management. The NETCONF protocol leverages the YANG data modeling language to manage devices with a well-defined data model. In the future, we want to implement a NETCONF API using YANG models to complement the REST API for ease of network configuration. This will allow operators to integrate P4TG seamlessly into their existing network environment.

Automated ARP replies facilitate traffic generation in IPv4 networks. However, for IPv6 traffic, NDP is required which complements ARP handling. In future work, automated NDP handling will be added to P4TG.

TABLE II
MAXIMUM TRAFFIC GENERATION RATE OF P4TG PER PORT ON TOFINO™ 1 AND 2 FOR DIFFERENT FRAME SIZES.

Frame size	Tofino 1	Tofino 2
64 B	99.37 Gb/s	294.00 Gb/s
128 B	98.56 Gb/s	388.50 Gb/s
256 B	99.77 Gb/s	396.73 Gb/s
512 B	99.55 Gb/s	399.83 Gb/s
1024 B	99.41 Gb/s	399.72 Gb/s
1518 B	99.62 Gb/s	398.47 Gb/s

IV. GENERATION ACCURACY OF P4TG

We evaluate the traffic generation accuracy of P4TG. We generate the maximum possible target rate per port, i.e., 100 Gb/s for Tofino™ 1 and 400 Gb/s for Tofino™ 2. We loop the traffic back to an ingress port to measure the achieved L1 rate. Table II compiles the results.

P4TG achieves almost line rate for 100 Gb/s on Tofino™ 1 for any frame size. However, for a target rate of 400 Gb/s on Tofino™ 2, the desired traffic rate is approximatly obtained only for frame sizes of 256 B or larger. When large encapsulation headers like SRv6 (48 B plus 16 B per segment) or VxLAN (50 B) are utilized, frames tend to be larger, which diminishes the relevance of the reported problem.

REFERENCES

[1] TRex Team. TRex – Realistic Traffic Generator. https://trex-tgn.cisco.com/, visited on 2025-01-17.
[2] P. Emmerich et al. Mind the Gap - A Comparison of Software Packet Generators. In *ANCS*, pp. 191–203, 2017.
[3] F. G. Costa et al. PIPO-TG: Parameterizable High-Performance Traffic Generation. In *NOMS*, pp. 1–9, 2024.
[4] S. Lindner et al. P4TG: 1 Tb/s Traffic Generation for Ethernet/IP Networks. *IEEE Access*, 11:17525–17535, February 2023.
[5] S. Lindner et al. GitHub: P4TG. visited on 2025-02-12.
[6] Intel. Intel Tofino Native Architecture—Public Version. https://github.com/barefootnetworks/Open-Tofino.
[7] P4.org. Intel's Tofino P4 Software is Now Open Source. https://p4.org/intels-tofino-p4-software-is-now-open-source//, visited on 2025-01-27.
[8] L. Andersson et al. MPLS Network Actions (MNA) Framework. Internet-Draft mpls-mna-fwk-15, IETF, December 2024. WiP.
[9] W. Cheng et al. Compressed SRv6 Segment List Encoding (CSID). Internet-Draft spring-srv6-srh-compression-23, IETF, 2025. WiP.
[10] F. Ihle et al. P4-PSFP: P4-Based Per-Stream Filtering and Policing for Time-Sensitive Networking. *TNSM – Special Issue on Robust and Resilient Future Communication Networks*, 21(5):5273 – 5290, 2024.
[11] F. Ihle et al. MPLS Network Actions: Technological Overview and P4-Based Implementation on a High-Speed Switching ASIC. *ArxIv Preprint*, October 2024.
[12] E. Zink et al. Rust Barefoot Runtime (RBFRT): Fast Runtime Control for the Intel Tofino. In *ArxIv Preprint*, January 2025.
[13] S. Bradner et al. Benchmarking Methodology for Network Interconnect Devices. RFC 2544, IETF, March 1999.